

OpenBLT Bootloader - Reference Manual

1.12.1

Generated by Doxygen 1.8.13

Contents

1	OpenBLT Firmware Documentation	1
1.1	Introduction	1
1.2	Software Architecture	1
1.3	Copyright and Licensing	2
2	Module Index	3
2.1	Modules	3
3	Data Structure Index	9
3.1	Data Structures	9
4	File Index	11
4.1	File List	11
5	Module Documentation	61
5.1	Bootloader	61
5.1.1	Detailed Description	61
5.2	Template for demo programs	62
5.2.1	Detailed Description	62
5.3	User Program	63
5.3.1	Detailed Description	63
5.4	Bootloader	64
5.5	Demo for S32K118EVB/GCC	65
5.5.1	Detailed Description	65
5.6	User Program	66

5.7	Bootloader	67
5.8	Demo for S32K118EVB/IAR	68
5.8.1	Detailed Description	68
5.9	User Program	69
5.10	Bootloader	70
5.10.1	Detailed Description	70
5.11	Demo for STM32F0-Discovery/STM32CubeIDE	71
5.11.1	Detailed Description	71
5.12	User Program	72
5.12.1	Detailed Description	72
5.13	Bootloader	73
5.13.1	Detailed Description	73
5.14	Demo for STM32F0-Discovery/GCC	74
5.14.1	Detailed Description	74
5.15	User Program	75
5.15.1	Detailed Description	75
5.16	Bootloader	76
5.16.1	Detailed Description	76
5.17	Demo for STM32F0-Discovery/IAR	77
5.17.1	Detailed Description	77
5.18	User Program	78
5.18.1	Detailed Description	78
5.19	Bootloader	79
5.19.1	Detailed Description	79
5.20	Demo for STM32F0-Discovery/Keil	80
5.20.1	Detailed Description	80
5.21	User Program	81
5.21.1	Detailed Description	81
5.22	Bootloader	82
5.22.1	Detailed Description	82

5.23 Demo for Nucleo-F091RC/STM32CubeIDE	83
5.23.1 Detailed Description	83
5.24 User Program	84
5.24.1 Detailed Description	84
5.25 Bootloader	85
5.25.1 Detailed Description	85
5.26 Demo for Nucleo-F091RC/GCC	86
5.26.1 Detailed Description	86
5.27 User Program	87
5.27.1 Detailed Description	87
5.28 Bootloader	88
5.28.1 Detailed Description	88
5.29 Demo for Nucleo-F091RC/IAR	89
5.29.1 Detailed Description	89
5.30 User Program	90
5.30.1 Detailed Description	90
5.31 Bootloader	91
5.31.1 Detailed Description	91
5.32 Demo for Nucleo-F091RC/Keil	92
5.32.1 Detailed Description	92
5.33 User Program	93
5.33.1 Detailed Description	93
5.34 Bootloader	94
5.34.1 Detailed Description	94
5.35 Demo for Nucleo-G071RB/STM32CubeIDE	95
5.35.1 Detailed Description	95
5.36 User Program	96
5.36.1 Detailed Description	96
5.37 Bootloader	97
5.37.1 Detailed Description	97

5.38 Demo for Nucleo-G071RB/GCC	98
5.38.1 Detailed Description	98
5.39 User Program	99
5.39.1 Detailed Description	99
5.40 Bootloader	100
5.40.1 Detailed Description	100
5.41 Demo for Nucleo-G071RB/IAR	101
5.41.1 Detailed Description	101
5.42 User Program	102
5.42.1 Detailed Description	102
5.43 Bootloader	103
5.43.1 Detailed Description	103
5.44 Demo for Nucleo-G071RB/Keil	104
5.44.1 Detailed Description	104
5.45 User Program	105
5.45.1 Detailed Description	105
5.46 Bootloader	106
5.46.1 Detailed Description	106
5.47 Demo for XMC1400 Boot Kit/GCC	107
5.47.1 Detailed Description	107
5.48 User Program	108
5.48.1 Detailed Description	108
5.49 Bootloader	109
5.49.1 Detailed Description	109
5.50 Demo for XMC1400 Boot Kit/IAR	110
5.50.1 Detailed Description	110
5.51 User Program	111
5.51.1 Detailed Description	111
5.52 Bootloader	112
5.52.1 Detailed Description	112

5.53 Demo for Nucleo-L552ZE/STM32CubeIDE	113
5.53.1 Detailed Description	113
5.54 User Program	114
5.54.1 Detailed Description	114
5.55 Bootloader	115
5.55.1 Detailed Description	115
5.56 Demo for Nucleo-L552ZE/GCC	116
5.56.1 Detailed Description	116
5.57 User Program	117
5.57.1 Detailed Description	117
5.58 Bootloader	118
5.58.1 Detailed Description	118
5.59 Demo for Nucleo-L552ZE/IAR	119
5.59.1 Detailed Description	119
5.60 User Program	120
5.60.1 Detailed Description	120
5.61 Bootloader	121
5.61.1 Detailed Description	121
5.62 Demo for Nucleo-L552ZE/Keil	122
5.62.1 Detailed Description	122
5.63 User Program	123
5.63.1 Detailed Description	123
5.64 Bootloader	124
5.64.1 Detailed Description	124
5.65 Demo for Olimex EM-32G880F128-STK/GCC	125
5.65.1 Detailed Description	125
5.66 User Program	126
5.66.1 Detailed Description	126
5.67 Bootloader	127
5.67.1 Detailed Description	127

5.68 Demo for Olimex EM-32G880F128-STK/IAR	128
5.68.1 Detailed Description	128
5.69 User Program	129
5.69.1 Detailed Description	129
5.70 Bootloader	130
5.70.1 Detailed Description	130
5.71 Demo for Texas Instruments EK-LM3S6965/GCC	131
5.71.1 Detailed Description	131
5.72 User Program	132
5.72.1 Detailed Description	132
5.73 Bootloader	133
5.73.1 Detailed Description	133
5.74 Demo for Texas Instruments EK-LM3S6965/IAR	134
5.74.1 Detailed Description	134
5.75 User Program	135
5.75.1 Detailed Description	135
5.76 Bootloader	136
5.76.1 Detailed Description	136
5.77 Demo for Texas Instruments EK-LM3S8962/GCC	137
5.77.1 Detailed Description	137
5.78 User Program	138
5.78.1 Detailed Description	138
5.79 Bootloader	139
5.79.1 Detailed Description	139
5.80 Demo for Texas Instruments EK-LM3S8962/IAR	140
5.80.1 Detailed Description	140
5.81 User Program	141
5.81.1 Detailed Description	141
5.82 Bootloader	142
5.82.1 Detailed Description	142

5.83 Demo for Nucleo-F103RB/STM32CubeIDE	143
5.83.1 Detailed Description	143
5.84 User Program	144
5.84.1 Detailed Description	144
5.85 Bootloader	145
5.85.1 Detailed Description	145
5.86 Demo for Nucleo-F103RB/GCC	146
5.86.1 Detailed Description	146
5.87 User Program	147
5.87.1 Detailed Description	147
5.88 Bootloader	148
5.88.1 Detailed Description	148
5.89 Demo for Nucleo-F103RB	149
5.89.1 Detailed Description	149
5.90 User Program	150
5.90.1 Detailed Description	150
5.91 Bootloader	151
5.91.1 Detailed Description	151
5.92 Demo for Nucleo-F103RB/Keil	152
5.92.1 Detailed Description	152
5.93 User Program	153
5.93.1 Detailed Description	153
5.94 Bootloader	154
5.94.1 Detailed Description	154
5.95 Demo for Olimex STM32-H103/STM32CubeIDE	155
5.95.1 Detailed Description	155
5.96 User Program	156
5.96.1 Detailed Description	156
5.97 Bootloader	157
5.97.1 Detailed Description	157

5.98 Demo for Olimex STM32-H103/GCC	158
5.98.1 Detailed Description	158
5.99 User Program	159
5.99.1 Detailed Description	159
5.100 Bootloader	160
5.100.1 Detailed Description	160
5.101 Demo for Olimex STM32-H103/IAR	161
5.101.1 Detailed Description	161
5.102 User Program	162
5.102.1 Detailed Description	162
5.103 Bootloader	163
5.103.1 Detailed Description	163
5.104 Demo for Olimex STM32-H103/Keil	164
5.104.1 Detailed Description	164
5.105 User Program	165
5.105.1 Detailed Description	165
5.106 Bootloader	166
5.106.1 Detailed Description	166
5.107 Demo for Olimex STM32-P103/STM32CubeIDE	167
5.107.1 Detailed Description	167
5.108 User Program	168
5.108.1 Detailed Description	168
5.109 Bootloader	169
5.109.1 Detailed Description	169
5.110 Demo for Olimex STM32-P103/GCC	170
5.110.1 Detailed Description	170
5.111 User Program	171
5.111.1 Detailed Description	171
5.112 Bootloader	172
5.112.1 Detailed Description	172

5.113 Demo for Olimex STM32-P103/IAR	173
5.113.1 Detailed Description	173
5.114 User Program	174
5.114.1 Detailed Description	174
5.115 Bootloader	175
5.115.1 Detailed Description	175
5.116 Demo for Olimex STM32-P103/Keil	176
5.116.1 Detailed Description	176
5.117 User Program	177
5.117.1 Detailed Description	177
5.118 Bootloader	178
5.118.1 Detailed Description	178
5.119 Demo for Olimexino-STM32/STM32CubeIDE	179
5.119.1 Detailed Description	179
5.120 User Program	180
5.120.1 Detailed Description	180
5.121 Bootloader	181
5.121.1 Detailed Description	181
5.122 Demo for Olimexino-STM32/GCC	182
5.122.1 Detailed Description	182
5.123 User Program	183
5.123.1 Detailed Description	183
5.124 Bootloader	184
5.124.1 Detailed Description	184
5.125 Demo for Olimexino-STM32/IAR	185
5.125.1 Detailed Description	185
5.126 User Program	186
5.126.1 Detailed Description	186
5.127 Bootloader	187
5.127.1 Detailed Description	187

5.128 Demo for Olimexino-STM32/Keil	188
5.128.1 Detailed Description	188
5.129 User Program	189
5.129.1 Detailed Description	189
5.130 Bootloader	190
5.130.1 Detailed Description	190
5.131 Demo for Olimex STM32-P207/STM32CubeIDE	191
5.131.1 Detailed Description	191
5.132 User Program	192
5.132.1 Detailed Description	192
5.133 Bootloader	193
5.133.1 Detailed Description	193
5.134 Demo for Olimex STM32-P207/GCC	194
5.134.1 Detailed Description	194
5.135 User Program	195
5.135.1 Detailed Description	195
5.136 Bootloader	196
5.136.1 Detailed Description	196
5.137 Demo for Olimex STM32-P207/IAR	197
5.137.1 Detailed Description	197
5.138 User Program	198
5.138.1 Detailed Description	198
5.139 Bootloader	199
5.139.1 Detailed Description	199
5.140 Demo for Olimex STM32-P207/Keil	200
5.140.1 Detailed Description	200
5.141 User Program	201
5.141.1 Detailed Description	201
5.142 Bootloader	202
5.142.1 Detailed Description	202

5.143 Demo for S32K144EVB/GCC	203
5.143.1 Detailed Description	203
5.144 User Program	204
5.144.1 Detailed Description	204
5.145 Bootloader	205
5.145.1 Detailed Description	205
5.146 Demo for S32K144EVB/IAR	206
5.146.1 Detailed Description	206
5.147 User Program	207
5.147.1 Detailed Description	207
5.148 Bootloader	208
5.148.1 Detailed Description	208
5.149 Demo for STM32F3-Discovery/STM32CubeIDE	209
5.149.1 Detailed Description	209
5.150 User Program	210
5.150.1 Detailed Description	210
5.151 Bootloader	211
5.151.1 Detailed Description	211
5.152 Demo for STM32F3-Discovery/GCC	212
5.152.1 Detailed Description	212
5.153 User Program	213
5.153.1 Detailed Description	213
5.154 Bootloader	214
5.154.1 Detailed Description	214
5.155 Demo for STM32F3-Discovery/IAR	215
5.155.1 Detailed Description	215
5.156 User Program	216
5.156.1 Detailed Description	216
5.157 Bootloader	217
5.157.1 Detailed Description	217

5.158 Demo for STM32F3-Discovery/Keil	218
5.158.1 Detailed Description	218
5.159 User Program	219
5.159.1 Detailed Description	219
5.160 Bootloader	220
5.160.1 Detailed Description	220
5.161 Demo for Nucleo-F303K8/STM32CubeIDE	221
5.161.1 Detailed Description	221
5.162 User Program	222
5.162.1 Detailed Description	222
5.163 Bootloader	223
5.163.1 Detailed Description	223
5.164 Demo for Nucleo-F303K8/GCC	224
5.164.1 Detailed Description	224
5.165 User Program	225
5.165.1 Detailed Description	225
5.166 Bootloader	226
5.166.1 Detailed Description	226
5.167 Demo for Nucleo-F303K8/IAR	227
5.167.1 Detailed Description	227
5.168 User Program	228
5.168.1 Detailed Description	228
5.169 Bootloader	229
5.169.1 Detailed Description	229
5.170 Demo for Nucleo-F303K8/Keil	230
5.170.1 Detailed Description	230
5.171 User Program	231
5.171.1 Detailed Description	231
5.172 Bootloader	232
5.172.1 Detailed Description	232

5.173 Demo for Nucleo-F429ZI/STM32CubeIDE	233
5.173.1 Detailed Description	233
5.174 User Program	234
5.174.1 Detailed Description	234
5.175 Bootloader	235
5.175.1 Detailed Description	235
5.176 Demo for Nucleo-F429ZI/GCC	236
5.176.1 Detailed Description	236
5.177 User Program	237
5.177.1 Detailed Description	237
5.178 Bootloader	238
5.178.1 Detailed Description	238
5.179 Demo for Nucleo-F429ZI/IAR	239
5.179.1 Detailed Description	239
5.180 User Program	240
5.180.1 Detailed Description	240
5.181 Bootloader	241
5.181.1 Detailed Description	241
5.182 Demo for Nucleo-F429ZI/Keil	242
5.182.1 Detailed Description	242
5.183 User Program	243
5.183.1 Detailed Description	243
5.184 Bootloader	244
5.184.1 Detailed Description	244
5.185 Demo for Olimex STM32-P405/STM32CubeIDE	245
5.185.1 Detailed Description	245
5.186 User Program	246
5.186.1 Detailed Description	246
5.187 Bootloader	247
5.187.1 Detailed Description	247

5.188 Demo for Olimex STM32-P405/GCC	248
5.188.1 Detailed Description	248
5.189 User Program	249
5.189.1 Detailed Description	249
5.190 Bootloader	250
5.190.1 Detailed Description	250
5.191 Demo for Olimex STM32-P405/IAR	251
5.191.1 Detailed Description	251
5.192 User Program	252
5.192.1 Detailed Description	252
5.193 Bootloader	253
5.193.1 Detailed Description	253
5.194 Demo for Olimex STM32-P405/Keil	254
5.194.1 Detailed Description	254
5.195 User Program	255
5.195.1 Detailed Description	255
5.196 Bootloader	256
5.196.1 Detailed Description	256
5.197 Demo for Nucleo-L476RG/STM32CubeIDE	257
5.197.1 Detailed Description	257
5.198 User Program	258
5.198.1 Detailed Description	258
5.199 Bootloader	259
5.199.1 Detailed Description	259
5.200 Demo for Nucleo-L476RG/GCC	260
5.200.1 Detailed Description	260
5.201 User Program	261
5.201.1 Detailed Description	261
5.202 Bootloader	262
5.202.1 Detailed Description	262

5.203 Demo for Nucleo-L476RG/IAR	263
5.203.1 Detailed Description	263
5.204 User Program	264
5.204.1 Detailed Description	264
5.205 Bootloader	265
5.205.1 Detailed Description	265
5.206 Demo for Nucleo-L476RG/Keil	266
5.206.1 Detailed Description	266
5.207 User Program	267
5.207.1 Detailed Description	267
5.208 Bootloader	268
5.208.1 Detailed Description	268
5.209 Demo for Texas Instruments DK-TM4C123G/IAR	269
5.209.1 Detailed Description	269
5.210 User Program	270
5.210.1 Detailed Description	270
5.211 Bootloader	271
5.211.1 Detailed Description	271
5.212 Demo for XMC4700 Relax Kit/GCC	272
5.212.1 Detailed Description	272
5.213 User Program	273
5.213.1 Detailed Description	273
5.214 Bootloader	274
5.214.1 Detailed Description	274
5.215 Demo for XMC4700 Relax Kit/IAR	275
5.215.1 Detailed Description	275
5.216 User Program	276
5.216.1 Detailed Description	276
5.217 Bootloader	277
5.217.1 Detailed Description	277

5.218 Demo for Nucleo-F746ZG/STM32CubeIDE	278
5.218.1 Detailed Description	278
5.219 User Program	279
5.219.1 Detailed Description	279
5.220 Bootloader	280
5.220.1 Detailed Description	280
5.221 Demo for Nucleo-F746ZG/GCC	281
5.221.1 Detailed Description	281
5.222 User Program	282
5.222.1 Detailed Description	282
5.223 Bootloader	283
5.223.1 Detailed Description	283
5.224 Demo for Nucleo-F746ZG/IAR	284
5.224.1 Detailed Description	284
5.225 User Program	285
5.225.1 Detailed Description	285
5.226 Bootloader	286
5.226.1 Detailed Description	286
5.227 Demo for Nucleo-F746ZG/Keil	287
5.227.1 Detailed Description	287
5.228 User Program	288
5.228.1 Detailed Description	288
5.229 Bootloader	289
5.229.1 Detailed Description	289
5.230 Demo for Nucleo-F767ZI/STM32CubeIDE	290
5.230.1 Detailed Description	290
5.231 User Program	291
5.231.1 Detailed Description	291
5.232 Bootloader	292
5.232.1 Detailed Description	292

5.233 Demo for Nucleo-F767ZI/GCC	293
5.233.1 Detailed Description	293
5.234 User Program	294
5.234.1 Detailed Description	294
5.235 Bootloader	295
5.235.1 Detailed Description	295
5.236 Demo for Nucleo-F767ZI/IAR	296
5.236.1 Detailed Description	296
5.237 User Program	297
5.237.1 Detailed Description	297
5.238 Bootloader	298
5.238.1 Detailed Description	298
5.239 Demo for Nucleo-F767ZI/Keil	299
5.239.1 Detailed Description	299
5.240 User Program	300
5.240.1 Detailed Description	300
5.241 Bootloader	301
5.241.1 Detailed Description	301
5.242 Demo for Nucleo-H743ZI/STM32CubeIDE	302
5.242.1 Detailed Description	302
5.243 User Program	303
5.243.1 Detailed Description	303
5.244 Bootloader	304
5.244.1 Detailed Description	304
5.245 Demo for Nucleo-H743ZI/GCC	305
5.245.1 Detailed Description	305
5.246 User Program	306
5.246.1 Detailed Description	306
5.247 Bootloader	307
5.247.1 Detailed Description	307

5.248 Demo for Nucleo-H743ZI/IAR	308
5.248.1 Detailed Description	308
5.249 User Program	309
5.249.1 Detailed Description	309
5.250 Bootloader	310
5.250.1 Detailed Description	310
5.251 Demo for Nucleo-H743ZI/Keil	311
5.251.1 Detailed Description	311
5.252 User Program	312
5.252.1 Detailed Description	312
5.253 Bootloader Demos	313
5.253.1 Detailed Description	316
5.254 Bootloader	317
5.254.1 Detailed Description	317
5.255 Demo for NXP DevKit-S12G128/CodeWarrior	318
5.255.1 Detailed Description	318
5.256 User Program	319
5.256.1 Detailed Description	319
5.257 Bootloader	320
5.257.1 Detailed Description	320
5.258 Demo for Dragon12-plus/CodeWarrior	321
5.258.1 Detailed Description	321
5.259 User Program	322
5.259.1 Detailed Description	322
5.260 CAN driver of a port	323
5.260.1 Detailed Description	323
5.261 CPU driver of a port	324
5.261.1 Detailed Description	324
5.262 Flash driver of a port	325
5.262.1 Detailed Description	325

5.263	Compiler specifics of a port	326
5.263.1	Detailed Description	326
5.264	Non-volatile memory driver of a port	327
5.264.1	Detailed Description	327
5.265	RS232 UART driver of a port	328
5.265.1	Detailed Description	328
5.266	Target Port Template	329
5.266.1	Detailed Description	329
5.267	Timer driver of a port	330
5.267.1	Detailed Description	330
5.268	Type definitions of a port	331
5.268.1	Detailed Description	331
5.269	USB driver of a port	332
5.269.1	Detailed Description	332
5.270	Target ARMCM0 S32K11	333
5.270.1	Detailed Description	333
5.271	Target ARMCM0 STM32F0	334
5.271.1	Detailed Description	334
5.272	Target ARMCM0 STM32G0	335
5.272.1	Detailed Description	335
5.273	Target ARMCM0 XMC1	336
5.273.1	Detailed Description	336
5.274	Target ARMCM33 STM32L5	337
5.274.1	Detailed Description	337
5.275	Target ARMCM3 EFM32	338
5.275.1	Detailed Description	338
5.276	Target ARMCM3 LM3S	339
5.276.1	Detailed Description	339
5.277	Target ARMCM3 STM32F1	340
5.277.1	Detailed Description	340

5.278	Target ARMCM3 STM32F2	341
5.278.1	Detailed Description	341
5.279	Target ARMCM4 S32K14	342
5.279.1	Detailed Description	342
5.280	Target ARMCM4 STM32F3	343
5.280.1	Detailed Description	343
5.281	Target ARMCM4 STM32F4	344
5.281.1	Detailed Description	344
5.282	Target ARMCM4 STM32L4	345
5.282.1	Detailed Description	345
5.283	Target ARMCM4 TM4C	346
5.283.1	Detailed Description	346
5.284	Target ARMCM4 XMC4	347
5.284.1	Detailed Description	347
5.285	Target ARMCM7 STM32F7	348
5.285.1	Detailed Description	348
5.286	Target ARMCM7 STM32H7	349
5.286.1	Detailed Description	349
5.287	Bootloader Core	350
5.287.1	Detailed Description	351
5.288	Target HCS12	352
5.288.1	Detailed Description	352
5.289	Bootloader Ports	353
5.289.1	Detailed Description	353

6	Data Structure Documentation	355
6.1	tCanBusTiming Struct Reference	355
6.1.1	Detailed Description	355
6.1.2	Field Documentation	355
6.1.2.1	phaseSeg1 [1/2]	355
6.1.2.2	phaseSeg1 [2/2]	356
6.1.2.3	phaseSeg2 [1/2]	356
6.1.2.4	phaseSeg2 [2/2]	356
6.1.2.5	propSeg [1/2]	356
6.1.2.6	propSeg [2/2]	356
6.1.2.7	timeQuanta [1/2]	356
6.1.2.8	timeQuanta [2/2]	357
6.1.2.9	tseg1 [1/2]	357
6.1.2.10	tseg1 [2/2]	357
6.1.2.11	tseg2 [1/2]	357
6.1.2.12	tseg2 [2/2]	357
6.2	tCanRegs Struct Reference	358
6.2.1	Detailed Description	359
6.2.2	Field Documentation	359
6.2.2.1	cbtr0	359
6.2.2.2	cbtr1	359
6.2.2.3	cctl0	359
6.2.2.4	cctl1	359
6.2.2.5	cidac	359
6.2.2.6	cidar0	359
6.2.2.7	cidar1	360
6.2.2.8	cidar2	360
6.2.2.9	cidar3	360
6.2.2.10	cidar4	360
6.2.2.11	cidar5	360

6.2.2.12	cidar6	360
6.2.2.13	cidar7	360
6.2.2.14	cidmr0	360
6.2.2.15	cidmr1	361
6.2.2.16	cidmr2	361
6.2.2.17	cidmr3	361
6.2.2.18	cidmr4	361
6.2.2.19	cidmr5	361
6.2.2.20	cidmr6	361
6.2.2.21	cidmr7	361
6.2.2.22	crflg	361
6.2.2.23	crier	362
6.2.2.24	crxerr	362
6.2.2.25	ctaak	362
6.2.2.26	ctarq	362
6.2.2.27	ctbsel	362
6.2.2.28	ctflg	362
6.2.2.29	ctier	362
6.2.2.30	ctxerr	362
6.2.2.31	dummy1	363
6.2.2.32	rxSlot	363
6.2.2.33	txSlot	363
6.3	tCanRxMsgSlot Struct Reference	363
6.3.1	Detailed Description	363
6.3.2	Field Documentation	363
6.3.2.1	dIrr	364
6.3.2.2	dSr	364
6.3.2.3	dummy	364
6.3.2.4	iDr	364
6.3.2.5	tStamp	364

6.4	tCanTxMsgSlot Struct Reference	364
6.4.1	Detailed Description	365
6.4.2	Field Documentation	365
6.4.2.1	dlr	365
6.4.2.2	dsr	365
6.4.2.3	idr	365
6.4.2.4	tbpr	365
6.4.2.5	tstamp	365
6.5	tFatFsObjects Struct Reference	365
6.5.1	Detailed Description	366
6.5.2	Field Documentation	366
6.5.2.1	file	366
6.5.2.2	fs	366
6.6	tFifoCtrl Struct Reference	366
6.6.1	Detailed Description	367
6.6.2	Field Documentation	367
6.6.2.1	endptr	367
6.6.2.2	entries	367
6.6.2.3	fifoctrlptr	367
6.6.2.4	handle	367
6.6.2.5	length	368
6.6.2.6	readptr	368
6.6.2.7	startptr	368
6.6.2.8	writeptr	368
6.7	tFifoPipe Struct Reference	368
6.7.1	Detailed Description	369
6.7.2	Field Documentation	369
6.7.2.1	data	369
6.7.2.2	handle	369
6.8	tFileEraseInfo Struct Reference	369

6.8.1	Detailed Description	370
6.8.2	Field Documentation	370
6.8.2.1	start_address	370
6.8.2.2	total_size	370
6.9	tFlashBlockInfo Struct Reference	370
6.9.1	Detailed Description	370
6.10	tFlashPrescalerSysclockMapping Struct Reference	371
6.10.1	Detailed Description	371
6.10.2	Field Documentation	371
6.10.2.1	prescaler	371
6.10.2.2	sysclock_max	371
6.10.2.3	sysclock_min	371
6.11	tFlashRegs Struct Reference	372
6.11.1	Detailed Description	372
6.11.2	Field Documentation	372
6.11.2.1	dfprot	372
6.11.2.2	fccob	372
6.11.2.3	fccobix	373
6.11.2.4	fclkdiv	373
6.11.2.5	fcmd	373
6.11.2.6	fcnfg	373
6.11.2.7	fercnfg	373
6.11.2.8	ferstat	373
6.11.2.9	fopt	373
6.11.2.10	fprot	374
6.11.2.11	frsv0	374
6.11.2.12	frsv1	374
6.11.2.13	frsv2	374
6.11.2.14	frsv3	374
6.11.2.15	frsv4	374

6.11.2.16 frsv5	374
6.11.2.17 frsv6	375
6.11.2.18 frsv7	375
6.11.2.19 fsec	375
6.11.2.20 fstat	375
6.11.2.21 fstmod	375
6.12 tFlashSector Struct Reference	375
6.12.1 Detailed Description	376
6.12.2 Field Documentation	376
6.12.2.1 bank_num	376
6.12.2.2 sector_num	376
6.12.2.3 sector_size	376
6.12.2.4 sector_start	377
6.13 tIsrFunc Union Reference	377
6.13.1 Detailed Description	377
6.13.2 Field Documentation	377
6.13.2.1 func	377
6.13.2.2 ptr [1/3]	377
6.13.2.3 ptr [2/3]	378
6.13.2.4 ptr [3/3]	378
6.14 tSharedParamsBuffer Struct Reference	378
6.14.1 Detailed Description	378
6.15 tSrecLineParseObject Struct Reference	378
6.15.1 Detailed Description	379
6.15.2 Field Documentation	379
6.15.2.1 address	379
6.15.2.2 data	379
6.15.2.3 line	379
6.16 tSysTickRegs Struct Reference	380
6.16.1 Detailed Description	380

6.16.2	Field Documentation	380
6.16.2.1	CTRL	380
6.16.2.2	LOAD	380
6.16.2.3	VAL	380
6.17	tTimerRegs Struct Reference	380
6.17.1	Detailed Description	381
6.17.2	Field Documentation	381
6.17.2.1	cforc	381
6.17.2.2	oc7d	381
6.17.2.3	oc7m	381
6.17.2.4	tc	382
6.17.2.5	tcnt	382
6.17.2.6	tctl1	382
6.17.2.7	tctl2	382
6.17.2.8	tctl3	382
6.17.2.9	tctl4	382
6.17.2.10	tflg1	382
6.17.2.11	tflg2	382
6.17.2.12	tie	383
6.17.2.13	tios	383
6.17.2.14	tscr1	383
6.17.2.15	tscr2	383
6.17.2.16	ttov	383
6.18	tXcpInfo Struct Reference	383
6.18.1	Detailed Description	384
6.18.2	Field Documentation	384
6.18.2.1	connected	384
6.18.2.2	ctoData	384
6.18.2.3	ctoLen	384
6.18.2.4	ctoPending	385
6.18.2.5	mta	385
6.18.2.6	protection	385
6.18.2.7	s_n_k_resource	385
6.19	uip_tcp_appstate_t Struct Reference	385
6.19.1	Detailed Description	385

7	File Documentation	387
7.1	app.c File Reference	387
7.1.1	Detailed Description	387
7.1.2	Function Documentation	388
7.1.2.1	AppInit()	388
7.1.2.2	AppTask()	388
7.2	app.c File Reference	388
7.2.1	Detailed Description	389
7.2.2	Function Documentation	389
7.2.2.1	AppInit()	389
7.2.2.2	AppTask()	389
7.3	app.c File Reference	390
7.3.1	Detailed Description	390
7.3.2	Function Documentation	390
7.3.2.1	AppInit()	390
7.3.2.2	AppTask()	391
7.4	app.c File Reference	391
7.4.1	Detailed Description	391
7.4.2	Function Documentation	391
7.4.2.1	AppInit()	392
7.4.2.2	AppTask()	392
7.5	app.c File Reference	392
7.5.1	Detailed Description	393
7.5.2	Function Documentation	393
7.5.2.1	AppInit()	393
7.5.2.2	AppTask()	393
7.6	app.c File Reference	394
7.6.1	Detailed Description	394
7.6.2	Function Documentation	394
7.6.2.1	AppInit()	394

7.6.2.2	AppTask()	395
7.7	app.c File Reference	395
7.7.1	Detailed Description	395
7.7.2	Function Documentation	395
7.7.2.1	AppInit()	396
7.7.2.2	AppTask()	396
7.8	app.c File Reference	396
7.8.1	Detailed Description	397
7.8.2	Function Documentation	397
7.8.2.1	AppInit()	397
7.8.2.2	AppTask()	397
7.9	app.c File Reference	397
7.9.1	Detailed Description	398
7.9.2	Function Documentation	398
7.9.2.1	AppInit()	398
7.9.2.2	AppTask()	398
7.10	app.c File Reference	399
7.10.1	Detailed Description	399
7.10.2	Function Documentation	399
7.10.2.1	AppInit()	399
7.10.2.2	AppTask()	400
7.11	app.c File Reference	400
7.11.1	Detailed Description	400
7.11.2	Function Documentation	400
7.11.2.1	AppInit()	401
7.11.2.2	AppTask()	401
7.12	app.c File Reference	401
7.12.1	Detailed Description	402
7.12.2	Function Documentation	402
7.12.2.1	AppInit()	402

7.12.2.2	AppTask()	402
7.13	app.c File Reference	403
7.13.1	Detailed Description	403
7.13.2	Function Documentation	403
7.13.2.1	AppInit()	403
7.13.2.2	AppTask()	404
7.14	app.c File Reference	404
7.14.1	Detailed Description	404
7.14.2	Function Documentation	404
7.14.2.1	AppInit()	405
7.14.2.2	AppTask()	405
7.15	app.c File Reference	405
7.15.1	Detailed Description	406
7.15.2	Function Documentation	406
7.15.2.1	AppInit()	406
7.15.2.2	AppTask()	406
7.16	app.c File Reference	407
7.16.1	Detailed Description	407
7.16.2	Function Documentation	407
7.16.2.1	AppInit()	407
7.16.2.2	AppTask()	408
7.17	app.c File Reference	408
7.17.1	Detailed Description	408
7.17.2	Function Documentation	408
7.17.2.1	AppInit()	409
7.17.2.2	AppTask()	409
7.18	app.c File Reference	409
7.18.1	Detailed Description	410
7.18.2	Function Documentation	410
7.18.2.1	AppInit()	410

7.18.2.2	AppTask()	410
7.19	app.c File Reference	410
7.19.1	Detailed Description	411
7.19.2	Function Documentation	411
7.19.2.1	AppInit()	411
7.19.2.2	AppTask()	411
7.20	app.c File Reference	412
7.20.1	Detailed Description	412
7.20.2	Function Documentation	412
7.20.2.1	AppInit()	412
7.20.2.2	AppTask()	413
7.21	app.c File Reference	413
7.21.1	Detailed Description	413
7.21.2	Function Documentation	413
7.21.2.1	AppInit()	414
7.21.2.2	AppTask()	414
7.22	app.c File Reference	414
7.22.1	Detailed Description	415
7.22.2	Function Documentation	415
7.22.2.1	AppInit()	415
7.22.2.2	AppTask()	415
7.23	app.c File Reference	416
7.23.1	Detailed Description	416
7.23.2	Function Documentation	416
7.23.2.1	AppInit()	416
7.23.2.2	AppTask()	417
7.24	app.c File Reference	417
7.24.1	Detailed Description	417
7.24.2	Function Documentation	417
7.24.2.1	AppInit()	418

7.24.2.2	AppTask()	418
7.25	app.c File Reference	418
7.25.1	Detailed Description	419
7.25.2	Function Documentation	419
7.25.2.1	AppInit()	419
7.25.2.2	AppTask()	419
7.26	app.c File Reference	420
7.26.1	Detailed Description	420
7.26.2	Function Documentation	420
7.26.2.1	AppInit()	420
7.26.2.2	AppTask()	421
7.27	app.c File Reference	421
7.27.1	Detailed Description	421
7.27.2	Function Documentation	421
7.27.2.1	AppInit()	422
7.27.2.2	AppTask()	422
7.28	app.c File Reference	422
7.28.1	Detailed Description	423
7.28.2	Function Documentation	423
7.28.2.1	AppInit()	423
7.28.2.2	AppTask()	423
7.29	app.c File Reference	423
7.29.1	Detailed Description	424
7.29.2	Function Documentation	424
7.29.2.1	AppInit()	424
7.29.2.2	AppTask()	424
7.30	app.c File Reference	425
7.30.1	Detailed Description	425
7.30.2	Function Documentation	425
7.30.2.1	AppInit()	425

7.30.2.2	AppTask()	426
7.31	app.c File Reference	426
7.31.1	Detailed Description	426
7.31.2	Function Documentation	427
7.31.2.1	AppInit()	427
7.31.2.2	AppTask()	427
7.32	app.c File Reference	427
7.32.1	Detailed Description	428
7.32.2	Function Documentation	428
7.32.2.1	AppInit()	428
7.32.2.2	AppTask()	428
7.33	app.c File Reference	429
7.33.1	Detailed Description	429
7.33.2	Function Documentation	429
7.33.2.1	AppInit()	429
7.33.2.2	AppTask()	430
7.34	app.c File Reference	430
7.34.1	Detailed Description	430
7.34.2	Function Documentation	430
7.34.2.1	AppInit()	431
7.34.2.2	AppTask()	431
7.35	app.h File Reference	431
7.35.1	Detailed Description	431
7.35.2	Function Documentation	431
7.35.2.1	AppInit()	432
7.35.2.2	AppTask()	432
7.36	app.h File Reference	432
7.36.1	Detailed Description	433
7.36.2	Function Documentation	433
7.36.2.1	AppInit()	433

7.36.2.2	AppTask()	433
7.37	app.h File Reference	434
7.37.1	Detailed Description	434
7.37.2	Function Documentation	434
7.37.2.1	AppInit()	434
7.37.2.2	AppTask()	434
7.38	app.h File Reference	435
7.38.1	Detailed Description	435
7.38.2	Function Documentation	435
7.38.2.1	AppInit()	435
7.38.2.2	AppTask()	436
7.39	app.h File Reference	436
7.39.1	Detailed Description	436
7.39.2	Function Documentation	436
7.39.2.1	AppInit()	436
7.39.2.2	AppTask()	437
7.40	app.h File Reference	437
7.40.1	Detailed Description	438
7.40.2	Function Documentation	438
7.40.2.1	AppInit()	438
7.40.2.2	AppTask()	438
7.41	app.h File Reference	438
7.41.1	Detailed Description	439
7.41.2	Function Documentation	439
7.41.2.1	AppInit()	439
7.41.2.2	AppTask()	439
7.42	app.h File Reference	440
7.42.1	Detailed Description	440
7.42.2	Function Documentation	440
7.42.2.1	AppInit()	440

7.42.2.2	AppTask()	441
7.43	app.h File Reference	441
7.43.1	Detailed Description	441
7.43.2	Function Documentation	441
7.43.2.1	AppInit()	441
7.43.2.2	AppTask()	442
7.44	app.h File Reference	442
7.44.1	Detailed Description	443
7.44.2	Function Documentation	443
7.44.2.1	AppInit()	443
7.44.2.2	AppTask()	443
7.45	app.h File Reference	443
7.45.1	Detailed Description	444
7.45.2	Function Documentation	444
7.45.2.1	AppInit()	444
7.45.2.2	AppTask()	444
7.46	app.h File Reference	445
7.46.1	Detailed Description	445
7.46.2	Function Documentation	445
7.46.2.1	AppInit()	445
7.46.2.2	AppTask()	446
7.47	app.h File Reference	446
7.47.1	Detailed Description	446
7.47.2	Function Documentation	446
7.47.2.1	AppInit()	446
7.47.2.2	AppTask()	447
7.48	app.h File Reference	447
7.48.1	Detailed Description	448
7.48.2	Function Documentation	448
7.48.2.1	AppInit()	448

7.48.2.2	AppTask()	448
7.49	app.h File Reference	448
7.49.1	Detailed Description	449
7.49.2	Function Documentation	449
7.49.2.1	AppInit()	449
7.49.2.2	AppTask()	449
7.50	app.h File Reference	450
7.50.1	Detailed Description	450
7.50.2	Function Documentation	450
7.50.2.1	AppInit()	450
7.50.2.2	AppTask()	451
7.51	app.h File Reference	451
7.51.1	Detailed Description	451
7.51.2	Function Documentation	451
7.51.2.1	AppInit()	451
7.51.2.2	AppTask()	452
7.52	app.h File Reference	452
7.52.1	Detailed Description	453
7.52.2	Function Documentation	453
7.52.2.1	AppInit()	453
7.52.2.2	AppTask()	453
7.53	app.h File Reference	453
7.53.1	Detailed Description	454
7.53.2	Function Documentation	454
7.53.2.1	AppInit()	454
7.53.2.2	AppTask()	454
7.54	app.h File Reference	455
7.54.1	Detailed Description	455
7.54.2	Function Documentation	455
7.54.2.1	AppInit()	455

7.54.2.2	AppTask()	456
7.55	app.h File Reference	456
7.55.1	Detailed Description	456
7.55.2	Function Documentation	456
7.55.2.1	AppInit()	456
7.55.2.2	AppTask()	457
7.56	app.h File Reference	457
7.56.1	Detailed Description	458
7.56.2	Function Documentation	458
7.56.2.1	AppInit()	458
7.56.2.2	AppTask()	458
7.57	app.h File Reference	458
7.57.1	Detailed Description	459
7.57.2	Function Documentation	459
7.57.2.1	AppInit()	459
7.57.2.2	AppTask()	459
7.58	app.h File Reference	460
7.58.1	Detailed Description	460
7.58.2	Function Documentation	460
7.58.2.1	AppInit()	460
7.58.2.2	AppTask()	461
7.59	app.h File Reference	461
7.59.1	Detailed Description	461
7.59.2	Function Documentation	461
7.59.2.1	AppInit()	461
7.59.2.2	AppTask()	462
7.60	app.h File Reference	462
7.60.1	Detailed Description	463
7.60.2	Function Documentation	463
7.60.2.1	AppInit()	463

7.60.2.2	AppTask()	463
7.61	app.h File Reference	463
7.61.1	Detailed Description	464
7.61.2	Function Documentation	464
7.61.2.1	AppInit()	464
7.61.2.2	AppTask()	464
7.62	app.h File Reference	465
7.62.1	Detailed Description	465
7.62.2	Function Documentation	465
7.62.2.1	AppInit()	465
7.62.2.2	AppTask()	466
7.63	app.h File Reference	466
7.63.1	Detailed Description	466
7.63.2	Function Documentation	466
7.63.2.1	AppInit()	466
7.63.2.2	AppTask()	467
7.64	app.h File Reference	467
7.64.1	Detailed Description	468
7.64.2	Function Documentation	468
7.64.2.1	AppInit()	468
7.64.2.2	AppTask()	468
7.65	app.h File Reference	468
7.65.1	Detailed Description	469
7.65.2	Function Documentation	469
7.65.2.1	AppInit()	469
7.65.2.2	AppTask()	469
7.66	app.h File Reference	470
7.66.1	Detailed Description	470
7.66.2	Function Documentation	470
7.66.2.1	AppInit()	470

7.66.2.2	AppTask()	471
7.67	app.h File Reference	471
7.67.1	Detailed Description	471
7.67.2	Function Documentation	471
7.67.2.1	AppInit()	471
7.67.2.2	AppTask()	472
7.68	app.h File Reference	472
7.68.1	Detailed Description	473
7.68.2	Function Documentation	473
7.68.2.1	AppInit()	473
7.68.2.2	AppTask()	473
7.69	asserts.c File Reference	474
7.69.1	Detailed Description	474
7.69.2	Function Documentation	474
7.69.2.1	AssertFailure()	474
7.70	asserts.h File Reference	475
7.70.1	Detailed Description	475
7.70.2	Function Documentation	475
7.70.2.1	AssertFailure()	475
7.71	backdoor.c File Reference	476
7.71.1	Detailed Description	476
7.71.2	Function Documentation	476
7.71.2.1	BackDoorCheck()	477
7.71.2.2	BackDoorEntryHook()	477
7.71.2.3	BackDoorInit()	477
7.71.2.4	BackDoorInitHook()	478
7.72	backdoor.h File Reference	478
7.72.1	Detailed Description	478
7.72.2	Function Documentation	478
7.72.2.1	BackDoorCheck()	479

7.72.2.2	BackDoorInit()	479
7.73	blt_conf.h File Reference	479
7.73.1	Detailed Description	480
7.74	blt_conf.h File Reference	480
7.74.1	Detailed Description	482
7.75	blt_conf.h File Reference	482
7.75.1	Detailed Description	483
7.76	blt_conf.h File Reference	483
7.76.1	Detailed Description	484
7.77	blt_conf.h File Reference	484
7.77.1	Detailed Description	486
7.78	blt_conf.h File Reference	486
7.78.1	Detailed Description	487
7.79	blt_conf.h File Reference	487
7.79.1	Detailed Description	488
7.80	blt_conf.h File Reference	489
7.80.1	Detailed Description	490
7.81	blt_conf.h File Reference	490
7.81.1	Detailed Description	491
7.82	blt_conf.h File Reference	492
7.82.1	Detailed Description	493
7.83	blt_conf.h File Reference	493
7.83.1	Detailed Description	494
7.84	blt_conf.h File Reference	494
7.84.1	Detailed Description	495
7.85	blt_conf.h File Reference	495
7.85.1	Detailed Description	496
7.86	blt_conf.h File Reference	496
7.86.1	Detailed Description	498
7.87	blt_conf.h File Reference	498

7.87.1 Detailed Description	499
7.88 blt_conf.h File Reference	500
7.88.1 Detailed Description	501
7.89 blt_conf.h File Reference	501
7.89.1 Detailed Description	503
7.90 blt_conf.h File Reference	503
7.90.1 Detailed Description	504
7.91 blt_conf.h File Reference	504
7.91.1 Detailed Description	506
7.92 blt_conf.h File Reference	506
7.92.1 Detailed Description	507
7.93 blt_conf.h File Reference	507
7.93.1 Detailed Description	509
7.94 blt_conf.h File Reference	509
7.94.1 Detailed Description	511
7.95 blt_conf.h File Reference	511
7.95.1 Detailed Description	513
7.96 blt_conf.h File Reference	513
7.96.1 Detailed Description	514
7.97 blt_conf.h File Reference	514
7.97.1 Detailed Description	515
7.98 blt_conf.h File Reference	516
7.98.1 Detailed Description	517
7.99 blt_conf.h File Reference	517
7.99.1 Detailed Description	518
7.100 blt_conf.h File Reference	518
7.100.1 Detailed Description	520
7.101 blt_conf.h File Reference	520
7.101.1 Detailed Description	521
7.102 blt_conf.h File Reference	521

7.102.1 Detailed Description	522
7.103blt_conf.h File Reference	522
7.103.1 Detailed Description	523
7.104blt_conf.h File Reference	523
7.104.1 Detailed Description	525
7.105blt_conf.h File Reference	525
7.105.1 Detailed Description	526
7.106blt_conf.h File Reference	526
7.106.1 Detailed Description	528
7.107blt_conf.h File Reference	528
7.107.1 Detailed Description	529
7.108blt_conf.h File Reference	530
7.108.1 Detailed Description	531
7.109blt_conf.h File Reference	531
7.109.1 Detailed Description	533
7.110blt_conf.h File Reference	533
7.110.1 Detailed Description	535
7.111blt_conf.h File Reference	535
7.111.1 Detailed Description	537
7.112blt_conf.h File Reference	537
7.112.1 Detailed Description	539
7.113blt_conf.h File Reference	539
7.113.1 Detailed Description	541
7.114blt_conf.h File Reference	541
7.114.1 Detailed Description	543
7.115blt_conf.h File Reference	543
7.115.1 Detailed Description	545
7.116blt_conf.h File Reference	545
7.116.1 Detailed Description	546
7.117blt_conf.h File Reference	546

7.117.1 Detailed Description	548
7.118blt_conf.h File Reference	548
7.118.1 Detailed Description	549
7.119blt_conf.h File Reference	550
7.119.1 Detailed Description	551
7.120blt_conf.h File Reference	551
7.120.1 Detailed Description	552
7.121blt_conf.h File Reference	552
7.121.1 Detailed Description	553
7.122blt_conf.h File Reference	553
7.122.1 Detailed Description	554
7.123blt_conf.h File Reference	554
7.123.1 Detailed Description	555
7.124blt_conf.h File Reference	555
7.124.1 Detailed Description	556
7.125blt_conf.h File Reference	557
7.125.1 Detailed Description	558
7.126blt_conf.h File Reference	558
7.126.1 Detailed Description	559
7.127blt_conf.h File Reference	560
7.127.1 Detailed Description	561
7.128blt_conf.h File Reference	561
7.128.1 Detailed Description	563
7.129blt_conf.h File Reference	564
7.129.1 Detailed Description	566
7.130blt_conf.h File Reference	566
7.130.1 Detailed Description	568
7.131blt_conf.h File Reference	569
7.131.1 Detailed Description	571
7.132blt_conf.h File Reference	571

7.132.1 Detailed Description	573
7.133blt_conf.h File Reference	573
7.133.1 Detailed Description	575
7.134blt_conf.h File Reference	575
7.134.1 Detailed Description	576
7.135blt_conf.h File Reference	577
7.135.1 Detailed Description	578
7.136blt_conf.h File Reference	578
7.136.1 Detailed Description	580
7.137blt_conf.h File Reference	580
7.137.1 Detailed Description	581
7.138blt_conf.h File Reference	582
7.138.1 Detailed Description	583
7.139blt_conf.h File Reference	583
7.139.1 Detailed Description	584
7.140blt_conf.h File Reference	585
7.140.1 Detailed Description	586
7.141blt_conf.h File Reference	586
7.141.1 Detailed Description	588
7.142blt_conf.h File Reference	589
7.142.1 Detailed Description	591
7.143blt_conf.h File Reference	591
7.143.1 Detailed Description	593
7.144blt_conf.h File Reference	593
7.144.1 Detailed Description	594
7.145blt_conf.h File Reference	594
7.145.1 Detailed Description	596
7.146blt_conf.h File Reference	596
7.146.1 Detailed Description	598
7.147blt_conf.h File Reference	598

7.147.1 Detailed Description	600
7.148blt_conf.h File Reference	600
7.148.1 Detailed Description	602
7.149blt_conf.h File Reference	602
7.149.1 Detailed Description	604
7.150blt_conf.h File Reference	604
7.150.1 Detailed Description	606
7.151blt_conf.h File Reference	606
7.151.1 Detailed Description	607
7.152blt_conf.h File Reference	607
7.152.1 Detailed Description	609
7.153blt_conf.h File Reference	609
7.153.1 Detailed Description	611
7.154blt_conf.h File Reference	611
7.154.1 Detailed Description	612
7.155blt_conf.h File Reference	612
7.155.1 Detailed Description	614
7.156blt_conf.h File Reference	614
7.156.1 Detailed Description	615
7.157boot.c File Reference	615
7.157.1 Detailed Description	616
7.157.2 Function Documentation	616
7.157.2.1 BootActivate()	617
7.157.2.2 BootComCheckActivationRequest()	617
7.157.2.3 BootComInit()	617
7.157.2.4 BootComRs232CheckActivationRequest()	618
7.157.2.5 BootComRs232Init()	618
7.157.2.6 Rs232ReceiveByte()	618
7.158boot.c File Reference	619
7.158.1 Detailed Description	620

7.158.2 Function Documentation	620
7.158.2.1 BootActivate()	620
7.158.2.2 BootComCheckActivationRequest()	620
7.158.2.3 BootComInit()	620
7.158.2.4 BootComRs232CheckActivationRequest()	621
7.158.2.5 BootComRs232Init()	621
7.158.2.6 Rs232ReceiveByte()	621
7.159boot.c File Reference	622
7.159.1 Detailed Description	623
7.159.2 Function Documentation	623
7.159.2.1 BootActivate()	623
7.159.2.2 BootComCheckActivationRequest()	623
7.159.2.3 BootComInit()	623
7.159.2.4 BootComRs232CheckActivationRequest()	624
7.159.2.5 BootComRs232Init()	624
7.159.2.6 Rs232ReceiveByte()	624
7.160boot.c File Reference	625
7.160.1 Detailed Description	626
7.160.2 Function Documentation	626
7.160.2.1 BootActivate()	626
7.160.2.2 BootComCheckActivationRequest()	626
7.160.2.3 BootComInit()	626
7.160.2.4 BootComRs232CheckActivationRequest()	627
7.160.2.5 BootComRs232Init()	627
7.160.2.6 Rs232ReceiveByte()	627
7.161boot.c File Reference	628
7.161.1 Detailed Description	629
7.161.2 Function Documentation	629
7.161.2.1 BootActivate()	629
7.161.2.2 BootComCheckActivationRequest()	629

7.161.2.3 BootComInit()	629
7.161.2.4 BootComRs232CheckActivationRequest()	630
7.161.2.5 BootComRs232Init()	630
7.161.2.6 Rs232ReceiveByte()	630
7.162boot.c File Reference	631
7.162.1 Detailed Description	632
7.162.2 Function Documentation	632
7.162.2.1 BootActivate()	632
7.162.2.2 BootComCanCheckActivationRequest()	632
7.162.2.3 BootComCanInit()	633
7.162.2.4 BootComCheckActivationRequest()	633
7.162.2.5 BootComInit()	633
7.162.2.6 BootComRs232CheckActivationRequest()	633
7.162.2.7 BootComRs232Init()	634
7.162.2.8 CanGetSpeedConfig()	634
7.162.2.9 Rs232ReceiveByte()	634
7.162.3 Variable Documentation	635
7.162.3.1 canTiming	635
7.163boot.c File Reference	636
7.163.1 Detailed Description	637
7.163.2 Function Documentation	637
7.163.2.1 BootActivate()	637
7.163.2.2 BootComCanCheckActivationRequest()	637
7.163.2.3 BootComCanInit()	638
7.163.2.4 BootComCheckActivationRequest()	638
7.163.2.5 BootComInit()	638
7.163.2.6 BootComRs232CheckActivationRequest()	638
7.163.2.7 BootComRs232Init()	639
7.163.2.8 CanGetSpeedConfig()	639
7.163.2.9 Rs232ReceiveByte()	639

7.163.3 Variable Documentation	640
7.163.3.1 canTiming	640
7.164boot.c File Reference	641
7.164.1 Detailed Description	642
7.164.2 Function Documentation	642
7.164.2.1 BootActivate()	642
7.164.2.2 BootComCanCheckActivationRequest()	642
7.164.2.3 BootComCanInit()	643
7.164.2.4 BootComCheckActivationRequest()	643
7.164.2.5 BootComInit()	643
7.164.2.6 BootComRs232CheckActivationRequest()	643
7.164.2.7 BootComRs232Init()	644
7.164.2.8 CanGetSpeedConfig()	644
7.164.2.9 Rs232ReceiveByte()	644
7.164.3 Variable Documentation	645
7.164.3.1 canTiming	645
7.165boot.c File Reference	646
7.165.1 Detailed Description	647
7.165.2 Function Documentation	647
7.165.2.1 BootActivate()	647
7.165.2.2 BootComCanCheckActivationRequest()	647
7.165.2.3 BootComCanInit()	648
7.165.2.4 BootComCheckActivationRequest()	648
7.165.2.5 BootComInit()	648
7.165.2.6 BootComRs232CheckActivationRequest()	648
7.165.2.7 BootComRs232Init()	649
7.165.2.8 CanGetSpeedConfig()	649
7.165.2.9 Rs232ReceiveByte()	649
7.165.3 Variable Documentation	650
7.165.3.1 canTiming	650

7.166boot.c File Reference	651
7.166.1 Detailed Description	652
7.166.2 Function Documentation	652
7.166.2.1 BootActivate()	652
7.166.2.2 BootComCheckActivationRequest()	652
7.166.2.3 BootComInit()	652
7.166.2.4 BootComRs232CheckActivationRequest()	653
7.166.2.5 BootComRs232Init()	653
7.166.2.6 Rs232ReceiveByte()	653
7.167boot.c File Reference	654
7.167.1 Detailed Description	655
7.167.2 Function Documentation	655
7.167.2.1 BootActivate()	655
7.167.2.2 BootComCheckActivationRequest()	655
7.167.2.3 BootComInit()	655
7.167.2.4 BootComRs232CheckActivationRequest()	656
7.167.2.5 BootComRs232Init()	656
7.167.2.6 Rs232ReceiveByte()	656
7.168boot.c File Reference	657
7.168.1 Detailed Description	658
7.168.2 Function Documentation	658
7.168.2.1 BootActivate()	658
7.168.2.2 BootComCheckActivationRequest()	658
7.168.2.3 BootComInit()	658
7.168.2.4 BootComRs232CheckActivationRequest()	659
7.168.2.5 BootComRs232Init()	659
7.168.2.6 Rs232ReceiveByte()	659
7.169boot.c File Reference	660
7.169.1 Detailed Description	661
7.169.2 Function Documentation	661

7.169.2.1 BootActivate()	661
7.169.2.2 BootComCheckActivationRequest()	661
7.169.2.3 BootComInit()	661
7.169.2.4 BootComRs232CheckActivationRequest()	662
7.169.2.5 BootComRs232Init()	662
7.169.2.6 Rs232ReceiveByte()	662
7.170boot.c File Reference	663
7.170.1 Detailed Description	664
7.170.2 Function Documentation	664
7.170.2.1 BootActivate()	664
7.170.2.2 BootComCanCheckActivationRequest()	664
7.170.2.3 BootComCanInit()	665
7.170.2.4 BootComCheckActivationRequest()	665
7.170.2.5 BootComInit()	665
7.170.2.6 BootComRs232CheckActivationRequest()	665
7.170.2.7 BootComRs232Init()	666
7.170.2.8 Rs232ReceiveByte()	666
7.171boot.c File Reference	666
7.171.1 Detailed Description	667
7.171.2 Function Documentation	667
7.171.2.1 BootActivate()	667
7.171.2.2 BootComCanCheckActivationRequest()	668
7.171.2.3 BootComCanInit()	668
7.171.2.4 BootComCheckActivationRequest()	668
7.171.2.5 BootComInit()	669
7.171.2.6 BootComRs232CheckActivationRequest()	669
7.171.2.7 BootComRs232Init()	669
7.171.2.8 Rs232ReceiveByte()	669
7.172boot.c File Reference	670
7.172.1 Detailed Description	671

7.172.2 Function Documentation	671
7.172.2.1 BootActivate()	671
7.172.2.2 BootComCanCheckActivationRequest()	672
7.172.2.3 BootComCanInit()	672
7.172.2.4 BootComCheckActivationRequest()	672
7.172.2.5 BootComInit()	673
7.172.2.6 BootComRs232CheckActivationRequest()	673
7.172.2.7 BootComRs232Init()	673
7.172.2.8 CanGetSpeedConfig()	673
7.172.2.9 Rs232ReceiveByte()	674
7.172.3 Variable Documentation	674
7.172.3.1 canTiming	674
7.173boot.c File Reference	675
7.173.1 Detailed Description	676
7.173.2 Function Documentation	676
7.173.2.1 BootActivate()	676
7.173.2.2 BootComCanCheckActivationRequest()	677
7.173.2.3 BootComCanInit()	677
7.173.2.4 BootComCheckActivationRequest()	677
7.173.2.5 BootComInit()	678
7.173.2.6 BootComRs232CheckActivationRequest()	678
7.173.2.7 BootComRs232Init()	678
7.173.2.8 CanGetSpeedConfig()	678
7.173.2.9 Rs232ReceiveByte()	679
7.173.3 Variable Documentation	679
7.173.3.1 canTiming	679
7.174boot.c File Reference	680
7.174.1 Detailed Description	681
7.174.2 Function Documentation	681
7.174.2.1 BootActivate()	681

7.174.2.2 BootComCanCheckActivationRequest()	682
7.174.2.3 BootComCanInit()	682
7.174.2.4 BootComCheckActivationRequest()	682
7.174.2.5 BootComInit()	683
7.174.2.6 BootComRs232CheckActivationRequest()	683
7.174.2.7 BootComRs232Init()	683
7.174.2.8 CanGetSpeedConfig()	683
7.174.2.9 Rs232ReceiveByte()	684
7.174.3 Variable Documentation	684
7.174.3.1 canTiming	684
7.175boot.c File Reference	685
7.175.1 Detailed Description	686
7.175.2 Function Documentation	686
7.175.2.1 BootActivate()	686
7.175.2.2 BootComCanCheckActivationRequest()	687
7.175.2.3 BootComCanInit()	687
7.175.2.4 BootComCheckActivationRequest()	687
7.175.2.5 BootComInit()	688
7.175.2.6 BootComRs232CheckActivationRequest()	688
7.175.2.7 BootComRs232Init()	688
7.175.2.8 CanGetSpeedConfig()	688
7.175.2.9 Rs232ReceiveByte()	689
7.175.3 Variable Documentation	689
7.175.3.1 canTiming	689
7.176boot.c File Reference	690
7.176.1 Detailed Description	691
7.176.2 Function Documentation	691
7.176.2.1 BootActivate()	691
7.176.2.2 BootComCheckActivationRequest()	691
7.176.2.3 BootComInit()	692

7.176.2.4 BootComRs232CheckActivationRequest()	692
7.176.2.5 BootComRs232Init()	692
7.176.2.6 Rs232ReceiveByte()	692
7.177boot.c File Reference	693
7.177.1 Detailed Description	694
7.177.2 Function Documentation	694
7.177.2.1 BootActivate()	694
7.177.2.2 BootComCheckActivationRequest()	694
7.177.2.3 BootComInit()	694
7.177.2.4 BootComRs232CheckActivationRequest()	695
7.177.2.5 BootComRs232Init()	695
7.177.2.6 Rs232ReceiveByte()	695
7.178boot.c File Reference	696
7.178.1 Detailed Description	696
7.178.2 Function Documentation	696
7.178.2.1 BootActivate()	697
7.178.2.2 BootComCheckActivationRequest()	697
7.178.2.3 BootComInit()	697
7.178.2.4 BootComRs232CheckActivationRequest()	697
7.178.2.5 BootComRs232Init()	698
7.178.2.6 Rs232ReceiveByte()	698
7.179boot.c File Reference	698
7.179.1 Detailed Description	699
7.179.2 Function Documentation	699
7.179.2.1 BootActivate()	699
7.179.2.2 BootComCheckActivationRequest()	699
7.179.2.3 BootComInit()	700
7.179.2.4 BootComRs232CheckActivationRequest()	700
7.179.2.5 BootComRs232Init()	700
7.179.2.6 Rs232ReceiveByte()	700

7.180boot.c File Reference	701
7.180.1 Detailed Description	702
7.180.2 Function Documentation	702
7.180.2.1 BootActivate()	702
7.180.2.2 BootComCanCheckActivationRequest()	702
7.180.2.3 BootComCanInit()	703
7.180.2.4 BootComCheckActivationRequest()	703
7.180.2.5 BootComInit()	703
7.180.2.6 BootComRs232CheckActivationRequest()	703
7.180.2.7 BootComRs232Init()	704
7.180.2.8 CanSetBittiming()	704
7.180.2.9 Rs232ReceiveByte()	704
7.181boot.c File Reference	705
7.181.1 Detailed Description	706
7.181.2 Function Documentation	706
7.181.2.1 BootActivate()	706
7.181.2.2 BootComCanCheckActivationRequest()	706
7.181.2.3 BootComCanInit()	706
7.181.2.4 BootComCheckActivationRequest()	707
7.181.2.5 BootComInit()	707
7.181.2.6 BootComRs232CheckActivationRequest()	707
7.181.2.7 BootComRs232Init()	707
7.181.2.8 CanSetBittiming()	708
7.181.2.9 Rs232ReceiveByte()	708
7.182boot.c File Reference	708
7.182.1 Detailed Description	709
7.182.2 Function Documentation	710
7.182.2.1 BootActivate()	710
7.182.2.2 BootComCheckActivationRequest()	710
7.182.2.3 BootComInit()	710

7.182.2.4 BootComRs232CheckActivationRequest()	711
7.182.2.5 BootComRs232Init()	711
7.182.2.6 Rs232ReceiveByte()	711
7.183boot.c File Reference	712
7.183.1 Detailed Description	713
7.183.2 Function Documentation	713
7.183.2.1 BootActivate()	713
7.183.2.2 BootComCheckActivationRequest()	713
7.183.2.3 BootComInit()	713
7.183.2.4 BootComRs232CheckActivationRequest()	714
7.183.2.5 BootComRs232Init()	714
7.183.2.6 Rs232ReceiveByte()	714
7.184boot.c File Reference	715
7.184.1 Detailed Description	716
7.184.2 Function Documentation	716
7.184.2.1 BootActivate()	716
7.184.2.2 BootComCheckActivationRequest()	716
7.184.2.3 BootComInit()	716
7.184.2.4 BootComRs232CheckActivationRequest()	717
7.184.2.5 BootComRs232Init()	717
7.184.2.6 Rs232ReceiveByte()	717
7.185boot.c File Reference	718
7.185.1 Detailed Description	719
7.185.2 Function Documentation	719
7.185.2.1 BootActivate()	719
7.185.2.2 BootComCheckActivationRequest()	719
7.185.2.3 BootComInit()	719
7.185.2.4 BootComRs232CheckActivationRequest()	720
7.185.2.5 BootComRs232Init()	720
7.185.2.6 Rs232ReceiveByte()	720

7.186boot.c File Reference	721
7.186.1 Detailed Description	722
7.186.2 Function Documentation	722
7.186.2.1 BootActivate()	722
7.186.2.2 BootComCanCheckActivationRequest()	722
7.186.2.3 BootComCanInit()	723
7.186.2.4 BootComCheckActivationRequest()	723
7.186.2.5 BootComInit()	723
7.186.2.6 BootComRs232CheckActivationRequest()	723
7.186.2.7 BootComRs232Init()	724
7.186.2.8 CanGetSpeedConfig()	724
7.186.2.9 Rs232ReceiveByte()	724
7.186.3 Variable Documentation	725
7.186.3.1 canTiming	725
7.187boot.c File Reference	726
7.187.1 Detailed Description	727
7.187.2 Function Documentation	727
7.187.2.1 BootActivate()	727
7.187.2.2 BootComCanCheckActivationRequest()	727
7.187.2.3 BootComCanInit()	728
7.187.2.4 BootComCheckActivationRequest()	728
7.187.2.5 BootComInit()	728
7.187.2.6 BootComRs232CheckActivationRequest()	728
7.187.2.7 BootComRs232Init()	729
7.187.2.8 CanGetSpeedConfig()	729
7.187.2.9 Rs232ReceiveByte()	729
7.187.3 Variable Documentation	730
7.187.3.1 canTiming	730
7.188boot.c File Reference	731
7.188.1 Detailed Description	732

7.188.2 Function Documentation	732
7.188.2.1 BootActivate()	732
7.188.2.2 BootComCanCheckActivationRequest()	732
7.188.2.3 BootComCanInit()	733
7.188.2.4 BootComCheckActivationRequest()	733
7.188.2.5 BootComInit()	733
7.188.2.6 BootComRs232CheckActivationRequest()	733
7.188.2.7 BootComRs232Init()	734
7.188.2.8 CanGetSpeedConfig()	734
7.188.2.9 Rs232ReceiveByte()	734
7.188.3 Variable Documentation	735
7.188.3.1 canTiming	735
7.189boot.c File Reference	736
7.189.1 Detailed Description	737
7.189.2 Function Documentation	737
7.189.2.1 BootActivate()	737
7.189.2.2 BootComCanCheckActivationRequest()	737
7.189.2.3 BootComCanInit()	738
7.189.2.4 BootComCheckActivationRequest()	738
7.189.2.5 BootComInit()	738
7.189.2.6 BootComRs232CheckActivationRequest()	738
7.189.2.7 BootComRs232Init()	739
7.189.2.8 CanGetSpeedConfig()	739
7.189.2.9 Rs232ReceiveByte()	739
7.189.3 Variable Documentation	740
7.189.3.1 canTiming	740
7.190boot.c File Reference	741
7.190.1 Detailed Description	742
7.190.2 Function Documentation	742
7.190.2.1 BootActivate()	742

7.190.2.2 BootComCanCheckActivationRequest()	742
7.190.2.3 BootComCanInit()	742
7.190.2.4 BootComCheckActivationRequest()	743
7.190.2.5 BootComInit()	743
7.190.2.6 CanGetSpeedConfig()	743
7.190.3 Variable Documentation	744
7.190.3.1 canTiming	744
7.191boot.c File Reference	744
7.191.1 Detailed Description	745
7.191.2 Function Documentation	745
7.191.2.1 BootActivate()	745
7.191.2.2 BootComCanCheckActivationRequest()	746
7.191.2.3 BootComCanInit()	746
7.191.2.4 BootComCheckActivationRequest()	746
7.191.2.5 BootComInit()	747
7.191.2.6 CanGetSpeedConfig()	747
7.191.3 Variable Documentation	747
7.191.3.1 canTiming	747
7.192boot.c File Reference	748
7.192.1 Detailed Description	749
7.192.2 Function Documentation	749
7.192.2.1 BootActivate()	749
7.192.2.2 BootComCanCheckActivationRequest()	750
7.192.2.3 BootComCanInit()	750
7.192.2.4 BootComCheckActivationRequest()	750
7.192.2.5 BootComInit()	751
7.192.2.6 CanGetSpeedConfig()	751
7.192.3 Variable Documentation	751
7.192.3.1 canTiming	751
7.193boot.c File Reference	752

7.193.1 Detailed Description	753
7.193.2 Function Documentation	753
7.193.2.1 BootActivate()	753
7.193.2.2 BootComCanCheckActivationRequest()	754
7.193.2.3 BootComCanInit()	754
7.193.2.4 BootComCheckActivationRequest()	754
7.193.2.5 BootComInit()	755
7.193.2.6 CanGetSpeedConfig()	755
7.193.3 Variable Documentation	755
7.193.3.1 canTiming	755
7.194boot.c File Reference	756
7.194.1 Detailed Description	757
7.194.2 Function Documentation	757
7.194.2.1 BootActivate()	758
7.194.2.2 BootComCanCheckActivationRequest()	758
7.194.2.3 BootComCanInit()	758
7.194.2.4 BootComCheckActivationRequest()	759
7.194.2.5 BootComInit()	759
7.194.2.6 BootComRs232CheckActivationRequest()	759
7.194.2.7 BootComRs232Init()	759
7.194.2.8 CanGetSpeedConfig()	760
7.194.2.9 Rs232ReceiveByte()	760
7.194.3 Variable Documentation	760
7.194.3.1 canTiming	761
7.195boot.c File Reference	761
7.195.1 Detailed Description	762
7.195.2 Function Documentation	763
7.195.2.1 BootActivate()	763
7.195.2.2 BootComCanCheckActivationRequest()	763
7.195.2.3 BootComCanInit()	763

7.195.2.4 BootComCheckActivationRequest()	764
7.195.2.5 BootComInit()	764
7.195.2.6 BootComRs232CheckActivationRequest()	764
7.195.2.7 BootComRs232Init()	764
7.195.2.8 CanGetSpeedConfig()	765
7.195.2.9 Rs232ReceiveByte()	765
7.195.3 Variable Documentation	765
7.195.3.1 canTiming	766
7.196boot.c File Reference	766
7.196.1 Detailed Description	767
7.196.2 Function Documentation	768
7.196.2.1 BootActivate()	768
7.196.2.2 BootComCanCheckActivationRequest()	768
7.196.2.3 BootComCanInit()	768
7.196.2.4 BootComCheckActivationRequest()	769
7.196.2.5 BootComInit()	769
7.196.2.6 BootComRs232CheckActivationRequest()	769
7.196.2.7 BootComRs232Init()	769
7.196.2.8 CanGetSpeedConfig()	770
7.196.2.9 Rs232ReceiveByte()	770
7.196.3 Variable Documentation	770
7.196.3.1 canTiming	771
7.197boot.c File Reference	771
7.197.1 Detailed Description	772
7.197.2 Function Documentation	773
7.197.2.1 BootActivate()	773
7.197.2.2 BootComCanCheckActivationRequest()	773
7.197.2.3 BootComCanInit()	773
7.197.2.4 BootComCheckActivationRequest()	774
7.197.2.5 BootComInit()	774

7.197.2.6 BootComRs232CheckActivationRequest()	774
7.197.2.7 BootComRs232Init()	774
7.197.2.8 CanGetSpeedConfig()	775
7.197.2.9 Rs232ReceiveByte()	775
7.197.3 Variable Documentation	775
7.197.3.1 canTiming	776
7.198boot.c File Reference	776
7.198.1 Detailed Description	778
7.198.2 Function Documentation	778
7.198.2.1 BootActivate()	778
7.198.2.2 BootComCanCheckActivationRequest()	778
7.198.2.3 BootComCanInit()	779
7.198.2.4 BootComCheckActivationRequest()	779
7.198.2.5 BootComInit()	779
7.198.2.6 BootComRs232CheckActivationRequest()	779
7.198.2.7 BootComRs232Init()	780
7.198.2.8 CanDisabledModeEnter()	780
7.198.2.9 CanDisabledModeExit()	780
7.198.2.10CanFreezeModeEnter()	781
7.198.2.11CanFreezeModeExit()	781
7.198.2.12CanGetSpeedConfig()	781
7.198.2.13Rs232ReceiveByte()	782
7.198.3 Variable Documentation	782
7.198.3.1 canTiming	782
7.199boot.c File Reference	783
7.199.1 Detailed Description	784
7.199.2 Function Documentation	784
7.199.2.1 BootActivate()	785
7.199.2.2 BootComCanCheckActivationRequest()	785
7.199.2.3 BootComCanInit()	785

7.199.2.4 BootComCheckActivationRequest()	786
7.199.2.5 BootComInit()	786
7.199.2.6 BootComRs232CheckActivationRequest()	786
7.199.2.7 BootComRs232Init()	786
7.199.2.8 CanDisabledModeEnter()	787
7.199.2.9 CanDisabledModeExit()	787
7.199.2.10 CanFreezeModeEnter()	787
7.199.2.11 CanFreezeModeExit()	788
7.199.2.12 CanGetSpeedConfig()	788
7.199.2.13 Rs232ReceiveByte()	788
7.199.3 Variable Documentation	789
7.199.3.1 canTiming	789
7.200 boot.c File Reference	790
7.200.1 Detailed Description	791
7.200.2 Function Documentation	791
7.200.2.1 BootActivate()	791
7.200.2.2 BootComCanCheckActivationRequest()	791
7.200.2.3 BootComCanInit()	792
7.200.2.4 BootComCheckActivationRequest()	792
7.200.2.5 BootComInit()	792
7.200.2.6 BootComRs232CheckActivationRequest()	792
7.200.2.7 BootComRs232Init()	793
7.200.2.8 CanGetSpeedConfig()	793
7.200.2.9 Rs232ReceiveByte()	793
7.200.3 Variable Documentation	794
7.200.3.1 canTiming	794
7.201 boot.c File Reference	795
7.201.1 Detailed Description	796
7.201.2 Function Documentation	796
7.201.2.1 BootActivate()	796

7.201.2.2 BootComCanCheckActivationRequest()	796
7.201.2.3 BootComCanInit()	797
7.201.2.4 BootComCheckActivationRequest()	797
7.201.2.5 BootComInit()	797
7.201.2.6 BootComRs232CheckActivationRequest()	797
7.201.2.7 BootComRs232Init()	798
7.201.2.8 CanGetSpeedConfig()	798
7.201.2.9 Rs232ReceiveByte()	798
7.201.3 Variable Documentation	799
7.201.3.1 canTiming	799
7.202boot.c File Reference	800
7.202.1 Detailed Description	801
7.202.2 Function Documentation	801
7.202.2.1 BootActivate()	801
7.202.2.2 BootComCanCheckActivationRequest()	801
7.202.2.3 BootComCanInit()	802
7.202.2.4 BootComCheckActivationRequest()	802
7.202.2.5 BootComInit()	802
7.202.2.6 BootComRs232CheckActivationRequest()	802
7.202.2.7 BootComRs232Init()	803
7.202.2.8 CanGetSpeedConfig()	803
7.202.2.9 Rs232ReceiveByte()	803
7.202.3 Variable Documentation	804
7.202.3.1 canTiming	804
7.203boot.c File Reference	805
7.203.1 Detailed Description	806
7.203.2 Function Documentation	806
7.203.2.1 BootActivate()	806
7.203.2.2 BootComCanCheckActivationRequest()	806
7.203.2.3 BootComCanInit()	807

7.203.2.4 BootComCheckActivationRequest()	807
7.203.2.5 BootComInit()	807
7.203.2.6 BootComRs232CheckActivationRequest()	807
7.203.2.7 BootComRs232Init()	808
7.203.2.8 CanGetSpeedConfig()	808
7.203.2.9 Rs232ReceiveByte()	808
7.203.3 Variable Documentation	809
7.203.3.1 canTiming	809
7.204boot.c File Reference	810
7.204.1 Detailed Description	811
7.204.2 Function Documentation	811
7.204.2.1 BootActivate()	811
7.204.2.2 BootComCanCheckActivationRequest()	811
7.204.2.3 BootComCanInit()	812
7.204.2.4 BootComCheckActivationRequest()	812
7.204.2.5 BootComInit()	812
7.204.2.6 BootComRs232CheckActivationRequest()	812
7.204.2.7 BootComRs232Init()	813
7.204.2.8 CanGetSpeedConfig()	813
7.204.2.9 Rs232ReceiveByte()	813
7.204.3 Variable Documentation	814
7.204.3.1 canTiming	814
7.205boot.c File Reference	815
7.205.1 Detailed Description	816
7.205.2 Function Documentation	816
7.205.2.1 BootActivate()	816
7.205.2.2 BootComCanCheckActivationRequest()	816
7.205.2.3 BootComCanInit()	817
7.205.2.4 BootComCheckActivationRequest()	817
7.205.2.5 BootComInit()	817

7.205.2.6 BootComRs232CheckActivationRequest()	817
7.205.2.7 BootComRs232Init()	818
7.205.2.8 CanGetSpeedConfig()	818
7.205.2.9 Rs232ReceiveByte()	818
7.205.3 Variable Documentation	819
7.205.3.1 canTiming	819
7.206boot.c File Reference	820
7.206.1 Detailed Description	821
7.206.2 Function Documentation	821
7.206.2.1 BootActivate()	821
7.206.2.2 BootComCanCheckActivationRequest()	821
7.206.2.3 BootComCanInit()	822
7.206.2.4 BootComCheckActivationRequest()	822
7.206.2.5 BootComInit()	822
7.206.2.6 BootComRs232CheckActivationRequest()	822
7.206.2.7 BootComRs232Init()	823
7.206.2.8 CanGetSpeedConfig()	823
7.206.2.9 Rs232ReceiveByte()	823
7.206.3 Variable Documentation	824
7.206.3.1 canTiming	824
7.207boot.c File Reference	825
7.207.1 Detailed Description	826
7.207.2 Function Documentation	826
7.207.2.1 BootActivate()	826
7.207.2.2 BootComCanCheckActivationRequest()	826
7.207.2.3 BootComCanInit()	827
7.207.2.4 BootComCheckActivationRequest()	827
7.207.2.5 BootComInit()	827
7.207.2.6 BootComRs232CheckActivationRequest()	827
7.207.2.7 BootComRs232Init()	828

7.207.2.8 CanGetSpeedConfig()	828
7.207.2.9 Rs232ReceiveByte()	828
7.207.3 Variable Documentation	829
7.207.3.1 canTiming	829
7.208boot.c File Reference	830
7.208.1 Detailed Description	831
7.208.2 Function Documentation	831
7.208.2.1 BootActivate()	831
7.208.2.2 BootComCanCheckActivationRequest()	831
7.208.2.3 BootComCanInit()	832
7.208.2.4 BootComCheckActivationRequest()	832
7.208.2.5 BootComInit()	832
7.208.2.6 BootComRs232CheckActivationRequest()	832
7.208.2.7 BootComRs232Init()	833
7.208.2.8 CanGetSpeedConfig()	833
7.208.2.9 Rs232ReceiveByte()	833
7.208.3 Variable Documentation	834
7.208.3.1 canTiming	834
7.209boot.c File Reference	835
7.209.1 Detailed Description	836
7.209.2 Function Documentation	836
7.209.2.1 BootActivate()	836
7.209.2.2 BootComCanCheckActivationRequest()	836
7.209.2.3 BootComCanInit()	837
7.209.2.4 BootComCheckActivationRequest()	837
7.209.2.5 BootComInit()	837
7.209.2.6 BootComRs232CheckActivationRequest()	837
7.209.2.7 BootComRs232Init()	838
7.209.2.8 CanGetSpeedConfig()	838
7.209.2.9 Rs232ReceiveByte()	838

7.209.3 Variable Documentation	839
7.209.3.1 canTiming	839
7.210boot.c File Reference	840
7.210.1 Detailed Description	841
7.210.2 Function Documentation	841
7.210.2.1 BootActivate()	841
7.210.2.2 BootComCanCheckActivationRequest()	841
7.210.2.3 BootComCanInit()	842
7.210.2.4 BootComCheckActivationRequest()	842
7.210.2.5 BootComInit()	842
7.210.2.6 BootComRs232CheckActivationRequest()	842
7.210.2.7 BootComRs232Init()	843
7.210.2.8 CanGetSpeedConfig()	843
7.210.2.9 Rs232ReceiveByte()	843
7.210.3 Variable Documentation	844
7.210.3.1 canTiming	844
7.211boot.c File Reference	845
7.211.1 Detailed Description	846
7.211.2 Function Documentation	846
7.211.2.1 BootActivate()	846
7.211.2.2 BootComCanCheckActivationRequest()	846
7.211.2.3 BootComCanInit()	847
7.211.2.4 BootComCheckActivationRequest()	847
7.211.2.5 BootComInit()	847
7.211.2.6 BootComRs232CheckActivationRequest()	847
7.211.2.7 BootComRs232Init()	848
7.211.2.8 CanGetSpeedConfig()	848
7.211.2.9 Rs232ReceiveByte()	848
7.211.3 Variable Documentation	849
7.211.3.1 canTiming	849

7.212boot.c File Reference	850
7.212.1 Detailed Description	851
7.212.2 Function Documentation	851
7.212.2.1 BootActivate()	851
7.212.2.2 BootComCanCheckActivationRequest()	851
7.212.2.3 BootComCanInit()	852
7.212.2.4 BootComCheckActivationRequest()	852
7.212.2.5 BootComInit()	852
7.212.2.6 BootComRs232CheckActivationRequest()	852
7.212.2.7 BootComRs232Init()	853
7.212.2.8 CanGetSpeedConfig()	853
7.212.2.9 Rs232ReceiveByte()	853
7.212.3 Variable Documentation	854
7.212.3.1 canTiming	854
7.213boot.c File Reference	855
7.213.1 Detailed Description	856
7.213.2 Function Documentation	856
7.213.2.1 BootActivate()	856
7.213.2.2 BootComCanCheckActivationRequest()	856
7.213.2.3 BootComCanInit()	857
7.213.2.4 BootComCheckActivationRequest()	857
7.213.2.5 BootComInit()	857
7.213.2.6 BootComRs232CheckActivationRequest()	857
7.213.2.7 BootComRs232Init()	858
7.213.2.8 CanGetSpeedConfig()	858
7.213.2.9 Rs232ReceiveByte()	858
7.213.3 Variable Documentation	859
7.213.3.1 canTiming	859
7.214boot.c File Reference	860
7.214.1 Detailed Description	861

7.214.2 Function Documentation	861
7.214.2.1 BootActivate()	861
7.214.2.2 BootComCanCheckActivationRequest()	861
7.214.2.3 BootComCanInit()	862
7.214.2.4 BootComCheckActivationRequest()	862
7.214.2.5 BootComInit()	862
7.214.2.6 BootComRs232CheckActivationRequest()	862
7.214.2.7 BootComRs232Init()	863
7.214.2.8 CanGetSpeedConfig()	863
7.214.2.9 Rs232ReceiveByte()	863
7.214.3 Variable Documentation	864
7.214.3.1 canTiming	864
7.215boot.c File Reference	865
7.215.1 Detailed Description	866
7.215.2 Function Documentation	866
7.215.2.1 BootActivate()	866
7.215.2.2 BootComCanCheckActivationRequest()	866
7.215.2.3 BootComCanInit()	867
7.215.2.4 BootComCheckActivationRequest()	867
7.215.2.5 BootComInit()	867
7.215.2.6 BootComRs232CheckActivationRequest()	867
7.215.2.7 BootComRs232Init()	868
7.215.2.8 CanGetSpeedConfig()	868
7.215.2.9 Rs232ReceiveByte()	868
7.215.3 Variable Documentation	869
7.215.3.1 canTiming	869
7.216boot.c File Reference	869
7.216.1 Detailed Description	870
7.216.2 Function Documentation	870
7.216.2.1 BootActivate()	870

7.216.2.2 BootComCheckActivationRequest()	871
7.216.2.3 BootComInit()	871
7.216.2.4 BootComRs232CheckActivationRequest()	871
7.216.2.5 BootComRs232Init()	871
7.216.2.6 Rs232ReceiveByte()	871
7.217boot.c File Reference	872
7.217.1 Detailed Description	873
7.217.2 Function Documentation	873
7.217.2.1 BootActivate()	873
7.217.2.2 BootComCanCheckActivationRequest()	873
7.217.2.3 BootComCanInit()	874
7.217.2.4 BootComCheckActivationRequest()	874
7.217.2.5 BootComInit()	874
7.217.2.6 BootComRs232CheckActivationRequest()	874
7.217.2.7 BootComRs232Init()	875
7.217.2.8 Rs232ReceiveByte()	875
7.218boot.c File Reference	875
7.218.1 Detailed Description	876
7.218.2 Function Documentation	877
7.218.2.1 BootActivate()	877
7.218.2.2 BootComCanCheckActivationRequest()	877
7.218.2.3 BootComCanInit()	877
7.218.2.4 BootComCheckActivationRequest()	878
7.218.2.5 BootComInit()	878
7.218.2.6 BootComRs232CheckActivationRequest()	878
7.218.2.7 BootComRs232Init()	878
7.218.2.8 Rs232ReceiveByte()	878
7.219boot.c File Reference	879
7.219.1 Detailed Description	880
7.219.2 Function Documentation	880

7.219.2.1 BootActivate()	880
7.219.2.2 BootComCanCheckActivationRequest()	881
7.219.2.3 BootComCanInit()	881
7.219.2.4 BootComCheckActivationRequest()	881
7.219.2.5 BootComInit()	882
7.219.2.6 BootComRs232CheckActivationRequest()	882
7.219.2.7 BootComRs232Init()	882
7.219.2.8 CanGetSpeedConfig()	882
7.219.2.9 Rs232ReceiveByte()	883
7.219.3 Variable Documentation	883
7.219.3.1 canTiming	883
7.220boot.c File Reference	884
7.220.1 Detailed Description	885
7.220.2 Function Documentation	885
7.220.2.1 BootActivate()	885
7.220.2.2 BootComCanCheckActivationRequest()	886
7.220.2.3 BootComCanInit()	886
7.220.2.4 BootComCheckActivationRequest()	886
7.220.2.5 BootComInit()	887
7.220.2.6 BootComRs232CheckActivationRequest()	887
7.220.2.7 BootComRs232Init()	887
7.220.2.8 CanGetSpeedConfig()	887
7.220.2.9 Rs232ReceiveByte()	888
7.220.3 Variable Documentation	888
7.220.3.1 canTiming	888
7.221boot.c File Reference	889
7.221.1 Detailed Description	890
7.221.2 Function Documentation	890
7.221.2.1 BootActivate()	890
7.221.2.2 BootComCanCheckActivationRequest()	891

7.221.2.3 BootComCanInit()	891
7.221.2.4 BootComCheckActivationRequest()	891
7.221.2.5 BootComInit()	892
7.221.2.6 BootComRs232CheckActivationRequest()	892
7.221.2.7 BootComRs232Init()	892
7.221.2.8 CanGetSpeedConfig()	892
7.221.2.9 Rs232ReceiveByte()	893
7.221.3 Variable Documentation	893
7.221.3.1 canTiming	893
7.222boot.c File Reference	894
7.222.1 Detailed Description	895
7.222.2 Function Documentation	895
7.222.2.1 BootActivate()	895
7.222.2.2 BootComCanCheckActivationRequest()	896
7.222.2.3 BootComCanInit()	896
7.222.2.4 BootComCheckActivationRequest()	896
7.222.2.5 BootComInit()	897
7.222.2.6 BootComRs232CheckActivationRequest()	897
7.222.2.7 BootComRs232Init()	897
7.222.2.8 CanGetSpeedConfig()	897
7.222.2.9 Rs232ReceiveByte()	898
7.222.3 Variable Documentation	898
7.222.3.1 canTiming	898
7.223boot.c File Reference	899
7.223.1 Detailed Description	900
7.223.2 Function Documentation	900
7.223.2.1 BootActivate()	900
7.223.2.2 BootComCheckActivationRequest()	901
7.223.2.3 BootComInit()	901
7.223.2.4 BootComRs232CheckActivationRequest()	901

7.223.2.5 BootComRs232Init()	901
7.223.2.6 Rs232ReceiveByte()	901
7.224boot.c File Reference	902
7.224.1 Detailed Description	903
7.224.2 Function Documentation	903
7.224.2.1 BootActivate()	903
7.224.2.2 BootComCheckActivationRequest()	903
7.224.2.3 BootComInit()	904
7.224.2.4 BootComRs232CheckActivationRequest()	904
7.224.2.5 BootComRs232Init()	904
7.224.2.6 Rs232ReceiveByte()	904
7.225boot.c File Reference	905
7.225.1 Detailed Description	906
7.225.2 Function Documentation	906
7.225.2.1 BootActivate()	906
7.225.2.2 BootComCheckActivationRequest()	906
7.225.2.3 BootComInit()	907
7.225.2.4 BootComRs232CheckActivationRequest()	907
7.225.2.5 BootComRs232Init()	907
7.225.2.6 Rs232ReceiveByte()	907
7.226boot.c File Reference	908
7.226.1 Detailed Description	909
7.226.2 Function Documentation	909
7.226.2.1 BootActivate()	909
7.226.2.2 BootComCheckActivationRequest()	909
7.226.2.3 BootComInit()	910
7.226.2.4 BootComRs232CheckActivationRequest()	910
7.226.2.5 BootComRs232Init()	910
7.226.2.6 Rs232ReceiveByte()	910
7.227boot.c File Reference	911

7.227.1 Detailed Description	912
7.227.2 Function Documentation	912
7.227.2.1 BootActivate()	912
7.227.2.2 BootComCanCheckActivationRequest()	913
7.227.2.3 BootComCanInit()	913
7.227.2.4 BootComCheckActivationRequest()	913
7.227.2.5 BootComInit()	914
7.227.2.6 BootComRs232CheckActivationRequest()	914
7.227.2.7 BootComRs232Init()	914
7.227.2.8 CanGetSpeedConfig()	914
7.227.2.9 Rs232ReceiveByte()	915
7.227.3 Variable Documentation	915
7.227.3.1 canTiming	915
7.228boot.c File Reference	916
7.228.1 Detailed Description	917
7.228.2 Function Documentation	917
7.228.2.1 BootActivate()	917
7.228.2.2 BootComCanCheckActivationRequest()	918
7.228.2.3 BootComCanInit()	918
7.228.2.4 BootComCheckActivationRequest()	918
7.228.2.5 BootComInit()	919
7.228.2.6 BootComRs232CheckActivationRequest()	919
7.228.2.7 BootComRs232Init()	919
7.228.2.8 CanGetSpeedConfig()	919
7.228.2.9 Rs232ReceiveByte()	920
7.228.3 Variable Documentation	920
7.228.3.1 canTiming	920
7.229boot.c File Reference	921
7.229.1 Detailed Description	922
7.229.2 Function Documentation	922

7.229.2.1 BootActivate()	922
7.229.2.2 BootComCanCheckActivationRequest()	923
7.229.2.3 BootComCanInit()	923
7.229.2.4 BootComCheckActivationRequest()	923
7.229.2.5 BootComInit()	924
7.229.2.6 BootComRs232CheckActivationRequest()	924
7.229.2.7 BootComRs232Init()	924
7.229.2.8 CanGetSpeedConfig()	924
7.229.2.9 Rs232ReceiveByte()	925
7.229.3 Variable Documentation	925
7.229.3.1 canTiming	925
7.230boot.c File Reference	926
7.230.1 Detailed Description	927
7.230.2 Function Documentation	927
7.230.2.1 BootActivate()	927
7.230.2.2 BootComCanCheckActivationRequest()	928
7.230.2.3 BootComCanInit()	928
7.230.2.4 BootComCheckActivationRequest()	928
7.230.2.5 BootComInit()	929
7.230.2.6 BootComRs232CheckActivationRequest()	929
7.230.2.7 BootComRs232Init()	929
7.230.2.8 CanGetSpeedConfig()	929
7.230.2.9 Rs232ReceiveByte()	930
7.230.3 Variable Documentation	930
7.230.3.1 canTiming	930
7.231boot.c File Reference	931
7.231.1 Detailed Description	932
7.231.2 Function Documentation	932
7.231.2.1 BootActivate()	932
7.231.2.2 BootComCanCheckActivationRequest()	933

7.231.2.3 BootComCanInit()	933
7.231.2.4 BootComCheckActivationRequest()	933
7.231.2.5 BootComInit()	934
7.231.2.6 BootComRs232CheckActivationRequest()	934
7.231.2.7 BootComRs232Init()	934
7.231.2.8 CanGetSpeedConfig()	934
7.231.2.9 Rs232ReceiveByte()	935
7.231.3 Variable Documentation	935
7.231.3.1 canTiming	935
7.232boot.c File Reference	936
7.232.1 Detailed Description	936
7.232.2 Function Documentation	937
7.232.2.1 BootInit()	937
7.232.2.2 BootTask()	937
7.233boot.h File Reference	937
7.233.1 Detailed Description	938
7.233.2 Function Documentation	938
7.233.2.1 BootActivate()	938
7.233.2.2 BootComCheckActivationRequest()	938
7.233.2.3 BootComInit()	939
7.234boot.h File Reference	939
7.234.1 Detailed Description	939
7.234.2 Function Documentation	940
7.234.2.1 BootActivate()	940
7.234.2.2 BootComCheckActivationRequest()	940
7.234.2.3 BootComInit()	940
7.235boot.h File Reference	941
7.235.1 Detailed Description	941
7.235.2 Function Documentation	941
7.235.2.1 BootActivate()	941

7.235.2.2 BootComCheckActivationRequest()	942
7.235.2.3 BootComInit()	942
7.236boot.h File Reference	942
7.236.1 Detailed Description	943
7.236.2 Function Documentation	943
7.236.2.1 BootActivate()	943
7.236.2.2 BootComCheckActivationRequest()	943
7.236.2.3 BootComInit()	944
7.237boot.h File Reference	944
7.237.1 Detailed Description	944
7.237.2 Function Documentation	945
7.237.2.1 BootActivate()	945
7.237.2.2 BootComCheckActivationRequest()	945
7.237.2.3 BootComInit()	945
7.238boot.h File Reference	946
7.238.1 Detailed Description	946
7.238.2 Function Documentation	946
7.238.2.1 BootActivate()	946
7.238.2.2 BootComCheckActivationRequest()	947
7.238.2.3 BootComInit()	947
7.239boot.h File Reference	947
7.239.1 Detailed Description	948
7.239.2 Function Documentation	948
7.239.2.1 BootActivate()	948
7.239.2.2 BootComCheckActivationRequest()	948
7.239.2.3 BootComInit()	949
7.240boot.h File Reference	949
7.240.1 Detailed Description	949
7.240.2 Function Documentation	950
7.240.2.1 BootActivate()	950

7.240.2.2 BootComCheckActivationRequest()	950
7.240.2.3 BootComInit()	950
7.241boot.h File Reference	951
7.241.1 Detailed Description	951
7.241.2 Function Documentation	951
7.241.2.1 BootActivate()	951
7.241.2.2 BootComCheckActivationRequest()	952
7.241.2.3 BootComInit()	952
7.242boot.h File Reference	952
7.242.1 Detailed Description	953
7.242.2 Function Documentation	953
7.242.2.1 BootActivate()	953
7.242.2.2 BootComCheckActivationRequest()	953
7.242.2.3 BootComInit()	954
7.243boot.h File Reference	954
7.243.1 Detailed Description	954
7.243.2 Function Documentation	955
7.243.2.1 BootActivate()	955
7.243.2.2 BootComCheckActivationRequest()	955
7.243.2.3 BootComInit()	955
7.244boot.h File Reference	956
7.244.1 Detailed Description	956
7.244.2 Function Documentation	956
7.244.2.1 BootActivate()	956
7.244.2.2 BootComCheckActivationRequest()	957
7.244.2.3 BootComInit()	957
7.245boot.h File Reference	957
7.245.1 Detailed Description	958
7.245.2 Function Documentation	958
7.245.2.1 BootActivate()	958

7.245.2.2 BootComCheckActivationRequest()	958
7.245.2.3 BootComInit()	959
7.246boot.h File Reference	959
7.246.1 Detailed Description	959
7.246.2 Function Documentation	960
7.246.2.1 BootActivate()	960
7.246.2.2 BootComCheckActivationRequest()	960
7.246.2.3 BootComInit()	960
7.247boot.h File Reference	961
7.247.1 Detailed Description	961
7.247.2 Function Documentation	961
7.247.2.1 BootActivate()	961
7.247.2.2 BootComCheckActivationRequest()	962
7.247.2.3 BootComInit()	962
7.248boot.h File Reference	962
7.248.1 Detailed Description	963
7.248.2 Function Documentation	963
7.248.2.1 BootActivate()	963
7.248.2.2 BootComCheckActivationRequest()	963
7.248.2.3 BootComInit()	964
7.249boot.h File Reference	964
7.249.1 Detailed Description	964
7.249.2 Function Documentation	965
7.249.2.1 BootActivate()	965
7.249.2.2 BootComCheckActivationRequest()	965
7.249.2.3 BootComInit()	965
7.250boot.h File Reference	966
7.250.1 Detailed Description	966
7.250.2 Function Documentation	966
7.250.2.1 BootActivate()	966

7.250.2.2 BootComCheckActivationRequest()	967
7.250.2.3 BootComInit()	967
7.251boot.h File Reference	967
7.251.1 Detailed Description	968
7.251.2 Function Documentation	968
7.251.2.1 BootActivate()	968
7.251.2.2 BootComCheckActivationRequest()	968
7.251.2.3 BootComInit()	969
7.252boot.h File Reference	969
7.252.1 Detailed Description	969
7.252.2 Function Documentation	969
7.252.2.1 BootActivate()	970
7.252.2.2 BootComCheckActivationRequest()	970
7.252.2.3 BootComInit()	970
7.253boot.h File Reference	970
7.253.1 Detailed Description	971
7.253.2 Function Documentation	971
7.253.2.1 BootActivate()	971
7.253.2.2 BootComCheckActivationRequest()	971
7.253.2.3 BootComInit()	972
7.254boot.h File Reference	972
7.254.1 Detailed Description	972
7.254.2 Function Documentation	972
7.254.2.1 BootActivate()	973
7.254.2.2 BootComCheckActivationRequest()	973
7.254.2.3 BootComInit()	973
7.255boot.h File Reference	973
7.255.1 Detailed Description	974
7.255.2 Function Documentation	974
7.255.2.1 BootActivate()	974

7.255.2.2 BootComCheckActivationRequest()	974
7.255.2.3 BootComInit()	975
7.256boot.h File Reference	975
7.256.1 Detailed Description	975
7.256.2 Function Documentation	975
7.256.2.1 BootActivate()	976
7.256.2.2 BootComCheckActivationRequest()	976
7.256.2.3 BootComInit()	976
7.257boot.h File Reference	976
7.257.1 Detailed Description	977
7.257.2 Function Documentation	977
7.257.2.1 BootActivate()	977
7.257.2.2 BootComCheckActivationRequest()	977
7.257.2.3 BootComInit()	978
7.258boot.h File Reference	978
7.258.1 Detailed Description	978
7.258.2 Function Documentation	979
7.258.2.1 BootActivate()	979
7.258.2.2 BootComCheckActivationRequest()	979
7.258.2.3 BootComInit()	979
7.259boot.h File Reference	980
7.259.1 Detailed Description	980
7.259.2 Function Documentation	980
7.259.2.1 BootActivate()	980
7.259.2.2 BootComCheckActivationRequest()	981
7.259.2.3 BootComInit()	981
7.260boot.h File Reference	981
7.260.1 Detailed Description	982
7.260.2 Function Documentation	982
7.260.2.1 BootActivate()	982

7.260.2.2 BootComCheckActivationRequest()	982
7.260.2.3 BootComInit()	983
7.261boot.h File Reference	983
7.261.1 Detailed Description	983
7.261.2 Function Documentation	984
7.261.2.1 BootActivate()	984
7.261.2.2 BootComCheckActivationRequest()	984
7.261.2.3 BootComInit()	984
7.262boot.h File Reference	985
7.262.1 Detailed Description	985
7.262.2 Function Documentation	985
7.262.2.1 BootActivate()	985
7.262.2.2 BootComCheckActivationRequest()	986
7.262.2.3 BootComInit()	986
7.263boot.h File Reference	986
7.263.1 Detailed Description	987
7.263.2 Function Documentation	987
7.263.2.1 BootActivate()	987
7.263.2.2 BootComCheckActivationRequest()	987
7.263.2.3 BootComInit()	988
7.264boot.h File Reference	988
7.264.1 Detailed Description	988
7.264.2 Function Documentation	989
7.264.2.1 BootActivate()	989
7.264.2.2 BootComCheckActivationRequest()	989
7.264.2.3 BootComInit()	989
7.265boot.h File Reference	990
7.265.1 Detailed Description	990
7.265.2 Function Documentation	990
7.265.2.1 BootActivate()	990

7.265.2.2 BootComCheckActivationRequest()	991
7.265.2.3 BootComInit()	991
7.266boot.h File Reference	991
7.266.1 Detailed Description	992
7.266.2 Function Documentation	992
7.266.2.1 BootActivate()	992
7.266.2.2 BootComCheckActivationRequest()	992
7.266.2.3 BootComInit()	993
7.267boot.h File Reference	993
7.267.1 Detailed Description	993
7.267.2 Function Documentation	994
7.267.2.1 BootActivate()	994
7.267.2.2 BootComCheckActivationRequest()	994
7.267.2.3 BootComInit()	994
7.268boot.h File Reference	995
7.268.1 Detailed Description	995
7.268.2 Function Documentation	995
7.268.2.1 BootActivate()	995
7.268.2.2 BootComCheckActivationRequest()	996
7.268.2.3 BootComInit()	996
7.269boot.h File Reference	996
7.269.1 Detailed Description	997
7.269.2 Function Documentation	997
7.269.2.1 BootActivate()	997
7.269.2.2 BootComCheckActivationRequest()	997
7.269.2.3 BootComInit()	998
7.270boot.h File Reference	998
7.270.1 Detailed Description	998
7.270.2 Function Documentation	999
7.270.2.1 BootActivate()	999

7.270.2.2 BootComCheckActivationRequest()	999
7.270.2.3 BootComInit()	999
7.271boot.h File Reference	1000
7.271.1 Detailed Description	1000
7.271.2 Function Documentation	1000
7.271.2.1 BootActivate()	1000
7.271.2.2 BootComCheckActivationRequest()	1001
7.271.2.3 BootComInit()	1001
7.272boot.h File Reference	1001
7.272.1 Detailed Description	1002
7.272.2 Function Documentation	1002
7.272.2.1 BootActivate()	1002
7.272.2.2 BootComCheckActivationRequest()	1002
7.272.2.3 BootComInit()	1003
7.273boot.h File Reference	1003
7.273.1 Detailed Description	1003
7.273.2 Function Documentation	1004
7.273.2.1 BootActivate()	1004
7.273.2.2 BootComCheckActivationRequest()	1004
7.273.2.3 BootComInit()	1004
7.274boot.h File Reference	1005
7.274.1 Detailed Description	1005
7.274.2 Function Documentation	1005
7.274.2.1 BootActivate()	1005
7.274.2.2 BootComCheckActivationRequest()	1006
7.274.2.3 BootComInit()	1006
7.275boot.h File Reference	1006
7.275.1 Detailed Description	1007
7.275.2 Function Documentation	1007
7.275.2.1 BootActivate()	1007

7.275.2.2 BootComCheckActivationRequest()	1007
7.275.2.3 BootComInit()	1008
7.276boot.h File Reference	1008
7.276.1 Detailed Description	1008
7.276.2 Function Documentation	1009
7.276.2.1 BootActivate()	1009
7.276.2.2 BootComCheckActivationRequest()	1009
7.276.2.3 BootComInit()	1009
7.277boot.h File Reference	1010
7.277.1 Detailed Description	1010
7.277.2 Function Documentation	1010
7.277.2.1 BootActivate()	1010
7.277.2.2 BootComCheckActivationRequest()	1011
7.277.2.3 BootComInit()	1011
7.278boot.h File Reference	1011
7.278.1 Detailed Description	1012
7.278.2 Function Documentation	1012
7.278.2.1 BootActivate()	1012
7.278.2.2 BootComCheckActivationRequest()	1012
7.278.2.3 BootComInit()	1013
7.279boot.h File Reference	1013
7.279.1 Detailed Description	1013
7.279.2 Function Documentation	1014
7.279.2.1 BootActivate()	1014
7.279.2.2 BootComCheckActivationRequest()	1014
7.279.2.3 BootComInit()	1014
7.280boot.h File Reference	1015
7.280.1 Detailed Description	1015
7.280.2 Function Documentation	1015
7.280.2.1 BootActivate()	1015

7.280.2.2 BootComCheckActivationRequest()	1016
7.280.2.3 BootComInit()	1016
7.281boot.h File Reference	1016
7.281.1 Detailed Description	1017
7.281.2 Function Documentation	1017
7.281.2.1 BootActivate()	1017
7.281.2.2 BootComCheckActivationRequest()	1017
7.281.2.3 BootComInit()	1018
7.282boot.h File Reference	1018
7.282.1 Detailed Description	1018
7.282.2 Function Documentation	1019
7.282.2.1 BootActivate()	1019
7.282.2.2 BootComCheckActivationRequest()	1019
7.282.2.3 BootComInit()	1019
7.283boot.h File Reference	1020
7.283.1 Detailed Description	1020
7.283.2 Function Documentation	1020
7.283.2.1 BootActivate()	1020
7.283.2.2 BootComCheckActivationRequest()	1021
7.283.2.3 BootComInit()	1021
7.284boot.h File Reference	1021
7.284.1 Detailed Description	1022
7.284.2 Function Documentation	1022
7.284.2.1 BootActivate()	1022
7.284.2.2 BootComCheckActivationRequest()	1022
7.284.2.3 BootComInit()	1023
7.285boot.h File Reference	1023
7.285.1 Detailed Description	1023
7.285.2 Function Documentation	1024
7.285.2.1 BootActivate()	1024

7.285.2.2 BootComCheckActivationRequest()	1024
7.285.2.3 BootComInit()	1024
7.286boot.h File Reference	1025
7.286.1 Detailed Description	1025
7.286.2 Function Documentation	1025
7.286.2.1 BootActivate()	1025
7.286.2.2 BootComCheckActivationRequest()	1026
7.286.2.3 BootComInit()	1026
7.287boot.h File Reference	1026
7.287.1 Detailed Description	1027
7.287.2 Function Documentation	1027
7.287.2.1 BootActivate()	1027
7.287.2.2 BootComCheckActivationRequest()	1027
7.287.2.3 BootComInit()	1028
7.288boot.h File Reference	1028
7.288.1 Detailed Description	1028
7.288.2 Function Documentation	1029
7.288.2.1 BootActivate()	1029
7.288.2.2 BootComCheckActivationRequest()	1029
7.288.2.3 BootComInit()	1029
7.289boot.h File Reference	1030
7.289.1 Detailed Description	1030
7.289.2 Function Documentation	1030
7.289.2.1 BootActivate()	1030
7.289.2.2 BootComCheckActivationRequest()	1031
7.289.2.3 BootComInit()	1031
7.290boot.h File Reference	1031
7.290.1 Detailed Description	1032
7.290.2 Function Documentation	1032
7.290.2.1 BootActivate()	1032

7.290.2.2 BootComCheckActivationRequest()	1032
7.290.2.3 BootComInit()	1033
7.291boot.h File Reference	1033
7.291.1 Detailed Description	1033
7.291.2 Function Documentation	1034
7.291.2.1 BootActivate()	1034
7.291.2.2 BootComCheckActivationRequest()	1034
7.291.2.3 BootComInit()	1034
7.292boot.h File Reference	1035
7.292.1 Detailed Description	1035
7.292.2 Function Documentation	1035
7.292.2.1 BootActivate()	1035
7.292.2.2 BootComCheckActivationRequest()	1036
7.292.2.3 BootComInit()	1036
7.293boot.h File Reference	1036
7.293.1 Detailed Description	1037
7.293.2 Function Documentation	1037
7.293.2.1 BootActivate()	1037
7.293.2.2 BootComCheckActivationRequest()	1037
7.293.2.3 BootComInit()	1038
7.294boot.h File Reference	1038
7.294.1 Detailed Description	1038
7.294.2 Function Documentation	1039
7.294.2.1 BootActivate()	1039
7.294.2.2 BootComCheckActivationRequest()	1039
7.294.2.3 BootComInit()	1039
7.295boot.h File Reference	1040
7.295.1 Detailed Description	1040
7.295.2 Function Documentation	1040
7.295.2.1 BootActivate()	1040

7.295.2.2 BootComCheckActivationRequest()	1041
7.295.2.3 BootComInit()	1041
7.296boot.h File Reference	1041
7.296.1 Detailed Description	1042
7.296.2 Function Documentation	1042
7.296.2.1 BootActivate()	1042
7.296.2.2 BootComCheckActivationRequest()	1042
7.296.2.3 BootComInit()	1043
7.297boot.h File Reference	1043
7.297.1 Detailed Description	1043
7.297.2 Function Documentation	1044
7.297.2.1 BootActivate()	1044
7.297.2.2 BootComCheckActivationRequest()	1044
7.297.2.3 BootComInit()	1044
7.298boot.h File Reference	1045
7.298.1 Detailed Description	1045
7.298.2 Function Documentation	1045
7.298.2.1 BootActivate()	1045
7.298.2.2 BootComCheckActivationRequest()	1046
7.298.2.3 BootComInit()	1046
7.299boot.h File Reference	1046
7.299.1 Detailed Description	1047
7.299.2 Function Documentation	1047
7.299.2.1 BootActivate()	1047
7.299.2.2 BootComCheckActivationRequest()	1047
7.299.2.3 BootComInit()	1048
7.300boot.h File Reference	1048
7.300.1 Detailed Description	1048
7.300.2 Function Documentation	1049
7.300.2.1 BootActivate()	1049

7.300.2.2 BootComCheckActivationRequest()	1049
7.300.2.3 BootComInit()	1049
7.301boot.h File Reference	1050
7.301.1 Detailed Description	1050
7.301.2 Function Documentation	1050
7.301.2.1 BootActivate()	1050
7.301.2.2 BootComCheckActivationRequest()	1051
7.301.2.3 BootComInit()	1051
7.302boot.h File Reference	1051
7.302.1 Detailed Description	1052
7.302.2 Function Documentation	1052
7.302.2.1 BootActivate()	1052
7.302.2.2 BootComCheckActivationRequest()	1052
7.302.2.3 BootComInit()	1053
7.303boot.h File Reference	1053
7.303.1 Detailed Description	1053
7.303.2 Function Documentation	1054
7.303.2.1 BootActivate()	1054
7.303.2.2 BootComCheckActivationRequest()	1054
7.303.2.3 BootComInit()	1054
7.304boot.h File Reference	1055
7.304.1 Detailed Description	1055
7.304.2 Function Documentation	1055
7.304.2.1 BootActivate()	1055
7.304.2.2 BootComCheckActivationRequest()	1056
7.304.2.3 BootComInit()	1056
7.305boot.h File Reference	1056
7.305.1 Detailed Description	1057
7.305.2 Function Documentation	1057
7.305.2.1 BootActivate()	1057

7.305.2.2 BootComCheckActivationRequest()	1057
7.305.2.3 BootComInit()	1058
7.306boot.h File Reference	1058
7.306.1 Detailed Description	1058
7.306.2 Function Documentation	1059
7.306.2.1 BootActivate()	1059
7.306.2.2 BootComCheckActivationRequest()	1059
7.306.2.3 BootComInit()	1059
7.307boot.h File Reference	1060
7.307.1 Detailed Description	1060
7.307.2 Function Documentation	1060
7.307.2.1 BootActivate()	1060
7.307.2.2 BootComCheckActivationRequest()	1061
7.307.2.3 BootComInit()	1061
7.308boot.h File Reference	1061
7.308.1 Detailed Description	1062
7.308.2 Function Documentation	1062
7.308.2.1 BootInit()	1062
7.308.2.2 BootTask()	1062
7.309can.c File Reference	1063
7.309.1 Detailed Description	1064
7.309.2 Function Documentation	1064
7.309.2.1 CanGetSpeedConfig()	1064
7.309.2.2 CanInit()	1064
7.309.2.3 CanReceivePacket()	1064
7.309.2.4 CanTransmitPacket()	1065
7.309.3 Variable Documentation	1065
7.309.3.1 canTiming	1065
7.310can.c File Reference	1066
7.310.1 Detailed Description	1067

7.310.2 Function Documentation	1068
7.310.2.1 CanDisabledModeEnter()	1068
7.310.2.2 CanDisabledModeExit()	1068
7.310.2.3 CanFreezeModeEnter()	1068
7.310.2.4 CanFreezeModeExit()	1069
7.310.2.5 CanGetSpeedConfig()	1069
7.310.2.6 CanInit()	1069
7.310.2.7 CanReceivePacket()	1070
7.310.2.8 CanTransmitPacket()	1070
7.310.3 Variable Documentation	1070
7.310.3.1 canTiming	1070
7.311 can.c File Reference	1071
7.311.1 Detailed Description	1072
7.311.2 Function Documentation	1072
7.311.2.1 CanGetSpeedConfig()	1072
7.311.2.2 CanInit()	1073
7.311.2.3 CanReceivePacket()	1073
7.311.2.4 CanTransmitPacket()	1074
7.311.3 Variable Documentation	1074
7.311.3.1 canTiming	1074
7.312 can.c File Reference	1075
7.312.1 Detailed Description	1076
7.312.2 Function Documentation	1076
7.312.2.1 CanInit()	1076
7.312.2.2 CanReceivePacket()	1076
7.312.2.3 CanTransmitPacket()	1077
7.313 can.c File Reference	1077
7.313.1 Detailed Description	1078
7.313.2 Function Documentation	1078
7.313.2.1 CanGetSpeedConfig()	1078

7.313.2.2 CanInit()	1079
7.313.2.3 CanReceivePacket()	1079
7.313.2.4 CanTransmitPacket()	1079
7.313.3 Variable Documentation	1080
7.313.3.1 canTiming	1080
7.314can.c File Reference	1081
7.314.1 Detailed Description	1081
7.314.2 Function Documentation	1082
7.314.2.1 CanInit()	1082
7.314.2.2 CanReceivePacket()	1082
7.314.2.3 CanSetBittiming()	1082
7.314.2.4 CanTransmitPacket()	1083
7.315can.c File Reference	1083
7.315.1 Detailed Description	1084
7.315.2 Function Documentation	1084
7.315.2.1 CanGetSpeedConfig()	1084
7.315.2.2 CanInit()	1085
7.315.2.3 CanReceivePacket()	1085
7.315.2.4 CanTransmitPacket()	1085
7.315.3 Variable Documentation	1086
7.315.3.1 canTiming	1086
7.316can.c File Reference	1086
7.316.1 Detailed Description	1088
7.316.2 Function Documentation	1088
7.316.2.1 CanGetSpeedConfig()	1088
7.316.2.2 CanInit()	1088
7.316.2.3 CanReceivePacket()	1088
7.316.2.4 CanTransmitPacket()	1089
7.316.3 Variable Documentation	1089
7.316.3.1 canTiming	1089

7.317can.c File Reference	1090
7.317.1 Detailed Description	1091
7.317.2 Function Documentation	1091
7.317.2.1 CanDisabledModeEnter()	1092
7.317.2.2 CanDisabledModeExit()	1092
7.317.2.3 CanFreezeModeEnter()	1092
7.317.2.4 CanFreezeModeExit()	1093
7.317.2.5 CanGetSpeedConfig()	1093
7.317.2.6 CanInit()	1093
7.317.2.7 CanReceivePacket()	1094
7.317.2.8 CanTransmitPacket()	1094
7.317.3 Variable Documentation	1094
7.317.3.1 canTiming	1094
7.318can.c File Reference	1095
7.318.1 Detailed Description	1096
7.318.2 Function Documentation	1096
7.318.2.1 CanGetSpeedConfig()	1096
7.318.2.2 CanInit()	1097
7.318.2.3 CanReceivePacket()	1097
7.318.2.4 CanTransmitPacket()	1098
7.318.3 Variable Documentation	1098
7.318.3.1 canTiming	1098
7.319can.c File Reference	1099
7.319.1 Detailed Description	1100
7.319.2 Function Documentation	1100
7.319.2.1 CanGetSpeedConfig()	1100
7.319.2.2 CanInit()	1100
7.319.2.3 CanReceivePacket()	1100
7.319.2.4 CanTransmitPacket()	1101
7.319.3 Variable Documentation	1101

7.319.3.1 canTiming	1101
7.320can.c File Reference	1102
7.320.1 Detailed Description	1103
7.320.2 Function Documentation	1103
7.320.2.1 CanGetSpeedConfig()	1103
7.320.2.2 CanInit()	1104
7.320.2.3 CanReceivePacket()	1104
7.320.2.4 CanTransmitPacket()	1104
7.320.3 Variable Documentation	1105
7.320.3.1 canTiming	1105
7.321can.c File Reference	1105
7.321.1 Detailed Description	1107
7.321.2 Function Documentation	1107
7.321.2.1 CanInit()	1107
7.321.2.2 CanReceivePacket()	1107
7.321.2.3 CanTransmitPacket()	1107
7.322can.c File Reference	1108
7.322.1 Detailed Description	1109
7.322.2 Function Documentation	1109
7.322.2.1 CanGetSpeedConfig()	1109
7.322.2.2 CanInit()	1110
7.322.2.3 CanReceivePacket()	1110
7.322.2.4 CanTransmitPacket()	1110
7.322.3 Variable Documentation	1111
7.322.3.1 canTiming	1111
7.323can.c File Reference	1111
7.323.1 Detailed Description	1113
7.323.2 Function Documentation	1113
7.323.2.1 CanGetSpeedConfig()	1113
7.323.2.2 CanInit()	1113

7.323.2.3 CanReceivePacket()	1113
7.323.2.4 CanTransmitPacket()	1114
7.323.3 Variable Documentation	1114
7.323.3.1 canTiming	1114
7.324can.c File Reference	1115
7.324.1 Detailed Description	1116
7.324.2 Function Documentation	1117
7.324.2.1 CanGetSpeedConfig()	1117
7.324.2.2 CanInit()	1117
7.324.2.3 CanReceivePacket()	1117
7.324.2.4 CanTransmitPacket()	1118
7.324.3 Variable Documentation	1118
7.324.3.1 canTiming	1118
7.325can.h File Reference	1119
7.325.1 Detailed Description	1119
7.325.2 Function Documentation	1120
7.325.2.1 CanInit()	1120
7.325.2.2 CanReceivePacket()	1120
7.325.2.3 CanTransmitPacket()	1120
7.326com.c File Reference	1121
7.326.1 Detailed Description	1122
7.326.2 Function Documentation	1122
7.326.2.1 ComFree()	1122
7.326.2.2 ComGetActiveInterfaceMaxRxLen()	1122
7.326.2.3 ComGetActiveInterfaceMaxTxLen()	1122
7.326.2.4 ComInit()	1123
7.326.2.5 ComIsConnected()	1123
7.326.2.6 ComTask()	1123
7.326.2.7 ComTransmitPacket()	1123
7.327com.h File Reference	1124

7.327.1 Detailed Description	1125
7.327.2 Enumeration Type Documentation	1125
7.327.2.1 tComInterfaceld	1125
7.327.3 Function Documentation	1126
7.327.3.1 ComFree()	1126
7.327.3.2 ComGetActiveInterfaceMaxRxLen()	1126
7.327.3.3 ComGetActiveInterfaceMaxTxLen()	1126
7.327.3.4 ComInit()	1127
7.327.3.5 ComIsConnected()	1127
7.327.3.6 ComTask()	1127
7.327.3.7 ComTransmitPacket()	1127
7.328cop.c File Reference	1128
7.328.1 Detailed Description	1128
7.328.2 Function Documentation	1129
7.328.2.1 CopInit()	1129
7.328.2.2 CopInitHook()	1129
7.328.2.3 CopService()	1129
7.328.2.4 CopServiceHook()	1130
7.329cop.h File Reference	1130
7.329.1 Detailed Description	1130
7.329.2 Function Documentation	1130
7.329.2.1 CopInit()	1130
7.329.2.2 CopService()	1131
7.330cpu.c File Reference	1131
7.330.1 Detailed Description	1132
7.330.2 Function Documentation	1132
7.330.2.1 CpuInit()	1132
7.330.2.2 CpuMemCopy()	1132
7.330.2.3 CpuMemSet()	1133
7.330.2.4 CpuStartUserProgram()	1133

7.330.2.5 CpuUserProgramStartHook()	1133
7.331cpu.c File Reference	1134
7.331.1 Detailed Description	1134
7.331.2 Function Documentation	1135
7.331.2.1 CpuInit()	1135
7.331.2.2 CpuMemCopy()	1135
7.331.2.3 CpuMemSet()	1135
7.331.2.4 CpuStartUserProgram()	1136
7.331.2.5 CpuUserProgramStartHook()	1136
7.332cpu.c File Reference	1137
7.332.1 Detailed Description	1137
7.332.2 Function Documentation	1138
7.332.2.1 CpuInit()	1138
7.332.2.2 CpuMemCopy()	1138
7.332.2.3 CpuMemSet()	1138
7.332.2.4 CpuStartUserProgram()	1139
7.332.2.5 CpuUserProgramStartHook()	1139
7.333cpu.c File Reference	1140
7.333.1 Detailed Description	1140
7.333.2 Function Documentation	1141
7.333.2.1 CpuInit()	1141
7.333.2.2 CpuMemCopy()	1141
7.333.2.3 CpuMemSet()	1141
7.333.2.4 CpuStartUserProgram()	1142
7.333.2.5 CpuUserProgramStartHook()	1142
7.334cpu.c File Reference	1143
7.334.1 Detailed Description	1143
7.334.2 Function Documentation	1144
7.334.2.1 CpuInit()	1144
7.334.2.2 CpuMemCopy()	1144

7.334.2.3 CpuMemSet()	1144
7.334.2.4 CpuStartUserProgram()	1145
7.334.2.5 CpuUserProgramStartHook()	1145
7.335cpu.c File Reference	1146
7.335.1 Detailed Description	1146
7.335.2 Function Documentation	1147
7.335.2.1 CpuInit()	1147
7.335.2.2 CpuMemCopy()	1147
7.335.2.3 CpuMemSet()	1147
7.335.2.4 CpuStartUserProgram()	1148
7.335.2.5 CpuUserProgramStartHook()	1148
7.336cpu.c File Reference	1149
7.336.1 Detailed Description	1149
7.336.2 Function Documentation	1150
7.336.2.1 CpuInit()	1150
7.336.2.2 CpuMemCopy()	1150
7.336.2.3 CpuMemSet()	1150
7.336.2.4 CpuStartUserProgram()	1151
7.336.2.5 CpuUserProgramStartHook()	1151
7.337cpu.c File Reference	1152
7.337.1 Detailed Description	1152
7.337.2 Function Documentation	1153
7.337.2.1 CpuInit()	1153
7.337.2.2 CpuMemCopy()	1153
7.337.2.3 CpuMemSet()	1153
7.337.2.4 CpuStartUserProgram()	1154
7.337.2.5 CpuUserProgramStartHook()	1154
7.338cpu.c File Reference	1155
7.338.1 Detailed Description	1155
7.338.2 Function Documentation	1156

7.338.2.1 CpuInit()	1156
7.338.2.2 CpuMemCopy()	1156
7.338.2.3 CpuMemSet()	1156
7.338.2.4 CpuStartUserProgram()	1157
7.338.2.5 CpuUserProgramStartHook()	1157
7.339cpu.c File Reference	1158
7.339.1 Detailed Description	1158
7.339.2 Function Documentation	1159
7.339.2.1 CpuInit()	1159
7.339.2.2 CpuMemCopy()	1159
7.339.2.3 CpuMemSet()	1159
7.339.2.4 CpuStartUserProgram()	1160
7.339.2.5 CpuUserProgramStartHook()	1160
7.340cpu.c File Reference	1161
7.340.1 Detailed Description	1161
7.340.2 Function Documentation	1162
7.340.2.1 CpuInit()	1162
7.340.2.2 CpuMemCopy()	1162
7.340.2.3 CpuMemSet()	1162
7.340.2.4 CpuStartUserProgram()	1163
7.340.2.5 CpuUserProgramStartHook()	1163
7.341cpu.c File Reference	1164
7.341.1 Detailed Description	1164
7.341.2 Function Documentation	1165
7.341.2.1 CpuInit()	1165
7.341.2.2 CpuMemCopy()	1165
7.341.2.3 CpuMemSet()	1165
7.341.2.4 CpuStartUserProgram()	1166
7.341.2.5 CpuUserProgramStartHook()	1166
7.342cpu.c File Reference	1167

7.342.1 Detailed Description	1167
7.342.2 Function Documentation	1168
7.342.2.1 CpuInit()	1168
7.342.2.2 CpuMemCopy()	1168
7.342.2.3 CpuMemSet()	1168
7.342.2.4 CpuStartUserProgram()	1169
7.342.2.5 CpuUserProgramStartHook()	1169
7.343cpu.c File Reference	1170
7.343.1 Detailed Description	1170
7.343.2 Function Documentation	1171
7.343.2.1 CpuInit()	1171
7.343.2.2 CpuMemCopy()	1171
7.343.2.3 CpuMemSet()	1171
7.343.2.4 CpuStartUserProgram()	1172
7.343.2.5 CpuUserProgramStartHook()	1172
7.344cpu.c File Reference	1173
7.344.1 Detailed Description	1173
7.344.2 Function Documentation	1174
7.344.2.1 CpuInit()	1174
7.344.2.2 CpuMemCopy()	1174
7.344.2.3 CpuMemSet()	1174
7.344.2.4 CpuStartUserProgram()	1175
7.344.2.5 CpuUserProgramStartHook()	1175
7.345cpu.c File Reference	1176
7.345.1 Detailed Description	1176
7.345.2 Function Documentation	1177
7.345.2.1 CpuInit()	1177
7.345.2.2 CpuMemCopy()	1177
7.345.2.3 CpuMemSet()	1177
7.345.2.4 CpuStartUserProgram()	1178

7.345.2.5 CpuUserProgramStartHook()	1178
7.346cpu.c File Reference	1179
7.346.1 Detailed Description	1179
7.346.2 Function Documentation	1180
7.346.2.1 CpuInit()	1180
7.346.2.2 CpuMemCopy()	1180
7.346.2.3 CpuMemSet()	1180
7.346.2.4 CpuStartUserProgram()	1181
7.346.2.5 CpuUserProgramStartHook()	1181
7.347cpu.c File Reference	1182
7.347.1 Detailed Description	1182
7.347.2 Function Documentation	1183
7.347.2.1 CpuInit()	1183
7.347.2.2 CpuMemCopy()	1183
7.347.2.3 CpuMemSet()	1183
7.347.2.4 CpuStartUserProgram()	1184
7.347.2.5 CpuUserProgramStartHook()	1184
7.348cpu.c File Reference	1185
7.348.1 Detailed Description	1185
7.348.2 Macro Definition Documentation	1185
7.348.2.1 CPU_USER_PROGRAM_STARTADDR_PTR	1186
7.348.3 Function Documentation	1186
7.348.3.1 CpuInit()	1186
7.348.3.2 CpuMemCopy()	1186
7.348.3.3 CpuMemSet()	1187
7.348.3.4 CpuStartUserProgram()	1187
7.348.3.5 CpuUserProgramStartHook()	1187
7.349cpu.h File Reference	1188
7.349.1 Detailed Description	1188
7.349.2 Function Documentation	1188

7.349.2.1 CpuInit()	1188
7.349.2.2 CpuIrqDisable()	1189
7.349.2.3 CpuIrqEnable()	1189
7.349.2.4 CpuMemCopy()	1189
7.349.2.5 CpuMemSet()	1190
7.349.2.6 CpuStartUserProgram()	1190
7.350cpu_comp.c File Reference	1190
7.350.1 Detailed Description	1191
7.350.2 Function Documentation	1191
7.350.2.1 CpuIrqDisable()	1191
7.350.2.2 CpuIrqEnable()	1192
7.351cpu_comp.c File Reference	1192
7.351.1 Detailed Description	1192
7.351.2 Function Documentation	1192
7.351.2.1 CpuIrqDisable()	1193
7.351.2.2 CpuIrqEnable()	1193
7.352cpu_comp.c File Reference	1193
7.352.1 Detailed Description	1194
7.352.2 Function Documentation	1194
7.352.2.1 CpuIrqDisable()	1194
7.352.2.2 CpuIrqEnable()	1194
7.353cpu_comp.c File Reference	1194
7.353.1 Detailed Description	1195
7.353.2 Function Documentation	1195
7.353.2.1 CpuIrqDisable()	1195
7.353.2.2 CpuIrqEnable()	1195
7.354cpu_comp.c File Reference	1196
7.354.1 Detailed Description	1196
7.354.2 Function Documentation	1196
7.354.2.1 CpuIrqDisable()	1196

7.354.2.2 CpuIrqEnable()	1197
7.355cpu_comp.c File Reference	1197
7.355.1 Detailed Description	1197
7.355.2 Function Documentation	1197
7.355.2.1 CpuIrqDisable()	1198
7.355.2.2 CpuIrqEnable()	1198
7.356cpu_comp.c File Reference	1198
7.356.1 Detailed Description	1199
7.356.2 Function Documentation	1199
7.356.2.1 CpuIrqDisable()	1199
7.356.2.2 CpuIrqEnable()	1199
7.357cpu_comp.c File Reference	1199
7.357.1 Detailed Description	1200
7.357.2 Function Documentation	1200
7.357.2.1 CpuIrqDisable()	1200
7.357.2.2 CpuIrqEnable()	1200
7.358cpu_comp.c File Reference	1201
7.358.1 Detailed Description	1201
7.358.2 Function Documentation	1201
7.358.2.1 CpuIrqDisable()	1201
7.358.2.2 CpuIrqEnable()	1202
7.359cpu_comp.c File Reference	1202
7.359.1 Detailed Description	1202
7.359.2 Function Documentation	1202
7.359.2.1 CpuIrqDisable()	1203
7.359.2.2 CpuIrqEnable()	1203
7.360cpu_comp.c File Reference	1203
7.360.1 Detailed Description	1204
7.360.2 Function Documentation	1204
7.360.2.1 CpuIrqDisable()	1204

7.360.2.2 CpuIrqEnable()	1204
7.361cpu_comp.c File Reference	1204
7.361.1 Detailed Description	1205
7.361.2 Function Documentation	1205
7.361.2.1 CpuIrqDisable()	1205
7.361.2.2 CpuIrqEnable()	1205
7.362cpu_comp.c File Reference	1206
7.362.1 Detailed Description	1206
7.362.2 Function Documentation	1206
7.362.2.1 CpuIrqDisable()	1206
7.362.2.2 CpuIrqEnable()	1207
7.363cpu_comp.c File Reference	1207
7.363.1 Detailed Description	1207
7.363.2 Function Documentation	1207
7.363.2.1 CpuIrqDisable()	1208
7.363.2.2 CpuIrqEnable()	1208
7.364cpu_comp.c File Reference	1208
7.364.1 Detailed Description	1209
7.364.2 Function Documentation	1209
7.364.2.1 CpuIrqDisable()	1209
7.364.2.2 CpuIrqEnable()	1209
7.365cpu_comp.c File Reference	1209
7.365.1 Detailed Description	1210
7.365.2 Function Documentation	1210
7.365.2.1 CpuIrqDisable()	1210
7.365.2.2 CpuIrqEnable()	1210
7.366cpu_comp.c File Reference	1211
7.366.1 Detailed Description	1211
7.366.2 Function Documentation	1211
7.366.2.1 CpuIrqDisable()	1211

7.366.2.2 CpuIrqEnable()	1212
7.367cpu_comp.c File Reference	1212
7.367.1 Detailed Description	1212
7.367.2 Function Documentation	1212
7.367.2.1 CpuIrqDisable()	1213
7.367.2.2 CpuIrqEnable()	1213
7.368cpu_comp.c File Reference	1213
7.368.1 Detailed Description	1214
7.368.2 Function Documentation	1214
7.368.2.1 CpuIrqDisable()	1214
7.368.2.2 CpuIrqEnable()	1214
7.369cpu_comp.c File Reference	1214
7.369.1 Detailed Description	1215
7.369.2 Function Documentation	1215
7.369.2.1 CpuIrqDisable()	1215
7.369.2.2 CpuIrqEnable()	1215
7.370cpu_comp.c File Reference	1216
7.370.1 Detailed Description	1216
7.370.2 Function Documentation	1216
7.370.2.1 CpuIrqDisable()	1216
7.370.2.2 CpuIrqEnable()	1217
7.371cpu_comp.c File Reference	1217
7.371.1 Detailed Description	1217
7.371.2 Function Documentation	1217
7.371.2.1 CpuIrqDisable()	1218
7.371.2.2 CpuIrqEnable()	1218
7.372cpu_comp.c File Reference	1218
7.372.1 Detailed Description	1219
7.372.2 Function Documentation	1219
7.372.2.1 CpuIrqDisable()	1219

7.372.2.2 CpulrqEnable()	1219
7.373cpu_comp.c File Reference	1219
7.373.1 Detailed Description	1220
7.373.2 Function Documentation	1220
7.373.2.1 CpulrqDisable()	1220
7.373.2.2 CpulrqEnable()	1220
7.374cpu_comp.c File Reference	1221
7.374.1 Detailed Description	1221
7.374.2 Function Documentation	1221
7.374.2.1 CpulrqDisable()	1221
7.374.2.2 CpulrqEnable()	1222
7.375cpu_comp.c File Reference	1222
7.375.1 Detailed Description	1222
7.375.2 Function Documentation	1222
7.375.2.1 CpulrqDisable()	1223
7.375.2.2 CpulrqEnable()	1223
7.376cpu_comp.c File Reference	1223
7.376.1 Detailed Description	1224
7.376.2 Function Documentation	1224
7.376.2.1 CpulrqDisable()	1224
7.376.2.2 CpulrqEnable()	1224
7.377cpu_comp.c File Reference	1224
7.377.1 Detailed Description	1225
7.377.2 Function Documentation	1225
7.377.2.1 CpulrqDisable()	1225
7.377.2.2 CpulrqEnable()	1225
7.378cpu_comp.c File Reference	1226
7.378.1 Detailed Description	1226
7.378.2 Function Documentation	1226
7.378.2.1 CpulrqDisable()	1226

7.378.2.2 CpulrqEnable()	1227
7.379cpu_comp.c File Reference	1227
7.379.1 Detailed Description	1227
7.379.2 Function Documentation	1227
7.379.2.1 CpulrqDisable()	1228
7.379.2.2 CpulrqEnable()	1228
7.380cpu_comp.c File Reference	1228
7.380.1 Detailed Description	1229
7.380.2 Function Documentation	1229
7.380.2.1 CpulrqDisable()	1229
7.380.2.2 CpulrqEnable()	1229
7.381cpu_comp.c File Reference	1229
7.381.1 Detailed Description	1230
7.381.2 Function Documentation	1230
7.381.2.1 CpulrqDisable()	1230
7.381.2.2 CpulrqEnable()	1230
7.382cpu_comp.c File Reference	1231
7.382.1 Detailed Description	1231
7.382.2 Function Documentation	1231
7.382.2.1 CpulrqDisable()	1231
7.382.2.2 CpulrqEnable()	1232
7.383cpu_comp.c File Reference	1232
7.383.1 Detailed Description	1232
7.383.2 Function Documentation	1232
7.383.2.1 CpulrqDisable()	1233
7.383.2.2 CpulrqEnable()	1233
7.384cpu_comp.c File Reference	1233
7.384.1 Detailed Description	1234
7.384.2 Function Documentation	1234
7.384.2.1 CpulrqDisable()	1234

7.384.2.2 CpuIrqEnable()	1234
7.385cpu_comp.c File Reference	1234
7.385.1 Detailed Description	1235
7.385.2 Function Documentation	1235
7.385.2.1 CpuIrqDisable()	1235
7.385.2.2 CpuIrqEnable()	1235
7.386cpu_comp.c File Reference	1236
7.386.1 Detailed Description	1236
7.386.2 Function Documentation	1236
7.386.2.1 CpuIrqDisable()	1236
7.386.2.2 CpuIrqEnable()	1237
7.387cpu_comp.c File Reference	1237
7.387.1 Detailed Description	1237
7.387.2 Function Documentation	1237
7.387.2.1 CpuIrqDisable()	1238
7.387.2.2 CpuIrqEnable()	1238
7.388cpu_comp.c File Reference	1238
7.388.1 Detailed Description	1239
7.388.2 Function Documentation	1239
7.388.2.1 CpuIrqDisable()	1239
7.388.2.2 CpuIrqEnable()	1239
7.389cpu_comp.c File Reference	1239
7.389.1 Detailed Description	1240
7.389.2 Function Documentation	1240
7.389.2.1 CpuIrqDisable()	1240
7.389.2.2 CpuIrqEnable()	1240
7.390cpu_comp.c File Reference	1241
7.390.1 Detailed Description	1241
7.390.2 Function Documentation	1241
7.390.2.1 CpuIrqDisable()	1241

7.390.2.2 CpulrqEnable()	1242
7.391cpu_comp.c File Reference	1242
7.391.1 Detailed Description	1242
7.391.2 Function Documentation	1242
7.391.2.1 CpulrqDisable()	1243
7.391.2.2 CpulrqEnable()	1243
7.392cpu_comp.c File Reference	1243
7.392.1 Detailed Description	1244
7.392.2 Function Documentation	1244
7.392.2.1 CpulrqDisable()	1244
7.392.2.2 CpulrqEnable()	1244
7.393cpu_comp.c File Reference	1244
7.393.1 Detailed Description	1245
7.393.2 Function Documentation	1245
7.393.2.1 CpulrqDisable()	1245
7.393.2.2 CpulrqEnable()	1245
7.394cpu_comp.c File Reference	1246
7.394.1 Detailed Description	1246
7.394.2 Function Documentation	1246
7.394.2.1 CpulrqDisable()	1246
7.394.2.2 CpulrqEnable()	1247
7.395cstart.c File Reference	1247
7.395.1 Detailed Description	1247
7.395.2 Function Documentation	1247
7.395.2.1 main()	1248
7.395.2.2 reset_handler()	1248
7.396cstart.c File Reference	1248
7.396.1 Detailed Description	1249
7.396.2 Function Documentation	1249
7.396.2.1 main()	1249

7.396.2.2 reset_handler()	1249
7.397cstart.c File Reference	1249
7.397.1 Detailed Description	1250
7.397.2 Function Documentation	1250
7.397.2.1 main()	1250
7.397.2.2 reset_handler()	1250
7.398cstart.c File Reference	1251
7.398.1 Detailed Description	1251
7.398.2 Function Documentation	1251
7.398.2.1 main()	1251
7.398.2.2 reset_handler()	1252
7.399cstart.c File Reference	1252
7.399.1 Detailed Description	1252
7.399.2 Function Documentation	1252
7.399.2.1 main()	1253
7.399.2.2 reset_handler()	1253
7.400cstart.c File Reference	1253
7.400.1 Detailed Description	1254
7.400.2 Function Documentation	1254
7.400.2.1 main()	1254
7.400.2.2 reset_handler()	1254
7.401file.c File Reference	1254
7.401.1 Detailed Description	1256
7.401.2 Enumeration Type Documentation	1256
7.401.2.1 tFirmwareUpdateState	1256
7.401.3 Function Documentation	1256
7.401.3.1 FileFirmwareUpdateCompletedHook()	1257
7.401.3.2 FileFirmwareUpdateErrorHook()	1257
7.401.3.3 FileFirmwareUpdateLogHook()	1257
7.401.3.4 FileFirmwareUpdateStartedHook()	1258

7.401.3.5 FileGetFirmwareFilenameHook()	1258
7.401.3.6 FileHandleFirmwareUpdateRequest()	1258
7.401.3.7 FileInit()	1259
7.401.3.8 FileIsFirmwareUpdateRequestedHook()	1259
7.401.3.9 FileIsIdle()	1259
7.401.3.10FileLibByteNibbleToChar()	1259
7.401.3.11FileLibByteToHexString()	1260
7.401.3.12FileLibHexStringToByte()	1260
7.401.3.13FileLibLongToIntString()	1261
7.401.3.14FileSrecGetLineType()	1261
7.401.3.15FileSrecParseLine()	1262
7.401.3.16FileSrecVerifyChecksum()	1262
7.401.3.17FileTask()	1262
7.402file.h File Reference	1263
7.402.1 Detailed Description	1264
7.402.2 Enumeration Type Documentation	1264
7.402.2.1 tSrecLineType	1264
7.402.3 Function Documentation	1265
7.402.3.1 FileHandleFirmwareUpdateRequest()	1265
7.402.3.2 FileInit()	1265
7.402.3.3 FileIsIdle()	1266
7.402.3.4 FileSrecGetLineType()	1266
7.402.3.5 FileSrecParseLine()	1266
7.402.3.6 FileSrecVerifyChecksum()	1267
7.402.3.7 FileTask()	1267
7.403flash.c File Reference	1268
7.403.1 Detailed Description	1269
7.403.2 Function Documentation	1270
7.403.2.1 FlashAddToBlock()	1270
7.403.2.2 FlashDone()	1270

7.403.2.3 FlashErase()	1270
7.403.2.4 FlashEraseSectors()	1271
7.403.2.5 FlashGetSectorIdx()	1271
7.403.2.6 FlashGetUserProgBaseAddress()	1272
7.403.2.7 FlashInit()	1272
7.403.2.8 FlashInitBlock()	1272
7.403.2.9 FlashReinit()	1273
7.403.2.10FlashSwitchBlock()	1273
7.403.2.11FlashVerifyChecksum()	1273
7.403.2.12FlashWrite()	1274
7.403.2.13FlashWriteBlock()	1274
7.403.2.14FlashWriteChecksum()	1275
7.403.3 Variable Documentation	1275
7.403.3.1 blockInfo	1275
7.403.3.2 bootBlockInfo	1275
7.403.3.3 flashLayout	1276
7.404flash.c File Reference	1276
7.404.1 Detailed Description	1278
7.404.2 Function Documentation	1278
7.404.2.1 FlashAddToBlock()	1278
7.404.2.2 FlashCommandSequence()	1279
7.404.2.3 FlashDone()	1280
7.404.2.4 FlashErase()	1280
7.404.2.5 FlashEraseSectors()	1280
7.404.2.6 FlashGetSectorIdx()	1281
7.404.2.7 FlashGetUserProgBaseAddress()	1281
7.404.2.8 FlashInit()	1281
7.404.2.9 FlashInitBlock()	1281
7.404.2.10FlashReinit()	1282
7.404.2.11FlashSwitchBlock()	1282

7.404.2.12FlashVerifyChecksum()	1283
7.404.2.13FlashWrite()	1283
7.404.2.14FlashWriteBlock()	1283
7.404.2.15FlashWriteChecksum()	1284
7.404.3 Variable Documentation	1284
7.404.3.1 blockInfo	1284
7.404.3.2 bootBlockInfo	1284
7.405flash.c File Reference	1285
7.405.1 Detailed Description	1287
7.405.2 Function Documentation	1287
7.405.2.1 FlashAddToBlock()	1287
7.405.2.2 FlashDone()	1287
7.405.2.3 FlashErase()	1287
7.405.2.4 FlashEraseSectors()	1288
7.405.2.5 FlashGetSector()	1288
7.405.2.6 FlashGetSectorBaseAddr()	1289
7.405.2.7 FlashGetSectorSize()	1289
7.405.2.8 FlashGetUserProgBaseAddress()	1289
7.405.2.9 FlashInit()	1290
7.405.2.10FlashInitBlock()	1290
7.405.2.11FlashReinit()	1290
7.405.2.12FlashSwitchBlock()	1290
7.405.2.13FlashVerifyChecksum()	1291
7.405.2.14FlashWrite()	1291
7.405.2.15FlashWriteBlock()	1292
7.405.2.16FlashWriteChecksum()	1292
7.405.3 Variable Documentation	1292
7.405.3.1 blockInfo	1292
7.405.3.2 bootBlockInfo	1293
7.405.3.3 flashLayout	1293

7.406flash.c File Reference	1293
7.406.1 Detailed Description	1295
7.406.2 Function Documentation	1295
7.406.2.1 FlashAddToBlock()	1295
7.406.2.2 FlashDone()	1296
7.406.2.3 FlashErase()	1296
7.406.2.4 FlashEraseSectors()	1297
7.406.2.5 FlashGetSector()	1297
7.406.2.6 FlashGetSectorBaseAddr()	1297
7.406.2.7 FlashGetSectorSize()	1298
7.406.2.8 FlashGetUserProgBaseAddress()	1298
7.406.2.9 FlashInit()	1298
7.406.2.10FlashInitBlock()	1299
7.406.2.11FlashReinit()	1299
7.406.2.12FlashSwitchBlock()	1299
7.406.2.13FlashVerifyChecksum()	1300
7.406.2.14FlashWrite()	1300
7.406.2.15FlashWriteBlock()	1300
7.406.2.16FlashWriteChecksum()	1301
7.406.3 Variable Documentation	1301
7.406.3.1 blockInfo	1301
7.406.3.2 bootBlockInfo	1302
7.406.3.3 flashLayout	1302
7.407flash.c File Reference	1302
7.407.1 Detailed Description	1304
7.407.2 Function Documentation	1304
7.407.2.1 FlashAddToBlock()	1304
7.407.2.2 FlashDone()	1305
7.407.2.3 FlashErase()	1305
7.407.2.4 FlashEraseSectors()	1306

7.407.2.5 FlashGetSector()	1306
7.407.2.6 FlashGetSectorBaseAddr()	1306
7.407.2.7 FlashGetUserProgBaseAddress()	1307
7.407.2.8 FlashInit()	1307
7.407.2.9 FlashInitBlock()	1307
7.407.2.10FlashReinit()	1308
7.407.2.11FlashSwitchBlock()	1308
7.407.2.12FlashVerifyChecksum()	1308
7.407.2.13FlashWrite()	1309
7.407.2.14FlashWriteBlock()	1309
7.407.2.15FlashWriteChecksum()	1309
7.407.3 Variable Documentation	1310
7.407.3.1 blockInfo	1310
7.407.3.2 bootBlockInfo	1310
7.407.3.3 flashLayout	1310
7.408flash.c File Reference	1311
7.408.1 Detailed Description	1313
7.408.2 Function Documentation	1313
7.408.2.1 FlashAddToBlock()	1313
7.408.2.2 FlashDone()	1313
7.408.2.3 FlashErase()	1313
7.408.2.4 FlashEraseSectors()	1314
7.408.2.5 FlashGetBank()	1314
7.408.2.6 FlashGetPage()	1315
7.408.2.7 FlashGetPageSize()	1315
7.408.2.8 FlashGetSectorIdx()	1315
7.408.2.9 FlashGetUserProgBaseAddress()	1316
7.408.2.10FlashInit()	1316
7.408.2.11FlashInitBlock()	1316
7.408.2.12FlashIsDualBankMode()	1317

7.408.2.13FlashReinit()	1317
7.408.2.14FlashSwitchBlock()	1317
7.408.2.15FlashVerifyChecksum()	1318
7.408.2.16FlashWrite()	1318
7.408.2.17FlashWriteBlock()	1318
7.408.2.18FlashWriteChecksum()	1319
7.408.3 Variable Documentation	1319
7.408.3.1 blockInfo	1319
7.408.3.2 bootBlockInfo	1320
7.408.3.3 flashLayout	1320
7.409flash.c File Reference	1320
7.409.1 Detailed Description	1322
7.409.2 Function Documentation	1322
7.409.2.1 FlashAddToBlock()	1322
7.409.2.2 FlashCalcPageSize()	1323
7.409.2.3 FlashDone()	1323
7.409.2.4 FlashErase()	1323
7.409.2.5 FlashEraseSectors()	1324
7.409.2.6 FlashGetSector()	1324
7.409.2.7 FlashGetSectorBaseAddr()	1325
7.409.2.8 FlashGetSectorSize()	1325
7.409.2.9 FlashGetUserProgBaseAddress()	1325
7.409.2.10FlashInit()	1326
7.409.2.11FlashInitBlock()	1326
7.409.2.12FlashReinit()	1326
7.409.2.13FlashSwitchBlock()	1326
7.409.2.14FlashVerifyChecksum()	1327
7.409.2.15FlashWrite()	1327
7.409.2.16FlashWriteBlock()	1328
7.409.2.17FlashWriteChecksum()	1328

7.409.3 Variable Documentation	1328
7.409.3.1 blockInfo	1328
7.409.3.2 bootBlockInfo	1329
7.409.3.3 flashLayout	1329
7.410flash.c File Reference	1329
7.410.1 Detailed Description	1331
7.410.2 Function Documentation	1331
7.410.2.1 FlashAddToBlock()	1331
7.410.2.2 FlashDone()	1332
7.410.2.3 FlashErase()	1332
7.410.2.4 FlashEraseSectors()	1333
7.410.2.5 FlashGetSector()	1333
7.410.2.6 FlashGetSectorBaseAddr()	1333
7.410.2.7 FlashGetSectorSize()	1334
7.410.2.8 FlashGetUserProgBaseAddress()	1334
7.410.2.9 FlashInit()	1334
7.410.2.10FlashInitBlock()	1335
7.410.2.11FlashReinit()	1335
7.410.2.12FlashSwitchBlock()	1335
7.410.2.13FlashVerifyChecksum()	1336
7.410.2.14FlashWrite()	1336
7.410.2.15FlashWriteBlock()	1336
7.410.2.16FlashWriteChecksum()	1337
7.410.3 Variable Documentation	1337
7.410.3.1 blockInfo	1337
7.410.3.2 bootBlockInfo	1338
7.410.3.3 flashLayout	1338
7.411flash.c File Reference	1338
7.411.1 Detailed Description	1340
7.411.2 Function Documentation	1340

7.411.2.1 FlashAddToBlock()	1340
7.411.2.2 FlashDone()	1341
7.411.2.3 FlashErase()	1341
7.411.2.4 FlashGetUserProgBaseAddress()	1341
7.411.2.5 FlashInit()	1342
7.411.2.6 FlashInitBlock()	1342
7.411.2.7 FlashReinit()	1342
7.411.2.8 FlashSwitchBlock()	1342
7.411.2.9 FlashVerifyChecksum()	1343
7.411.2.10FlashWrite()	1343
7.411.2.11FlashWriteBlock()	1344
7.411.2.12FlashWriteChecksum()	1344
7.411.3 Variable Documentation	1344
7.411.3.1 blockInfo	1344
7.411.3.2 bootBlockInfo	1345
7.411.3.3 flashLayout	1345
7.412flash.c File Reference	1345
7.412.1 Detailed Description	1347
7.412.2 Function Documentation	1347
7.412.2.1 FlashAddToBlock()	1347
7.412.2.2 FlashDone()	1348
7.412.2.3 FlashErase()	1348
7.412.2.4 FlashEraseSectors()	1348
7.412.2.5 FlashGetSector()	1349
7.412.2.6 FlashGetUserProgBaseAddress()	1349
7.412.2.7 FlashInit()	1350
7.412.2.8 FlashInitBlock()	1350
7.412.2.9 FlashReinit()	1350
7.412.2.10FlashSwitchBlock()	1350
7.412.2.11FlashVerifyChecksum()	1351

7.412.2.12FlashWrite()	1351
7.412.2.13FlashWriteBlock()	1352
7.412.2.14FlashWriteChecksum()	1352
7.412.3 Variable Documentation	1352
7.412.3.1 blockInfo	1352
7.412.3.2 bootBlockInfo	1353
7.412.3.3 flashLayout	1353
7.413flash.c File Reference	1354
7.413.1 Detailed Description	1355
7.413.2 Function Documentation	1356
7.413.2.1 FlashAddToBlock()	1356
7.413.2.2 FlashCommandSequence()	1356
7.413.2.3 FlashDone()	1357
7.413.2.4 FlashErase()	1357
7.413.2.5 FlashEraseSectors()	1357
7.413.2.6 FlashGetSectorIdx()	1358
7.413.2.7 FlashGetUserProgBaseAddress()	1358
7.413.2.8 FlashInit()	1358
7.413.2.9 FlashInitBlock()	1358
7.413.2.10FlashReinit()	1359
7.413.2.11FlashSwitchBlock()	1359
7.413.2.12FlashVerifyChecksum()	1360
7.413.2.13FlashWrite()	1360
7.413.2.14FlashWriteBlock()	1360
7.413.2.15FlashWriteChecksum()	1361
7.413.3 Variable Documentation	1361
7.413.3.1 blockInfo	1361
7.413.3.2 bootBlockInfo	1361
7.414flash.c File Reference	1362
7.414.1 Detailed Description	1363

7.414.2 Function Documentation	1364
7.414.2.1 FlashAddToBlock()	1364
7.414.2.2 FlashDone()	1364
7.414.2.3 FlashErase()	1364
7.414.2.4 FlashGetUserProgBaseAddress()	1365
7.414.2.5 FlashInit()	1365
7.414.2.6 FlashInitBlock()	1365
7.414.2.7 FlashReinit()	1366
7.414.2.8 FlashSwitchBlock()	1366
7.414.2.9 FlashVerifyChecksum()	1367
7.414.2.10FlashWrite()	1367
7.414.2.11FlashWriteBlock()	1367
7.414.2.12FlashWriteChecksum()	1368
7.414.3 Variable Documentation	1368
7.414.3.1 blockInfo	1368
7.414.3.2 bootBlockInfo	1368
7.414.3.3 flashLayout	1369
7.415flash.c File Reference	1369
7.415.1 Detailed Description	1371
7.415.2 Function Documentation	1371
7.415.2.1 FlashAddToBlock()	1371
7.415.2.2 FlashDone()	1372
7.415.2.3 FlashErase()	1372
7.415.2.4 FlashEraseSectors()	1372
7.415.2.5 FlashGetSector()	1373
7.415.2.6 FlashGetUserProgBaseAddress()	1373
7.415.2.7 FlashInit()	1373
7.415.2.8 FlashInitBlock()	1373
7.415.2.9 FlashReinit()	1374
7.415.2.10FlashSwitchBlock()	1374

7.415.2.11FlashVerifyChecksum()	1375
7.415.2.12FlashWrite()	1375
7.415.2.13FlashWriteBlock()	1375
7.415.2.14FlashWriteChecksum()	1376
7.415.3 Variable Documentation	1376
7.415.3.1 blockInfo	1376
7.415.3.2 bootBlockInfo	1376
7.415.3.3 flashLayout	1377
7.416flash.c File Reference	1377
7.416.1 Detailed Description	1379
7.416.2 Function Documentation	1379
7.416.2.1 FlashAddToBlock()	1379
7.416.2.2 FlashDone()	1380
7.416.2.3 FlashErase()	1380
7.416.2.4 FlashGetBank()	1380
7.416.2.5 FlashGetPage()	1382
7.416.2.6 FlashGetUserProgBaseAddress()	1382
7.416.2.7 FlashInit()	1383
7.416.2.8 FlashInitBlock()	1383
7.416.2.9 FlashReinit()	1383
7.416.2.10FlashSwitchBlock()	1383
7.416.2.11FlashVerifyChecksum()	1384
7.416.2.12FlashWrite()	1384
7.416.2.13FlashWriteBlock()	1385
7.416.2.14FlashWriteChecksum()	1385
7.416.3 Variable Documentation	1385
7.416.3.1 blockInfo	1385
7.416.3.2 bootBlockInfo	1386
7.416.3.3 flashLayout	1386
7.417flash.c File Reference	1386

7.417.1 Detailed Description	1388
7.417.2 Function Documentation	1388
7.417.2.1 FlashAddToBlock()	1388
7.417.2.2 FlashDone()	1389
7.417.2.3 FlashErase()	1389
7.417.2.4 FlashEraseSectors()	1390
7.417.2.5 FlashGetSector()	1390
7.417.2.6 FlashGetSectorBaseAddr()	1390
7.417.2.7 FlashGetSectorSize()	1391
7.417.2.8 FlashGetUserProgBaseAddress()	1391
7.417.2.9 FlashInit()	1391
7.417.2.10FlashInitBlock()	1392
7.417.2.11FlashReinit()	1392
7.417.2.12FlashSwitchBlock()	1392
7.417.2.13FlashVerifyChecksum()	1393
7.417.2.14FlashWrite()	1393
7.417.2.15FlashWriteBlock()	1393
7.417.2.16FlashWriteChecksum()	1394
7.417.3 Variable Documentation	1394
7.417.3.1 blockInfo	1394
7.417.3.2 bootBlockInfo	1395
7.417.3.3 flashLayout	1395
7.418flash.c File Reference	1395
7.418.1 Detailed Description	1397
7.418.2 Function Documentation	1397
7.418.2.1 FlashAddToBlock()	1398
7.418.2.2 FlashDone()	1398
7.418.2.3 FlashErase()	1398
7.418.2.4 FlashEraseSectors()	1399
7.418.2.5 FlashGetSector()	1399

7.418.2.6 FlashGetSectorBaseAddr()	1400
7.418.2.7 FlashGetUserProgBaseAddress()	1400
7.418.2.8 FlashInit()	1400
7.418.2.9 FlashInitBlock()	1400
7.418.2.10 FlashReinit()	1401
7.418.2.11 FlashSwitchBlock()	1401
7.418.2.12 FlashTranslateToNonCachedAddress()	1402
7.418.2.13 FlashVerifyChecksum()	1402
7.418.2.14 FlashWrite()	1402
7.418.2.15 FlashWriteBlock()	1403
7.418.2.16 FlashWriteChecksum()	1403
7.418.3 Variable Documentation	1403
7.418.3.1 blockInfo	1404
7.418.3.2 bootBlockInfo	1404
7.418.3.3 flashLayout	1404
7.419 flash.c File Reference	1405
7.419.1 Detailed Description	1407
7.419.2 Function Documentation	1407
7.419.2.1 FlashAddToBlock()	1407
7.419.2.2 FlashDone()	1408
7.419.2.3 FlashErase()	1408
7.419.2.4 FlashEraseSectors()	1408
7.419.2.5 FlashGetSector()	1409
7.419.2.6 FlashGetUserProgBaseAddress()	1409
7.419.2.7 FlashInit()	1409
7.419.2.8 FlashInitBlock()	1409
7.419.2.9 FlashIsSingleBankMode()	1410
7.419.2.10 FlashReinit()	1410
7.419.2.11 FlashSwitchBlock()	1410
7.419.2.12 FlashVerifyChecksum()	1411

7.419.2.13FlashWrite()	1411
7.419.2.14FlashWriteBlock()	1412
7.419.2.15FlashWriteChecksum()	1412
7.419.3 Variable Documentation	1412
7.419.3.1 blockInfo	1412
7.419.3.2 bootBlockInfo	1413
7.419.3.3 flashLayout	1413
7.420flash.c File Reference	1414
7.420.1 Detailed Description	1415
7.420.2 Function Documentation	1416
7.420.2.1 FlashAddToBlock()	1416
7.420.2.2 FlashDone()	1416
7.420.2.3 FlashErase()	1416
7.420.2.4 FlashEraseSectors()	1417
7.420.2.5 FlashGetSectorIdx()	1417
7.420.2.6 FlashGetUserProgBaseAddress()	1418
7.420.2.7 FlashInit()	1418
7.420.2.8 FlashInitBlock()	1418
7.420.2.9 FlashReinit()	1419
7.420.2.10FlashSwitchBlock()	1419
7.420.2.11FlashVerifyChecksum()	1419
7.420.2.12FlashWrite()	1419
7.420.2.13FlashWriteBlock()	1420
7.420.2.14FlashWriteChecksum()	1420
7.420.3 Variable Documentation	1420
7.420.3.1 blockInfo	1421
7.420.3.2 bootBlockInfo	1421
7.420.3.3 flashLayout	1421
7.421flash.c File Reference	1422
7.421.1 Detailed Description	1425

7.421.2 Function Documentation	1425
7.421.2.1 FlashAddToBlock()	1425
7.421.2.2 FlashDone()	1425
7.421.2.3 FlashErase()	1425
7.421.2.4 FlashExecuteCommand()	1426
7.421.2.5 FlashGetLinearAddrByte()	1426
7.421.2.6 FlashGetPhysAddr()	1427
7.421.2.7 FlashGetPhysPage()	1427
7.421.2.8 FlashGetUserProgBaseAddress()	1427
7.421.2.9 FlashInit()	1428
7.421.2.10 FlashInitBlock()	1428
7.421.2.11 FlashOperate()	1428
7.421.2.12 FlashReinit()	1429
7.421.2.13 FlashSwitchBlock()	1429
7.421.2.14 FlashVerifyChecksum()	1429
7.421.2.15 FlashWrite()	1430
7.421.2.16 FlashWriteBlock()	1430
7.421.2.17 FlashWriteChecksum()	1430
7.421.3 Variable Documentation	1431
7.421.3.1 blockInfo	1431
7.421.3.2 bootBlockInfo	1431
7.421.3.3 flashExecCmd	1431
7.421.3.4 flashLayout	1432
7.422 flash.h File Reference	1432
7.422.1 Detailed Description	1433
7.422.2 Function Documentation	1433
7.422.2.1 FlashDone()	1433
7.422.2.2 FlashErase()	1433
7.422.2.3 FlashGetUserProgBaseAddress()	1434
7.422.2.4 FlashInit()	1434

7.422.2.5 FlashReinit()	1435
7.422.2.6 FlashVerifyChecksum()	1435
7.422.2.7 FlashWrite()	1435
7.422.2.8 FlashWriteChecksum()	1436
7.423flash.h File Reference	1436
7.423.1 Detailed Description	1437
7.423.2 Function Documentation	1437
7.423.2.1 FlashDone()	1437
7.423.2.2 FlashErase()	1437
7.423.2.3 FlashGetUserProgBaseAddress()	1438
7.423.2.4 FlashInit()	1438
7.423.2.5 FlashReinit()	1438
7.423.2.6 FlashVerifyChecksum()	1439
7.423.2.7 FlashWrite()	1439
7.423.2.8 FlashWriteChecksum()	1439
7.424flash.h File Reference	1440
7.424.1 Detailed Description	1440
7.424.2 Function Documentation	1440
7.424.2.1 FlashDone()	1441
7.424.2.2 FlashErase()	1441
7.424.2.3 FlashGetUserProgBaseAddress()	1441
7.424.2.4 FlashInit()	1442
7.424.2.5 FlashReinit()	1442
7.424.2.6 FlashVerifyChecksum()	1442
7.424.2.7 FlashWrite()	1442
7.424.2.8 FlashWriteChecksum()	1443
7.425flash.h File Reference	1443
7.425.1 Detailed Description	1444
7.425.2 Function Documentation	1444
7.425.2.1 FlashDone()	1444

7.425.2.2 FlashErase()	1444
7.425.2.3 FlashGetUserProgBaseAddress()	1445
7.425.2.4 FlashInit()	1445
7.425.2.5 FlashReinit()	1446
7.425.2.6 FlashVerifyChecksum()	1446
7.425.2.7 FlashWrite()	1446
7.425.2.8 FlashWriteChecksum()	1447
7.426flash.h File Reference	1447
7.426.1 Detailed Description	1448
7.426.2 Function Documentation	1448
7.426.2.1 FlashDone()	1448
7.426.2.2 FlashErase()	1448
7.426.2.3 FlashGetUserProgBaseAddress()	1449
7.426.2.4 FlashInit()	1449
7.426.2.5 FlashReinit()	1449
7.426.2.6 FlashVerifyChecksum()	1450
7.426.2.7 FlashWrite()	1450
7.426.2.8 FlashWriteChecksum()	1450
7.427flash.h File Reference	1451
7.427.1 Detailed Description	1451
7.427.2 Function Documentation	1451
7.427.2.1 FlashDone()	1452
7.427.2.2 FlashErase()	1452
7.427.2.3 FlashGetUserProgBaseAddress()	1452
7.427.2.4 FlashInit()	1453
7.427.2.5 FlashReinit()	1453
7.427.2.6 FlashVerifyChecksum()	1453
7.427.2.7 FlashWrite()	1453
7.427.2.8 FlashWriteChecksum()	1454
7.428flash.h File Reference	1454

7.428.1 Detailed Description	1455
7.428.2 Function Documentation	1455
7.428.2.1 FlashDone()	1455
7.428.2.2 FlashErase()	1455
7.428.2.3 FlashGetUserProgBaseAddress()	1456
7.428.2.4 FlashInit()	1456
7.428.2.5 FlashReinit()	1457
7.428.2.6 FlashVerifyChecksum()	1457
7.428.2.7 FlashWrite()	1457
7.428.2.8 FlashWriteChecksum()	1458
7.429flash.h File Reference	1458
7.429.1 Detailed Description	1459
7.429.2 Function Documentation	1459
7.429.2.1 FlashDone()	1459
7.429.2.2 FlashErase()	1459
7.429.2.3 FlashGetUserProgBaseAddress()	1460
7.429.2.4 FlashInit()	1460
7.429.2.5 FlashReinit()	1460
7.429.2.6 FlashVerifyChecksum()	1461
7.429.2.7 FlashWrite()	1461
7.429.2.8 FlashWriteChecksum()	1461
7.430flash.h File Reference	1462
7.430.1 Detailed Description	1462
7.430.2 Function Documentation	1462
7.430.2.1 FlashDone()	1463
7.430.2.2 FlashErase()	1463
7.430.2.3 FlashGetUserProgBaseAddress()	1463
7.430.2.4 FlashInit()	1464
7.430.2.5 FlashReinit()	1464
7.430.2.6 FlashVerifyChecksum()	1464

7.430.2.7 FlashWrite()	1464
7.430.2.8 FlashWriteChecksum()	1465
7.431flash.h File Reference	1465
7.431.1 Detailed Description	1466
7.431.2 Function Documentation	1466
7.431.2.1 FlashDone()	1466
7.431.2.2 FlashErase()	1466
7.431.2.3 FlashGetUserProgBaseAddress()	1467
7.431.2.4 FlashInit()	1467
7.431.2.5 FlashReinit()	1468
7.431.2.6 FlashVerifyChecksum()	1468
7.431.2.7 FlashWrite()	1468
7.431.2.8 FlashWriteChecksum()	1469
7.432flash.h File Reference	1469
7.432.1 Detailed Description	1470
7.432.2 Function Documentation	1470
7.432.2.1 FlashDone()	1470
7.432.2.2 FlashErase()	1470
7.432.2.3 FlashGetUserProgBaseAddress()	1471
7.432.2.4 FlashInit()	1471
7.432.2.5 FlashReinit()	1471
7.432.2.6 FlashVerifyChecksum()	1472
7.432.2.7 FlashWrite()	1472
7.432.2.8 FlashWriteChecksum()	1472
7.433flash.h File Reference	1473
7.433.1 Detailed Description	1473
7.433.2 Function Documentation	1473
7.433.2.1 FlashDone()	1474
7.433.2.2 FlashErase()	1474
7.433.2.3 FlashGetUserProgBaseAddress()	1474

7.433.2.4 FlashInit()	1475
7.433.2.5 FlashReinit()	1475
7.433.2.6 FlashVerifyChecksum()	1475
7.433.2.7 FlashWrite()	1475
7.433.2.8 FlashWriteChecksum()	1476
7.434flash.h File Reference	1476
7.434.1 Detailed Description	1477
7.434.2 Function Documentation	1477
7.434.2.1 FlashDone()	1477
7.434.2.2 FlashErase()	1477
7.434.2.3 FlashGetUserProgBaseAddress()	1478
7.434.2.4 FlashInit()	1478
7.434.2.5 FlashReinit()	1479
7.434.2.6 FlashVerifyChecksum()	1479
7.434.2.7 FlashWrite()	1479
7.434.2.8 FlashWriteChecksum()	1480
7.435flash.h File Reference	1480
7.435.1 Detailed Description	1481
7.435.2 Function Documentation	1481
7.435.2.1 FlashDone()	1481
7.435.2.2 FlashErase()	1481
7.435.2.3 FlashGetUserProgBaseAddress()	1482
7.435.2.4 FlashInit()	1482
7.435.2.5 FlashReinit()	1482
7.435.2.6 FlashVerifyChecksum()	1483
7.435.2.7 FlashWrite()	1483
7.435.2.8 FlashWriteChecksum()	1483
7.436flash.h File Reference	1484
7.436.1 Detailed Description	1484
7.436.2 Function Documentation	1484

7.436.2.1 FlashDone()	1485
7.436.2.2 FlashErase()	1485
7.436.2.3 FlashGetUserProgBaseAddress()	1485
7.436.2.4 FlashInit()	1486
7.436.2.5 FlashReinit()	1486
7.436.2.6 FlashVerifyChecksum()	1486
7.436.2.7 FlashWrite()	1486
7.436.2.8 FlashWriteChecksum()	1487
7.437flash.h File Reference	1487
7.437.1 Detailed Description	1488
7.437.2 Function Documentation	1488
7.437.2.1 FlashDone()	1488
7.437.2.2 FlashErase()	1488
7.437.2.3 FlashGetUserProgBaseAddress()	1489
7.437.2.4 FlashInit()	1489
7.437.2.5 FlashReinit()	1490
7.437.2.6 FlashVerifyChecksum()	1490
7.437.2.7 FlashWrite()	1490
7.437.2.8 FlashWriteChecksum()	1491
7.438flash.h File Reference	1491
7.438.1 Detailed Description	1492
7.438.2 Function Documentation	1492
7.438.2.1 FlashDone()	1492
7.438.2.2 FlashErase()	1492
7.438.2.3 FlashGetUserProgBaseAddress()	1493
7.438.2.4 FlashInit()	1493
7.438.2.5 FlashReinit()	1493
7.438.2.6 FlashVerifyChecksum()	1494
7.438.2.7 FlashWrite()	1494
7.438.2.8 FlashWriteChecksum()	1494

7.439flash.h File Reference	1495
7.439.1 Detailed Description	1495
7.439.2 Function Documentation	1495
7.439.2.1 FlashDone()	1496
7.439.2.2 FlashErase()	1496
7.439.2.3 FlashGetUserProgBaseAddress()	1496
7.439.2.4 FlashInit()	1497
7.439.2.5 FlashReinit()	1497
7.439.2.6 FlashVerifyChecksum()	1497
7.439.2.7 FlashWrite()	1497
7.439.2.8 FlashWriteChecksum()	1498
7.440flash.h File Reference	1498
7.440.1 Detailed Description	1499
7.440.2 Function Documentation	1499
7.440.2.1 FlashDone()	1499
7.440.2.2 FlashErase()	1499
7.440.2.3 FlashGetUserProgBaseAddress()	1500
7.440.2.4 FlashInit()	1501
7.440.2.5 FlashReinit()	1501
7.440.2.6 FlashVerifyChecksum()	1501
7.440.2.7 FlashWrite()	1501
7.440.2.8 FlashWriteChecksum()	1502
7.441flash_ecc.c File Reference	1502
7.441.1 Detailed Description	1505
7.441.2 Function Documentation	1505
7.441.2.1 FlashAddToBlock()	1505
7.441.2.2 FlashDone()	1506
7.441.2.3 FlashErase()	1506
7.441.2.4 FlashExecuteCommand()	1507
7.441.2.5 FlashGetGlobalAddrByte()	1507

7.441.2.6 FlashGetPhysAddr()	1507
7.441.2.7 FlashGetPhysPage()	1508
7.441.2.8 FlashGetUserProgBaseAddress()	1508
7.441.2.9 FlashInit()	1508
7.441.2.10FlashInitBlock()	1509
7.441.2.11FlashOperate()	1509
7.441.2.12FlashReinit()	1509
7.441.2.13FlashSwitchBlock()	1510
7.441.2.14FlashVerifyChecksum()	1510
7.441.2.15FlashWrite()	1510
7.441.2.16FlashWriteBlock()	1511
7.441.2.17FlashWriteChecksum()	1511
7.441.3 Variable Documentation	1511
7.441.3.1 blockInfo	1512
7.441.3.2 bootBlockInfo	1512
7.441.3.3 flashExecCmd	1512
7.441.3.4 flashLayout	1513
7.442flash_layout.c File Reference	1513
7.442.1 Detailed Description	1513
7.442.2 Variable Documentation	1513
7.442.2.1 flashLayout	1514
7.443flash_layout.c File Reference	1514
7.443.1 Detailed Description	1514
7.443.2 Variable Documentation	1514
7.443.2.1 flashLayout	1515
7.444flash_layout.c File Reference	1515
7.444.1 Detailed Description	1515
7.444.2 Variable Documentation	1515
7.444.2.1 flashLayout	1516
7.445flash_layout.c File Reference	1516

7.445.1 Detailed Description	1516
7.445.2 Variable Documentation	1516
7.445.2.1 flashLayout	1517
7.446flash_layout.c File Reference	1517
7.446.1 Detailed Description	1517
7.446.2 Variable Documentation	1517
7.446.2.1 flashLayout	1518
7.447flash_layout.c File Reference	1518
7.447.1 Detailed Description	1518
7.447.2 Variable Documentation	1518
7.447.2.1 flashLayout	1519
7.448flash_layout.c File Reference	1519
7.448.1 Detailed Description	1519
7.448.2 Variable Documentation	1519
7.448.2.1 flashLayout	1520
7.449flash_layout.c File Reference	1520
7.449.1 Detailed Description	1520
7.449.2 Variable Documentation	1520
7.449.2.1 flashLayout	1521
7.450flash_layout.c File Reference	1521
7.450.1 Detailed Description	1521
7.450.2 Variable Documentation	1521
7.450.2.1 flashLayout	1522
7.451flash_layout.c File Reference	1522
7.451.1 Detailed Description	1522
7.451.2 Variable Documentation	1522
7.451.2.1 flashLayout	1523
7.452flash_layout.c File Reference	1523
7.452.1 Detailed Description	1523
7.452.2 Variable Documentation	1523

7.452.2.1 flashLayout	1524
7.453flash_layout.c File Reference	1524
7.453.1 Detailed Description	1524
7.453.2 Variable Documentation	1524
7.453.2.1 flashLayout	1525
7.454flash_layout.c File Reference	1525
7.454.1 Detailed Description	1525
7.454.2 Variable Documentation	1525
7.454.2.1 flashLayout	1526
7.455flash_layout.c File Reference	1526
7.455.1 Detailed Description	1526
7.455.2 Variable Documentation	1526
7.455.2.1 flashLayout	1527
7.456flash_layout.c File Reference	1527
7.456.1 Detailed Description	1527
7.456.2 Variable Documentation	1527
7.456.2.1 flashLayout	1528
7.457flash_layout.c File Reference	1528
7.457.1 Detailed Description	1528
7.457.2 Variable Documentation	1528
7.457.2.1 flashLayout	1529
7.458flash_layout.c File Reference	1529
7.458.1 Detailed Description	1529
7.458.2 Variable Documentation	1529
7.458.2.1 flashLayout	1530
7.459flash_layout.c File Reference	1530
7.459.1 Detailed Description	1530
7.459.2 Variable Documentation	1530
7.459.2.1 flashLayout	1531
7.460flash_layout.c File Reference	1531

7.460.1 Detailed Description	1531
7.460.2 Variable Documentation	1531
7.460.2.1 flashLayout	1532
7.461 flash_layout.c File Reference	1532
7.461.1 Detailed Description	1532
7.461.2 Variable Documentation	1532
7.461.2.1 flashLayout	1533
7.462 flash_layout.c File Reference	1533
7.462.1 Detailed Description	1533
7.462.2 Variable Documentation	1533
7.462.2.1 flashLayout	1534
7.463 flash_layout.c File Reference	1534
7.463.1 Detailed Description	1534
7.463.2 Variable Documentation	1534
7.463.2.1 flashLayout	1535
7.464 header.h File Reference	1535
7.464.1 Detailed Description	1536
7.465 header.h File Reference	1536
7.465.1 Detailed Description	1537
7.466 header.h File Reference	1537
7.466.1 Detailed Description	1538
7.467 header.h File Reference	1538
7.467.1 Detailed Description	1539
7.468 header.h File Reference	1539
7.468.1 Detailed Description	1540
7.469 header.h File Reference	1540
7.469.1 Detailed Description	1541
7.470 header.h File Reference	1541
7.470.1 Detailed Description	1542
7.471 header.h File Reference	1542

7.471.1 Detailed Description	1543
7.472header.h File Reference	1543
7.472.1 Detailed Description	1544
7.473header.h File Reference	1544
7.473.1 Detailed Description	1545
7.474header.h File Reference	1545
7.474.1 Detailed Description	1546
7.475header.h File Reference	1546
7.475.1 Detailed Description	1547
7.476header.h File Reference	1547
7.476.1 Detailed Description	1548
7.477header.h File Reference	1548
7.477.1 Detailed Description	1549
7.478header.h File Reference	1549
7.478.1 Detailed Description	1550
7.479header.h File Reference	1550
7.479.1 Detailed Description	1551
7.480header.h File Reference	1551
7.480.1 Detailed Description	1552
7.481header.h File Reference	1552
7.481.1 Detailed Description	1553
7.482header.h File Reference	1553
7.482.1 Detailed Description	1554
7.483header.h File Reference	1554
7.483.1 Detailed Description	1555
7.484header.h File Reference	1555
7.484.1 Detailed Description	1556
7.485header.h File Reference	1556
7.485.1 Detailed Description	1557
7.486header.h File Reference	1557

7.486.1 Detailed Description	1558
7.487header.h File Reference	1558
7.487.1 Detailed Description	1559
7.488header.h File Reference	1559
7.488.1 Detailed Description	1560
7.489header.h File Reference	1560
7.489.1 Detailed Description	1561
7.490header.h File Reference	1561
7.490.1 Detailed Description	1562
7.491header.h File Reference	1562
7.491.1 Detailed Description	1563
7.492header.h File Reference	1563
7.492.1 Detailed Description	1564
7.493header.h File Reference	1564
7.493.1 Detailed Description	1565
7.494header.h File Reference	1565
7.494.1 Detailed Description	1566
7.495header.h File Reference	1566
7.495.1 Detailed Description	1567
7.496header.h File Reference	1567
7.496.1 Detailed Description	1568
7.497header.h File Reference	1568
7.497.1 Detailed Description	1569
7.498header.h File Reference	1569
7.498.1 Detailed Description	1570
7.499header.h File Reference	1570
7.499.1 Detailed Description	1571
7.500header.h File Reference	1571
7.500.1 Detailed Description	1572
7.501header.h File Reference	1572

7.501.1 Detailed Description	1573
7.502header.h File Reference	1573
7.502.1 Detailed Description	1574
7.503header.h File Reference	1574
7.503.1 Detailed Description	1575
7.504header.h File Reference	1575
7.504.1 Detailed Description	1576
7.505header.h File Reference	1576
7.505.1 Detailed Description	1577
7.506header.h File Reference	1577
7.506.1 Detailed Description	1578
7.507header.h File Reference	1578
7.507.1 Detailed Description	1579
7.508header.h File Reference	1579
7.508.1 Detailed Description	1580
7.509header.h File Reference	1580
7.509.1 Detailed Description	1581
7.510header.h File Reference	1581
7.510.1 Detailed Description	1582
7.511header.h File Reference	1582
7.511.1 Detailed Description	1583
7.512header.h File Reference	1583
7.512.1 Detailed Description	1584
7.513header.h File Reference	1584
7.513.1 Detailed Description	1585
7.514header.h File Reference	1585
7.514.1 Detailed Description	1586
7.515header.h File Reference	1586
7.515.1 Detailed Description	1587
7.516header.h File Reference	1587

7.516.1 Detailed Description	1588
7.517header.h File Reference	1588
7.517.1 Detailed Description	1589
7.518header.h File Reference	1589
7.518.1 Detailed Description	1590
7.519header.h File Reference	1590
7.519.1 Detailed Description	1591
7.520header.h File Reference	1591
7.520.1 Detailed Description	1592
7.521header.h File Reference	1592
7.521.1 Detailed Description	1593
7.522header.h File Reference	1593
7.522.1 Detailed Description	1594
7.523header.h File Reference	1594
7.523.1 Detailed Description	1595
7.524header.h File Reference	1595
7.524.1 Detailed Description	1596
7.525header.h File Reference	1596
7.525.1 Detailed Description	1597
7.526header.h File Reference	1597
7.526.1 Detailed Description	1598
7.527header.h File Reference	1598
7.527.1 Detailed Description	1599
7.528header.h File Reference	1599
7.528.1 Detailed Description	1600
7.529header.h File Reference	1600
7.529.1 Detailed Description	1601
7.530header.h File Reference	1601
7.530.1 Detailed Description	1602
7.531header.h File Reference	1602

7.531.1 Detailed Description	1603
7.532header.h File Reference	1603
7.532.1 Detailed Description	1604
7.533header.h File Reference	1604
7.533.1 Detailed Description	1605
7.534header.h File Reference	1605
7.534.1 Detailed Description	1606
7.535header.h File Reference	1606
7.535.1 Detailed Description	1607
7.536header.h File Reference	1607
7.536.1 Detailed Description	1608
7.537header.h File Reference	1608
7.537.1 Detailed Description	1609
7.538header.h File Reference	1609
7.538.1 Detailed Description	1610
7.539header.h File Reference	1610
7.539.1 Detailed Description	1611
7.540header.h File Reference	1611
7.540.1 Detailed Description	1612
7.541header.h File Reference	1612
7.541.1 Detailed Description	1613
7.542header.h File Reference	1613
7.542.1 Detailed Description	1614
7.543header.h File Reference	1614
7.543.1 Detailed Description	1615
7.544header.h File Reference	1615
7.544.1 Detailed Description	1616
7.545header.h File Reference	1616
7.545.1 Detailed Description	1617
7.546header.h File Reference	1617

7.546.1 Detailed Description	1618
7.546.2 Macro Definition Documentation	1618
7.546.2.1 BDM_DEBUGGING_ENABLED	1618
7.547header.h File Reference	1619
7.547.1 Detailed Description	1619
7.547.2 Macro Definition Documentation	1619
7.547.2.1 BDM_DEBUGGING_ENABLED	1620
7.548hooks.c File Reference	1620
7.548.1 Detailed Description	1621
7.548.2 Function Documentation	1621
7.548.2.1 BackDoorEntryHook()	1621
7.548.2.2 BackDoorInitHook()	1621
7.548.2.3 CopInitHook()	1622
7.548.2.4 CopServiceHook()	1622
7.548.2.5 CpuUserProgramStartHook()	1622
7.548.2.6 NvmDoneHook()	1622
7.548.2.7 NvmEraseHook()	1623
7.548.2.8 NvmInitHook()	1623
7.548.2.9 NvmReinitHook()	1623
7.548.2.10NvmWriteHook()	1623
7.548.2.11XcpGetSeedHook()	1624
7.548.2.12XcpVerifyKeyHook()	1624
7.549hooks.c File Reference	1625
7.549.1 Detailed Description	1626
7.549.2 Function Documentation	1626
7.549.2.1 BackDoorEntryHook()	1626
7.549.2.2 BackDoorInitHook()	1626
7.549.2.3 CopInitHook()	1627
7.549.2.4 CopServiceHook()	1627
7.549.2.5 CpuUserProgramStartHook()	1627

7.549.2.6 NvmDoneHook()	1627
7.549.2.7 NvmEraseHook()	1628
7.549.2.8 NvmInitHook()	1628
7.549.2.9 NvmReinitHook()	1628
7.549.2.10 NvmWriteHook()	1628
7.549.2.11 XcpGetSeedHook()	1629
7.549.2.12 XcpVerifyKeyHook()	1629
7.550 hooks.c File Reference	1630
7.550.1 Detailed Description	1631
7.550.2 Function Documentation	1631
7.550.2.1 BackDoorEntryHook()	1631
7.550.2.2 BackDoorInitHook()	1631
7.550.2.3 CopInitHook()	1632
7.550.2.4 CopServiceHook()	1632
7.550.2.5 CpuUserProgramStartHook()	1632
7.550.2.6 NvmDoneHook()	1632
7.550.2.7 NvmEraseHook()	1633
7.550.2.8 NvmInitHook()	1633
7.550.2.9 NvmReinitHook()	1633
7.550.2.10 NvmWriteHook()	1633
7.550.2.11 XcpGetSeedHook()	1634
7.550.2.12 XcpVerifyKeyHook()	1634
7.551 hooks.c File Reference	1635
7.551.1 Detailed Description	1636
7.551.2 Function Documentation	1636
7.551.2.1 BackDoorEntryHook()	1636
7.551.2.2 BackDoorInitHook()	1636
7.551.2.3 CopInitHook()	1637
7.551.2.4 CopServiceHook()	1637
7.551.2.5 CpuUserProgramStartHook()	1637

7.551.2.6 NvmDoneHook()	1637
7.551.2.7 NvmEraseHook()	1638
7.551.2.8 NvmInitHook()	1638
7.551.2.9 NvmReinitHook()	1638
7.551.2.10 NvmWriteHook()	1638
7.551.2.11 XcpGetSeedHook()	1639
7.551.2.12 XcpVerifyKeyHook()	1639
7.552hooks.c File Reference	1640
7.552.1 Detailed Description	1641
7.552.2 Function Documentation	1641
7.552.2.1 BackDoorEntryHook()	1641
7.552.2.2 BackDoorInitHook()	1641
7.552.2.3 CopInitHook()	1642
7.552.2.4 CopServiceHook()	1642
7.552.2.5 CpuUserProgramStartHook()	1642
7.552.2.6 NvmDoneHook()	1642
7.552.2.7 NvmEraseHook()	1643
7.552.2.8 NvmInitHook()	1643
7.552.2.9 NvmReinitHook()	1643
7.552.2.10 NvmWriteHook()	1643
7.552.2.11 XcpGetSeedHook()	1644
7.552.2.12 XcpVerifyKeyHook()	1644
7.553hooks.c File Reference	1645
7.553.1 Detailed Description	1646
7.553.2 Function Documentation	1646
7.553.2.1 BackDoorEntryHook()	1646
7.553.2.2 BackDoorInitHook()	1646
7.553.2.3 CopInitHook()	1647
7.553.2.4 CopServiceHook()	1647
7.553.2.5 CpuUserProgramStartHook()	1647

7.553.2.6 NvmDoneHook()	1647
7.553.2.7 NvmEraseHook()	1648
7.553.2.8 NvmInitHook()	1648
7.553.2.9 NvmReinitHook()	1648
7.553.2.10 NvmWriteHook()	1648
7.553.2.11 XcpGetSeedHook()	1649
7.553.2.12 XcpVerifyKeyHook()	1649
7.554hooks.c File Reference	1650
7.554.1 Detailed Description	1651
7.554.2 Function Documentation	1651
7.554.2.1 BackDoorEntryHook()	1651
7.554.2.2 BackDoorInitHook()	1651
7.554.2.3 CopInitHook()	1652
7.554.2.4 CopServiceHook()	1652
7.554.2.5 CpuUserProgramStartHook()	1652
7.554.2.6 NvmDoneHook()	1652
7.554.2.7 NvmEraseHook()	1653
7.554.2.8 NvmInitHook()	1653
7.554.2.9 NvmReinitHook()	1653
7.554.2.10 NvmWriteHook()	1653
7.554.2.11 XcpGetSeedHook()	1654
7.554.2.12 XcpVerifyKeyHook()	1654
7.555hooks.c File Reference	1655
7.555.1 Detailed Description	1656
7.555.2 Function Documentation	1656
7.555.2.1 BackDoorEntryHook()	1656
7.555.2.2 BackDoorInitHook()	1656
7.555.2.3 CopInitHook()	1657
7.555.2.4 CopServiceHook()	1657
7.555.2.5 CpuUserProgramStartHook()	1657

7.555.2.6 NvmDoneHook()	1657
7.555.2.7 NvmEraseHook()	1658
7.555.2.8 NvmInitHook()	1658
7.555.2.9 NvmReinitHook()	1658
7.555.2.10 NvmWriteHook()	1658
7.555.2.11 XcpGetSeedHook()	1659
7.555.2.12 XcpVerifyKeyHook()	1659
7.556hooks.c File Reference	1660
7.556.1 Detailed Description	1661
7.556.2 Function Documentation	1661
7.556.2.1 BackDoorEntryHook()	1661
7.556.2.2 BackDoorInitHook()	1661
7.556.2.3 CopInitHook()	1662
7.556.2.4 CopServiceHook()	1662
7.556.2.5 CpuUserProgramStartHook()	1662
7.556.2.6 NvmDoneHook()	1662
7.556.2.7 NvmEraseHook()	1663
7.556.2.8 NvmInitHook()	1663
7.556.2.9 NvmReinitHook()	1663
7.556.2.10 NvmWriteHook()	1663
7.556.2.11 XcpGetSeedHook()	1664
7.556.2.12 XcpVerifyKeyHook()	1664
7.557hooks.c File Reference	1665
7.557.1 Detailed Description	1666
7.557.2 Function Documentation	1666
7.557.2.1 BackDoorEntryHook()	1666
7.557.2.2 BackDoorInitHook()	1666
7.557.2.3 CopInitHook()	1667
7.557.2.4 CopServiceHook()	1667
7.557.2.5 CpuUserProgramStartHook()	1667

7.557.2.6 NvmDoneHook()	1667
7.557.2.7 NvmEraseHook()	1668
7.557.2.8 NvmInitHook()	1668
7.557.2.9 NvmReinitHook()	1668
7.557.2.10 NvmWriteHook()	1668
7.557.2.11 XcpGetSeedHook()	1669
7.557.2.12 XcpVerifyKeyHook()	1669
7.558hooks.c File Reference	1670
7.558.1 Detailed Description	1671
7.558.2 Function Documentation	1671
7.558.2.1 BackDoorEntryHook()	1671
7.558.2.2 BackDoorInitHook()	1671
7.558.2.3 CopInitHook()	1672
7.558.2.4 CopServiceHook()	1672
7.558.2.5 CpuUserProgramStartHook()	1672
7.558.2.6 NvmDoneHook()	1672
7.558.2.7 NvmEraseHook()	1673
7.558.2.8 NvmInitHook()	1673
7.558.2.9 NvmReinitHook()	1673
7.558.2.10 NvmWriteHook()	1673
7.558.2.11 XcpGetSeedHook()	1674
7.558.2.12 XcpVerifyKeyHook()	1674
7.559hooks.c File Reference	1675
7.559.1 Detailed Description	1676
7.559.2 Function Documentation	1676
7.559.2.1 BackDoorEntryHook()	1676
7.559.2.2 BackDoorInitHook()	1676
7.559.2.3 CopInitHook()	1677
7.559.2.4 CopServiceHook()	1677
7.559.2.5 CpuUserProgramStartHook()	1677

7.559.2.6 NvmDoneHook()	1677
7.559.2.7 NvmEraseHook()	1678
7.559.2.8 NvmInitHook()	1678
7.559.2.9 NvmReinitHook()	1678
7.559.2.10 NvmWriteHook()	1678
7.559.2.11 XcpGetSeedHook()	1679
7.559.2.12 XcpVerifyKeyHook()	1679
7.560 hooks.c File Reference	1680
7.560.1 Detailed Description	1681
7.560.2 Function Documentation	1681
7.560.2.1 BackDoorEntryHook()	1681
7.560.2.2 BackDoorInitHook()	1681
7.560.2.3 CopInitHook()	1682
7.560.2.4 CopServiceHook()	1682
7.560.2.5 CpuUserProgramStartHook()	1682
7.560.2.6 NvmDoneHook()	1682
7.560.2.7 NvmEraseHook()	1683
7.560.2.8 NvmInitHook()	1683
7.560.2.9 NvmReinitHook()	1683
7.560.2.10 NvmWriteHook()	1683
7.560.2.11 XcpGetSeedHook()	1684
7.560.2.12 XcpVerifyKeyHook()	1684
7.561 hooks.c File Reference	1685
7.561.1 Detailed Description	1686
7.561.2 Function Documentation	1686
7.561.2.1 BackDoorEntryHook()	1686
7.561.2.2 BackDoorInitHook()	1686
7.561.2.3 CopInitHook()	1687
7.561.2.4 CopServiceHook()	1687
7.561.2.5 CpuUserProgramStartHook()	1687

7.561.2.6 NvmDoneHook()	1687
7.561.2.7 NvmEraseHook()	1688
7.561.2.8 NvmInitHook()	1688
7.561.2.9 NvmReinitHook()	1688
7.561.2.10 NvmWriteHook()	1688
7.561.2.11 XcpGetSeedHook()	1689
7.561.2.12 XcpVerifyKeyHook()	1689
7.562 hooks.c File Reference	1690
7.562.1 Detailed Description	1691
7.562.2 Function Documentation	1691
7.562.2.1 BackDoorEntryHook()	1691
7.562.2.2 BackDoorInitHook()	1691
7.562.2.3 CopInitHook()	1692
7.562.2.4 CopServiceHook()	1692
7.562.2.5 CpuUserProgramStartHook()	1692
7.562.2.6 NvmDoneHook()	1692
7.562.2.7 NvmEraseHook()	1693
7.562.2.8 NvmInitHook()	1693
7.562.2.9 NvmReinitHook()	1693
7.562.2.10 NvmWriteHook()	1693
7.562.2.11 XcpGetSeedHook()	1694
7.562.2.12 XcpVerifyKeyHook()	1694
7.563 hooks.c File Reference	1695
7.563.1 Detailed Description	1696
7.563.2 Function Documentation	1696
7.563.2.1 BackDoorEntryHook()	1696
7.563.2.2 BackDoorInitHook()	1696
7.563.2.3 CopInitHook()	1697
7.563.2.4 CopServiceHook()	1697
7.563.2.5 CpuUserProgramStartHook()	1697

7.563.2.6 NvmDoneHook()	1697
7.563.2.7 NvmEraseHook()	1698
7.563.2.8 NvmInitHook()	1698
7.563.2.9 NvmReinitHook()	1698
7.563.2.10NvmWriteHook()	1698
7.563.2.11UsbConnectHook()	1699
7.563.2.12UsbEnterLowPowerModeHook()	1699
7.563.2.13UsbLeaveLowPowerModeHook()	1700
7.563.2.14XcpGetSeedHook()	1700
7.563.2.15XcpVerifyKeyHook()	1700
7.564hooks.c File Reference	1701
7.564.1 Detailed Description	1702
7.564.2 Function Documentation	1702
7.564.2.1 BackDoorEntryHook()	1702
7.564.2.2 BackDoorInitHook()	1702
7.564.2.3 CopInitHook()	1703
7.564.2.4 CopServiceHook()	1703
7.564.2.5 CpuUserProgramStartHook()	1703
7.564.2.6 NvmDoneHook()	1703
7.564.2.7 NvmEraseHook()	1704
7.564.2.8 NvmInitHook()	1704
7.564.2.9 NvmReinitHook()	1704
7.564.2.10NvmWriteHook()	1704
7.564.2.11UsbConnectHook()	1705
7.564.2.12UsbEnterLowPowerModeHook()	1705
7.564.2.13UsbLeaveLowPowerModeHook()	1705
7.564.2.14XcpGetSeedHook()	1706
7.564.2.15XcpVerifyKeyHook()	1706
7.565hooks.c File Reference	1706
7.565.1 Detailed Description	1708

7.565.2 Function Documentation	1708
7.565.2.1 BackDoorEntryHook()	1708
7.565.2.2 BackDoorInitHook()	1708
7.565.2.3 CopInitHook()	1709
7.565.2.4 CopServiceHook()	1709
7.565.2.5 CpuUserProgramStartHook()	1709
7.565.2.6 NvmDoneHook()	1709
7.565.2.7 NvmEraseHook()	1710
7.565.2.8 NvmInitHook()	1710
7.565.2.9 NvmReinitHook()	1710
7.565.2.10NvmWriteHook()	1710
7.565.2.11UsbConnectHook()	1711
7.565.2.12UsbEnterLowPowerModeHook()	1711
7.565.2.13UsbLeaveLowPowerModeHook()	1711
7.565.2.14XcpGetSeedHook()	1712
7.565.2.15XcpVerifyKeyHook()	1712
7.566hooks.c File Reference	1712
7.566.1 Detailed Description	1714
7.566.2 Function Documentation	1714
7.566.2.1 BackDoorEntryHook()	1714
7.566.2.2 BackDoorInitHook()	1714
7.566.2.3 CopInitHook()	1715
7.566.2.4 CopServiceHook()	1715
7.566.2.5 CpuUserProgramStartHook()	1715
7.566.2.6 NvmDoneHook()	1715
7.566.2.7 NvmEraseHook()	1716
7.566.2.8 NvmInitHook()	1716
7.566.2.9 NvmReinitHook()	1716
7.566.2.10NvmWriteHook()	1716
7.566.2.11UsbConnectHook()	1717

7.566.2.12	JsbsbEnterLowPowerModeHook()	1717
7.566.2.13	JsbsbLeaveLowPowerModeHook()	1717
7.566.2.14	XcpGetSeedHook()	1718
7.566.2.15	XcpVerifyKeyHook()	1718
7.567	hooks.c File Reference	1718
7.567.1	Detailed Description	1720
7.567.2	Function Documentation	1720
7.567.2.1	BackDoorEntryHook()	1720
7.567.2.2	BackDoorInitHook()	1720
7.567.2.3	CopInitHook()	1720
7.567.2.4	CopServiceHook()	1721
7.567.2.5	CpuUserProgramStartHook()	1721
7.567.2.6	NvmDoneHook()	1721
7.567.2.7	NvmEraseHook()	1721
7.567.2.8	NvmInitHook()	1722
7.567.2.9	NvmReinitHook()	1722
7.567.2.10	NvmWriteHook()	1722
7.567.2.11	XcpGetSeedHook()	1723
7.567.2.12	XcpVerifyKeyHook()	1723
7.568	hooks.c File Reference	1724
7.568.1	Detailed Description	1725
7.568.2	Function Documentation	1725
7.568.2.1	BackDoorEntryHook()	1725
7.568.2.2	BackDoorInitHook()	1725
7.568.2.3	CopInitHook()	1726
7.568.2.4	CopServiceHook()	1726
7.568.2.5	CpuUserProgramStartHook()	1726
7.568.2.6	NvmDoneHook()	1726
7.568.2.7	NvmEraseHook()	1727
7.568.2.8	NvmInitHook()	1727

7.568.2.9 NvmReinitHook()	1727
7.568.2.10NvmWriteHook()	1727
7.568.2.11XcpGetSeedHook()	1728
7.568.2.12XcpVerifyKeyHook()	1728
7.569hooks.c File Reference	1729
7.569.1 Detailed Description	1730
7.569.2 Function Documentation	1730
7.569.2.1 BackDoorEntryHook()	1731
7.569.2.2 BackDoorInitHook()	1731
7.569.2.3 CopInitHook()	1731
7.569.2.4 CopServiceHook()	1731
7.569.2.5 CpuUserProgramStartHook()	1732
7.569.2.6 FileFirmwareUpdateCompletedHook()	1732
7.569.2.7 FileFirmwareUpdateErrorHook()	1732
7.569.2.8 FileFirmwareUpdateLogHook()	1732
7.569.2.9 FileFirmwareUpdateStartedHook()	1733
7.569.2.10FileGetFirmwareFilenameHook()	1733
7.569.2.11FileIsFirmwareUpdateRequestedHook()	1733
7.569.2.12NvmDoneHook()	1734
7.569.2.13NvmEraseHook()	1734
7.569.2.14NvmInitHook()	1734
7.569.2.15NvmReinitHook()	1735
7.569.2.16NvmWriteHook()	1735
7.569.2.17XcpGetSeedHook()	1735
7.569.2.18XcpVerifyKeyHook()	1736
7.569.3 Variable Documentation	1736
7.569.3.1 canUse	1736
7.569.3.2 handle	1736
7.570hooks.c File Reference	1737
7.570.1 Detailed Description	1738

7.570.2 Function Documentation	1738
7.570.2.1 BackDoorEntryHook()	1738
7.570.2.2 BackDoorInitHook()	1739
7.570.2.3 CopInitHook()	1739
7.570.2.4 CopServiceHook()	1739
7.570.2.5 CpuUserProgramStartHook()	1739
7.570.2.6 FileFirmwareUpdateCompletedHook()	1740
7.570.2.7 FileFirmwareUpdateErrorHook()	1740
7.570.2.8 FileFirmwareUpdateLogHook()	1740
7.570.2.9 FileFirmwareUpdateStartedHook()	1740
7.570.2.10 FileGetFirmwareFilenameHook()	1741
7.570.2.11 FileIsFirmwareUpdateRequestedHook()	1741
7.570.2.12 NvmDoneHook()	1741
7.570.2.13 NvmEraseHook()	1741
7.570.2.14 NvmInitHook()	1742
7.570.2.15 NvmReinitHook()	1742
7.570.2.16 NvmWriteHook()	1742
7.570.2.17 XcpGetSeedHook()	1743
7.570.2.18 XcpVerifyKeyHook()	1743
7.570.3 Variable Documentation	1744
7.570.3.1 canUse	1744
7.570.3.2 handle	1744
7.571 hooks.c File Reference	1744
7.571.1 Detailed Description	1745
7.571.2 Function Documentation	1745
7.571.2.1 BackDoorEntryHook()	1746
7.571.2.2 BackDoorInitHook()	1746
7.571.2.3 CopInitHook()	1746
7.571.2.4 CopServiceHook()	1746
7.571.2.5 CpuUserProgramStartHook()	1747

7.571.2.6 NvmDoneHook()	1747
7.571.2.7 NvmEraseHook()	1747
7.571.2.8 NvmInitHook()	1748
7.571.2.9 NvmReinitHook()	1748
7.571.2.10 NvmWriteHook()	1748
7.571.2.11 XcpGetSeedHook()	1749
7.571.2.12 XcpVerifyKeyHook()	1749
7.572 hooks.c File Reference	1749
7.572.1 Detailed Description	1751
7.572.2 Function Documentation	1751
7.572.2.1 BackDoorEntryHook()	1751
7.572.2.2 BackDoorInitHook()	1751
7.572.2.3 CopInitHook()	1751
7.572.2.4 CopServiceHook()	1752
7.572.2.5 CpuUserProgramStartHook()	1752
7.572.2.6 NvmDoneHook()	1752
7.572.2.7 NvmEraseHook()	1752
7.572.2.8 NvmInitHook()	1753
7.572.2.9 NvmReinitHook()	1753
7.572.2.10 NvmWriteHook()	1753
7.572.2.11 XcpGetSeedHook()	1754
7.572.2.12 XcpVerifyKeyHook()	1754
7.573 hooks.c File Reference	1755
7.573.1 Detailed Description	1756
7.573.2 Function Documentation	1756
7.573.2.1 BackDoorEntryHook()	1756
7.573.2.2 BackDoorInitHook()	1756
7.573.2.3 CopInitHook()	1757
7.573.2.4 CopServiceHook()	1757
7.573.2.5 CpuUserProgramStartHook()	1757

7.573.2.6 NvmDoneHook()	1757
7.573.2.7 NvmEraseHook()	1758
7.573.2.8 NvmInitHook()	1758
7.573.2.9 NvmReinitHook()	1758
7.573.2.10 NvmWriteHook()	1758
7.573.2.11 XcpGetSeedHook()	1759
7.573.2.12 XcpVerifyKeyHook()	1759
7.574 hooks.c File Reference	1760
7.574.1 Detailed Description	1761
7.574.2 Function Documentation	1761
7.574.2.1 BackDoorEntryHook()	1761
7.574.2.2 BackDoorInitHook()	1761
7.574.2.3 CopInitHook()	1762
7.574.2.4 CopServiceHook()	1762
7.574.2.5 CpuUserProgramStartHook()	1762
7.574.2.6 NvmDoneHook()	1762
7.574.2.7 NvmEraseHook()	1763
7.574.2.8 NvmInitHook()	1763
7.574.2.9 NvmReinitHook()	1763
7.574.2.10 NvmWriteHook()	1763
7.574.2.11 XcpGetSeedHook()	1764
7.574.2.12 XcpVerifyKeyHook()	1764
7.575 hooks.c File Reference	1765
7.575.1 Detailed Description	1766
7.575.2 Function Documentation	1766
7.575.2.1 BackDoorEntryHook()	1766
7.575.2.2 BackDoorInitHook()	1766
7.575.2.3 CopInitHook()	1767
7.575.2.4 CopServiceHook()	1767
7.575.2.5 CpuUserProgramStartHook()	1767

7.575.2.6 NvmDoneHook()	1767
7.575.2.7 NvmEraseHook()	1768
7.575.2.8 NvmInitHook()	1768
7.575.2.9 NvmReinitHook()	1768
7.575.2.10 NvmWriteHook()	1768
7.575.2.11 XcpGetSeedHook()	1769
7.575.2.12 XcpVerifyKeyHook()	1769
7.576 hooks.c File Reference	1770
7.576.1 Detailed Description	1771
7.576.2 Function Documentation	1771
7.576.2.1 BackDoorEntryHook()	1771
7.576.2.2 BackDoorInitHook()	1771
7.576.2.3 CopInitHook()	1772
7.576.2.4 CopServiceHook()	1772
7.576.2.5 CpuUserProgramStartHook()	1772
7.576.2.6 NvmDoneHook()	1772
7.576.2.7 NvmEraseHook()	1773
7.576.2.8 NvmInitHook()	1773
7.576.2.9 NvmReinitHook()	1773
7.576.2.10 NvmWriteHook()	1773
7.576.2.11 XcpGetSeedHook()	1774
7.576.2.12 XcpVerifyKeyHook()	1774
7.577 hooks.c File Reference	1775
7.577.1 Detailed Description	1776
7.577.2 Function Documentation	1776
7.577.2.1 BackDoorEntryHook()	1776
7.577.2.2 BackDoorInitHook()	1776
7.577.2.3 CopInitHook()	1777
7.577.2.4 CopServiceHook()	1777
7.577.2.5 CpuUserProgramStartHook()	1777

7.577.2.6 NvmDoneHook()	1777
7.577.2.7 NvmEraseHook()	1778
7.577.2.8 NvmInitHook()	1778
7.577.2.9 NvmReinitHook()	1778
7.577.2.10NvmWriteHook()	1778
7.577.2.11UsbConnectHook()	1779
7.577.2.12UsbEnterLowPowerModeHook()	1779
7.577.2.13UsbLeaveLowPowerModeHook()	1779
7.577.2.14XcpGetSeedHook()	1780
7.577.2.15XcpVerifyKeyHook()	1780
7.578hooks.c File Reference	1780
7.578.1 Detailed Description	1782
7.578.2 Function Documentation	1782
7.578.2.1 BackDoorEntryHook()	1782
7.578.2.2 BackDoorInitHook()	1782
7.578.2.3 CopInitHook()	1782
7.578.2.4 CopServiceHook()	1783
7.578.2.5 CpuUserProgramStartHook()	1783
7.578.2.6 NvmDoneHook()	1783
7.578.2.7 NvmEraseHook()	1783
7.578.2.8 NvmInitHook()	1784
7.578.2.9 NvmReinitHook()	1784
7.578.2.10NvmWriteHook()	1784
7.578.2.11UsbConnectHook()	1785
7.578.2.12UsbEnterLowPowerModeHook()	1785
7.578.2.13UsbLeaveLowPowerModeHook()	1785
7.578.2.14XcpGetSeedHook()	1786
7.578.2.15XcpVerifyKeyHook()	1786
7.579hooks.c File Reference	1786
7.579.1 Detailed Description	1788

7.579.2 Function Documentation	1788
7.579.2.1 BackDoorEntryHook()	1788
7.579.2.2 BackDoorInitHook()	1788
7.579.2.3 CopInitHook()	1788
7.579.2.4 CopServiceHook()	1789
7.579.2.5 CpuUserProgramStartHook()	1789
7.579.2.6 NvmDoneHook()	1789
7.579.2.7 NvmEraseHook()	1789
7.579.2.8 NvmInitHook()	1790
7.579.2.9 NvmReinitHook()	1790
7.579.2.10 NvmWriteHook()	1790
7.579.2.11 UsbConnectHook()	1791
7.579.2.12 UsbEnterLowPowerModeHook()	1791
7.579.2.13 UsbLeaveLowPowerModeHook()	1791
7.579.2.14 XcpGetSeedHook()	1792
7.579.2.15 XcpVerifyKeyHook()	1792
7.580 hooks.c File Reference	1792
7.580.1 Detailed Description	1794
7.580.2 Function Documentation	1794
7.580.2.1 BackDoorEntryHook()	1794
7.580.2.2 BackDoorInitHook()	1794
7.580.2.3 CopInitHook()	1794
7.580.2.4 CopServiceHook()	1795
7.580.2.5 CpuUserProgramStartHook()	1795
7.580.2.6 NvmDoneHook()	1795
7.580.2.7 NvmEraseHook()	1795
7.580.2.8 NvmInitHook()	1796
7.580.2.9 NvmReinitHook()	1796
7.580.2.10 NvmWriteHook()	1796
7.580.2.11 UsbConnectHook()	1797

7.580.2.12	UsbEnterLowPowerModeHook()	1797
7.580.2.13	UsbLeaveLowPowerModeHook()	1797
7.580.2.14	XcpGetSeedHook()	1798
7.580.2.15	XcpVerifyKeyHook()	1798
7.581	hooks.c File Reference	1798
7.581.1	Detailed Description	1800
7.581.2	Function Documentation	1800
7.581.2.1	BackDoorEntryHook()	1800
7.581.2.2	BackDoorInitHook()	1801
7.581.2.3	CopInitHook()	1801
7.581.2.4	CopServiceHook()	1801
7.581.2.5	CpuUserProgramStartHook()	1801
7.581.2.6	FileFirmwareUpdateCompletedHook()	1802
7.581.2.7	FileFirmwareUpdateErrorHook()	1802
7.581.2.8	FileFirmwareUpdateLogHook()	1802
7.581.2.9	FileFirmwareUpdateStartedHook()	1802
7.581.2.10	FileGetFirmwareFilenameHook()	1803
7.581.2.11	FileIsFirmwareUpdateRequestedHook()	1803
7.581.2.12	NvmDoneHook()	1803
7.581.2.13	NvmEraseHook()	1803
7.581.2.14	NvmInitHook()	1804
7.581.2.15	NvmReinitHook()	1804
7.581.2.16	NvmWriteHook()	1804
7.581.2.17	XcpGetSeedHook()	1805
7.581.2.18	XcpVerifyKeyHook()	1805
7.581.3	Variable Documentation	1806
7.581.3.1	canUse	1806
7.581.3.2	handle	1806
7.582	hooks.c File Reference	1806
7.582.1	Detailed Description	1808

7.582.2 Function Documentation	1808
7.582.2.1 BackDoorEntryHook()	1808
7.582.2.2 BackDoorInitHook()	1808
7.582.2.3 CopInitHook()	1809
7.582.2.4 CopServiceHook()	1809
7.582.2.5 CpuUserProgramStartHook()	1809
7.582.2.6 FileFirmwareUpdateCompletedHook()	1809
7.582.2.7 FileFirmwareUpdateErrorHook()	1810
7.582.2.8 FileFirmwareUpdateLogHook()	1810
7.582.2.9 FileFirmwareUpdateStartedHook()	1810
7.582.2.10FileGetFirmwareFilenameHook()	1810
7.582.2.11FileIsFirmwareUpdateRequestedHook()	1811
7.582.2.12NvmDoneHook()	1811
7.582.2.13NvmEraseHook()	1811
7.582.2.14NvmInitHook()	1812
7.582.2.15NvmReinitHook()	1812
7.582.2.16NvmWriteHook()	1812
7.582.2.17XcpGetSeedHook()	1813
7.582.2.18XcpVerifyKeyHook()	1813
7.582.3 Variable Documentation	1813
7.582.3.1 canUse	1813
7.582.3.2 handle	1814
7.583hooks.c File Reference	1814
7.583.1 Detailed Description	1815
7.583.2 Function Documentation	1815
7.583.2.1 BackDoorEntryHook()	1816
7.583.2.2 BackDoorInitHook()	1816
7.583.2.3 CopInitHook()	1816
7.583.2.4 CopServiceHook()	1816
7.583.2.5 CpuUserProgramStartHook()	1817

7.583.2.6 FileFirmwareUpdateCompletedHook()	1817
7.583.2.7 FileFirmwareUpdateErrorHook()	1817
7.583.2.8 FileFirmwareUpdateLogHook()	1817
7.583.2.9 FileFirmwareUpdateStartedHook()	1818
7.583.2.10FileGetFirmwareFilenameHook()	1818
7.583.2.11FileIsFirmwareUpdateRequestedHook()	1818
7.583.2.12NvmDoneHook()	1819
7.583.2.13NvmEraseHook()	1819
7.583.2.14NvmInitHook()	1819
7.583.2.15NvmReinitHook()	1820
7.583.2.16NvmWriteHook()	1820
7.583.2.17XcpGetSeedHook()	1820
7.583.2.18XcpVerifyKeyHook()	1821
7.583.3 Variable Documentation	1821
7.583.3.1 canUse	1821
7.583.3.2 handle	1821
7.584hooks.c File Reference	1822
7.584.1 Detailed Description	1823
7.584.2 Function Documentation	1823
7.584.2.1 BackDoorEntryHook()	1823
7.584.2.2 BackDoorInitHook()	1824
7.584.2.3 CopInitHook()	1824
7.584.2.4 CopServiceHook()	1824
7.584.2.5 CpuUserProgramStartHook()	1824
7.584.2.6 FileFirmwareUpdateCompletedHook()	1825
7.584.2.7 FileFirmwareUpdateErrorHook()	1825
7.584.2.8 FileFirmwareUpdateLogHook()	1825
7.584.2.9 FileFirmwareUpdateStartedHook()	1825
7.584.2.10FileGetFirmwareFilenameHook()	1826
7.584.2.11FileIsFirmwareUpdateRequestedHook()	1826

7.584.2.12NvmDoneHook()	1826
7.584.2.13NvmEraseHook()	1826
7.584.2.14NvmInitHook()	1827
7.584.2.15NvmReinitHook()	1827
7.584.2.16NvmWriteHook()	1827
7.584.2.17XcpGetSeedHook()	1828
7.584.2.18XcpVerifyKeyHook()	1828
7.584.3 Variable Documentation	1829
7.584.3.1 canUse	1829
7.584.3.2 handle	1829
7.585hooks.c File Reference	1829
7.585.1 Detailed Description	1831
7.585.2 Function Documentation	1831
7.585.2.1 BackDoorEntryHook()	1831
7.585.2.2 BackDoorInitHook()	1831
7.585.2.3 CopInitHook()	1832
7.585.2.4 CopServiceHook()	1832
7.585.2.5 CpuUserProgramStartHook()	1832
7.585.2.6 FileFirmwareUpdateCompletedHook()	1832
7.585.2.7 FileFirmwareUpdateErrorHook()	1833
7.585.2.8 FileFirmwareUpdateLogHook()	1833
7.585.2.9 FileFirmwareUpdateStartedHook()	1833
7.585.2.10FileGetFirmwareFilenameHook()	1833
7.585.2.11FileIsFirmwareUpdateRequestedHook()	1834
7.585.2.12NvmDoneHook()	1834
7.585.2.13NvmEraseHook()	1834
7.585.2.14NvmInitHook()	1835
7.585.2.15NvmReinitHook()	1835
7.585.2.16NvmWriteHook()	1835
7.585.2.17UsbConnectHook()	1836

7.585.2.18	UsbEnterLowPowerModeHook()	1836
7.585.2.19	UsbLeaveLowPowerModeHook()	1836
7.585.2.20	XcpGetSeedHook()	1836
7.585.2.21	XcpVerifyKeyHook()	1837
7.585.3	Variable Documentation	1837
7.585.3.1	canUse	1837
7.585.3.2	handle	1837
7.586	hooks.c File Reference	1838
7.586.1	Detailed Description	1839
7.586.2	Function Documentation	1839
7.586.2.1	BackDoorEntryHook()	1840
7.586.2.2	BackDoorInitHook()	1840
7.586.2.3	CopInitHook()	1840
7.586.2.4	CopServiceHook()	1840
7.586.2.5	CpuUserProgramStartHook()	1841
7.586.2.6	FileFirmwareUpdateCompletedHook()	1841
7.586.2.7	FileFirmwareUpdateErrorHook()	1841
7.586.2.8	FileFirmwareUpdateLogHook()	1841
7.586.2.9	FileFirmwareUpdateStartedHook()	1842
7.586.2.10	FileGetFirmwareFilenameHook()	1842
7.586.2.11	FileIsFirmwareUpdateRequestedHook()	1842
7.586.2.12	NvmDoneHook()	1843
7.586.2.13	NvmEraseHook()	1843
7.586.2.14	NvmInitHook()	1843
7.586.2.15	NvmReinitHook()	1844
7.586.2.16	NvmWriteHook()	1844
7.586.2.17	UsbConnectHook()	1844
7.586.2.18	UsbEnterLowPowerModeHook()	1845
7.586.2.19	UsbLeaveLowPowerModeHook()	1845
7.586.2.20	XcpGetSeedHook()	1845

7.586.2.21XcpVerifyKeyHook()	1846
7.586.3 Variable Documentation	1846
7.586.3.1 canUse	1846
7.586.3.2 handle	1846
7.587hooks.c File Reference	1847
7.587.1 Detailed Description	1848
7.587.2 Function Documentation	1848
7.587.2.1 BackDoorEntryHook()	1849
7.587.2.2 BackDoorInitHook()	1849
7.587.2.3 CopInitHook()	1849
7.587.2.4 CopServiceHook()	1849
7.587.2.5 CpuUserProgramStartHook()	1850
7.587.2.6 FileFirmwareUpdateCompletedHook()	1850
7.587.2.7 FileFirmwareUpdateErrorHook()	1850
7.587.2.8 FileFirmwareUpdateLogHook()	1850
7.587.2.9 FileFirmwareUpdateStartedHook()	1851
7.587.2.10FileGetFirmwareFilenameHook()	1851
7.587.2.11FileIsFirmwareUpdateRequestedHook()	1851
7.587.2.12NvmDoneHook()	1852
7.587.2.13NvmEraseHook()	1852
7.587.2.14NvmInitHook()	1852
7.587.2.15NvmReinitHook()	1853
7.587.2.16NvmWriteHook()	1853
7.587.2.17UsbConnectHook()	1853
7.587.2.18UsbEnterLowPowerModeHook()	1854
7.587.2.19UsbLeaveLowPowerModeHook()	1854
7.587.2.20XcpGetSeedHook()	1854
7.587.2.21XcpVerifyKeyHook()	1855
7.587.3 Variable Documentation	1855
7.587.3.1 canUse	1855

7.587.3.2 handle	1855
7.588hooks.c File Reference	1856
7.588.1 Detailed Description	1857
7.588.2 Function Documentation	1857
7.588.2.1 BackDoorEntryHook()	1858
7.588.2.2 BackDoorInitHook()	1858
7.588.2.3 CopInitHook()	1858
7.588.2.4 CopServiceHook()	1858
7.588.2.5 CpuUserProgramStartHook()	1859
7.588.2.6 FileFirmwareUpdateCompletedHook()	1859
7.588.2.7 FileFirmwareUpdateErrorHook()	1859
7.588.2.8 FileFirmwareUpdateLogHook()	1859
7.588.2.9 FileFirmwareUpdateStartedHook()	1860
7.588.2.10FileGetFirmwareFilenameHook()	1860
7.588.2.11FileIsFirmwareUpdateRequestedHook()	1860
7.588.2.12NvmDoneHook()	1861
7.588.2.13NvmEraseHook()	1861
7.588.2.14NvmInitHook()	1861
7.588.2.15NvmReinitHook()	1862
7.588.2.16NvmWriteHook()	1862
7.588.2.17UsbConnectHook()	1862
7.588.2.18UsbEnterLowPowerModeHook()	1863
7.588.2.19UsbLeaveLowPowerModeHook()	1863
7.588.2.20XcpGetSeedHook()	1863
7.588.2.21XcpVerifyKeyHook()	1864
7.588.3 Variable Documentation	1864
7.588.3.1 canUse	1864
7.588.3.2 handle	1864
7.589hooks.c File Reference	1865
7.589.1 Detailed Description	1866

7.589.2 Function Documentation	1866
7.589.2.1 BackDoorEntryHook()	1866
7.589.2.2 BackDoorInitHook()	1867
7.589.2.3 CopInitHook()	1867
7.589.2.4 CopServiceHook()	1867
7.589.2.5 CpuUserProgramStartHook()	1867
7.589.2.6 FileFirmwareUpdateCompletedHook()	1868
7.589.2.7 FileFirmwareUpdateErrorHook()	1868
7.589.2.8 FileFirmwareUpdateLogHook()	1868
7.589.2.9 FileFirmwareUpdateStartedHook()	1868
7.589.2.10 FileGetFirmwareFilenameHook()	1869
7.589.2.11 FileIsFirmwareUpdateRequestedHook()	1869
7.589.2.12 NvmDoneHook()	1869
7.589.2.13 NvmEraseHook()	1869
7.589.2.14 NvmInitHook()	1870
7.589.2.15 NvmReinitHook()	1870
7.589.2.16 NvmWriteHook()	1870
7.589.2.17 XcpGetSeedHook()	1871
7.589.2.18 XcpVerifyKeyHook()	1871
7.589.3 Variable Documentation	1872
7.589.3.1 canUse	1872
7.589.3.2 handle	1872
7.590hooks.c File Reference	1872
7.590.1 Detailed Description	1874
7.590.2 Function Documentation	1874
7.590.2.1 BackDoorEntryHook()	1874
7.590.2.2 BackDoorInitHook()	1874
7.590.2.3 CopInitHook()	1875
7.590.2.4 CopServiceHook()	1875
7.590.2.5 CpuUserProgramStartHook()	1875

7.590.2.6 FileFirmwareUpdateCompletedHook()	1875
7.590.2.7 FileFirmwareUpdateErrorHook()	1876
7.590.2.8 FileFirmwareUpdateLogHook()	1876
7.590.2.9 FileFirmwareUpdateStartedHook()	1876
7.590.2.10 FileGetFirmwareFilenameHook()	1876
7.590.2.11 FileIsFirmwareUpdateRequestedHook()	1877
7.590.2.12 NvmDoneHook()	1877
7.590.2.13 NvmEraseHook()	1877
7.590.2.14 NvmInitHook()	1878
7.590.2.15 NvmReinitHook()	1878
7.590.2.16 NvmWriteHook()	1878
7.590.2.17 XcpGetSeedHook()	1879
7.590.2.18 XcpVerifyKeyHook()	1879
7.590.3 Variable Documentation	1879
7.590.3.1 canUse	1879
7.590.3.2 handle	1880
7.591 hooks.c File Reference	1880
7.591.1 Detailed Description	1881
7.591.2 Function Documentation	1881
7.591.2.1 BackDoorEntryHook()	1882
7.591.2.2 BackDoorInitHook()	1882
7.591.2.3 CopInitHook()	1882
7.591.2.4 CopServiceHook()	1882
7.591.2.5 CpuUserProgramStartHook()	1883
7.591.2.6 FileFirmwareUpdateCompletedHook()	1883
7.591.2.7 FileFirmwareUpdateErrorHook()	1883
7.591.2.8 FileFirmwareUpdateLogHook()	1883
7.591.2.9 FileFirmwareUpdateStartedHook()	1884
7.591.2.10 FileGetFirmwareFilenameHook()	1884
7.591.2.11 FileIsFirmwareUpdateRequestedHook()	1884

7.591.2.12NvmDoneHook()	1885
7.591.2.13NvmEraseHook()	1885
7.591.2.14NvmInitHook()	1885
7.591.2.15NvmReinitHook()	1886
7.591.2.16NvmWriteHook()	1886
7.591.2.17XcpGetSeedHook()	1886
7.591.2.18XcpVerifyKeyHook()	1887
7.591.3 Variable Documentation	1887
7.591.3.1 canUse	1887
7.591.3.2 handle	1887
7.592hooks.c File Reference	1888
7.592.1 Detailed Description	1889
7.592.2 Function Documentation	1889
7.592.2.1 BackDoorEntryHook()	1889
7.592.2.2 BackDoorInitHook()	1890
7.592.2.3 CopInitHook()	1890
7.592.2.4 CopServiceHook()	1890
7.592.2.5 CpuUserProgramStartHook()	1890
7.592.2.6 FileFirmwareUpdateCompletedHook()	1891
7.592.2.7 FileFirmwareUpdateErrorHook()	1891
7.592.2.8 FileFirmwareUpdateLogHook()	1891
7.592.2.9 FileFirmwareUpdateStartedHook()	1891
7.592.2.10FileGetFirmwareFilenameHook()	1892
7.592.2.11FileIsFirmwareUpdateRequestedHook()	1892
7.592.2.12NvmDoneHook()	1892
7.592.2.13NvmEraseHook()	1892
7.592.2.14NvmInitHook()	1893
7.592.2.15NvmReinitHook()	1893
7.592.2.16NvmWriteHook()	1893
7.592.2.17XcpGetSeedHook()	1894

7.592.2.18XcpVerifyKeyHook()	1894
7.592.3 Variable Documentation	1895
7.592.3.1 canUse	1895
7.592.3.2 handle	1895
7.593hooks.c File Reference	1895
7.593.1 Detailed Description	1896
7.593.2 Function Documentation	1896
7.593.2.1 BackDoorEntryHook()	1897
7.593.2.2 BackDoorInitHook()	1897
7.593.2.3 CopInitHook()	1897
7.593.2.4 CopServiceHook()	1897
7.593.2.5 CpuUserProgramStartHook()	1898
7.593.2.6 NvmDoneHook()	1898
7.593.2.7 NvmEraseHook()	1898
7.593.2.8 NvmInitHook()	1899
7.593.2.9 NvmReinitHook()	1899
7.593.2.10NvmWriteHook()	1899
7.593.2.11XcpGetSeedHook()	1900
7.593.2.12XcpVerifyKeyHook()	1900
7.594hooks.c File Reference	1900
7.594.1 Detailed Description	1902
7.594.2 Function Documentation	1902
7.594.2.1 BackDoorEntryHook()	1902
7.594.2.2 BackDoorInitHook()	1902
7.594.2.3 CopInitHook()	1902
7.594.2.4 CopServiceHook()	1903
7.594.2.5 CpuUserProgramStartHook()	1903
7.594.2.6 NvmDoneHook()	1903
7.594.2.7 NvmEraseHook()	1903
7.594.2.8 NvmInitHook()	1904

7.594.2.9 NvmReinitHook()	1904
7.594.2.10NvmWriteHook()	1904
7.594.2.11XcpGetSeedHook()	1905
7.594.2.12XcpVerifyKeyHook()	1905
7.595hooks.c File Reference	1906
7.595.1 Detailed Description	1907
7.595.2 Function Documentation	1907
7.595.2.1 BackDoorEntryHook()	1907
7.595.2.2 BackDoorInitHook()	1907
7.595.2.3 CopInitHook()	1908
7.595.2.4 CopServiceHook()	1908
7.595.2.5 CpuUserProgramStartHook()	1908
7.595.2.6 NvmDoneHook()	1908
7.595.2.7 NvmEraseHook()	1909
7.595.2.8 NvmInitHook()	1909
7.595.2.9 NvmReinitHook()	1909
7.595.2.10NvmWriteHook()	1909
7.595.2.11UsbConnectHook()	1910
7.595.2.12UsbEnterLowPowerModeHook()	1910
7.595.2.13UsbLeaveLowPowerModeHook()	1910
7.595.2.14XcpGetSeedHook()	1911
7.595.2.15XcpVerifyKeyHook()	1911
7.596hooks.c File Reference	1911
7.596.1 Detailed Description	1913
7.596.2 Function Documentation	1913
7.596.2.1 BackDoorEntryHook()	1913
7.596.2.2 BackDoorInitHook()	1913
7.596.2.3 CopInitHook()	1913
7.596.2.4 CopServiceHook()	1914
7.596.2.5 CpuUserProgramStartHook()	1914

7.596.2.6 NvmDoneHook()	1914
7.596.2.7 NvmEraseHook()	1914
7.596.2.8 NvmInitHook()	1915
7.596.2.9 NvmReinitHook()	1915
7.596.2.10 NvmWriteHook()	1915
7.596.2.11 UsbConnectHook()	1916
7.596.2.12 UsbEnterLowPowerModeHook()	1916
7.596.2.13 UsbLeaveLowPowerModeHook()	1916
7.596.2.14 XcpGetSeedHook()	1917
7.596.2.15 XcpVerifyKeyHook()	1917
7.597hooks.c File Reference	1917
7.597.1 Detailed Description	1919
7.597.2 Function Documentation	1919
7.597.2.1 BackDoorEntryHook()	1919
7.597.2.2 BackDoorInitHook()	1919
7.597.2.3 CopInitHook()	1919
7.597.2.4 CopServiceHook()	1920
7.597.2.5 CpuUserProgramStartHook()	1920
7.597.2.6 NvmDoneHook()	1920
7.597.2.7 NvmEraseHook()	1920
7.597.2.8 NvmInitHook()	1921
7.597.2.9 NvmReinitHook()	1921
7.597.2.10 NvmWriteHook()	1921
7.597.2.11 UsbConnectHook()	1922
7.597.2.12 UsbEnterLowPowerModeHook()	1922
7.597.2.13 UsbLeaveLowPowerModeHook()	1922
7.597.2.14 XcpGetSeedHook()	1923
7.597.2.15 XcpVerifyKeyHook()	1923
7.598hooks.c File Reference	1923
7.598.1 Detailed Description	1925

7.598.2 Function Documentation	1925
7.598.2.1 BackDoorEntryHook()	1925
7.598.2.2 BackDoorInitHook()	1925
7.598.2.3 CopInitHook()	1925
7.598.2.4 CopServiceHook()	1926
7.598.2.5 CpuUserProgramStartHook()	1926
7.598.2.6 NvmDoneHook()	1926
7.598.2.7 NvmEraseHook()	1926
7.598.2.8 NvmInitHook()	1927
7.598.2.9 NvmReinitHook()	1927
7.598.2.10NvmWriteHook()	1927
7.598.2.11UsbConnectHook()	1928
7.598.2.12UsbEnterLowPowerModeHook()	1928
7.598.2.13UsbLeaveLowPowerModeHook()	1928
7.598.2.14XcpGetSeedHook()	1929
7.598.2.15XcpVerifyKeyHook()	1929
7.599hooks.c File Reference	1929
7.599.1 Detailed Description	1931
7.599.2 Function Documentation	1931
7.599.2.1 BackDoorEntryHook()	1931
7.599.2.2 BackDoorInitHook()	1931
7.599.2.3 CopInitHook()	1931
7.599.2.4 CopServiceHook()	1932
7.599.2.5 CpuUserProgramStartHook()	1932
7.599.2.6 NvmDoneHook()	1932
7.599.2.7 NvmEraseHook()	1932
7.599.2.8 NvmInitHook()	1933
7.599.2.9 NvmReinitHook()	1933
7.599.2.10NvmWriteHook()	1933
7.599.2.11XcpGetSeedHook()	1934

7.599.2.12XcpVerifyKeyHook()	1934
7.600hooks.c File Reference	1935
7.600.1 Detailed Description	1936
7.600.2 Function Documentation	1936
7.600.2.1 BackDoorEntryHook()	1936
7.600.2.2 BackDoorInitHook()	1936
7.600.2.3 CopInitHook()	1937
7.600.2.4 CopServiceHook()	1937
7.600.2.5 CpuUserProgramStartHook()	1937
7.600.2.6 NvmDoneHook()	1937
7.600.2.7 NvmEraseHook()	1938
7.600.2.8 NvmInitHook()	1938
7.600.2.9 NvmReinitHook()	1938
7.600.2.10NvmWriteHook()	1938
7.600.2.11XcpGetSeedHook()	1939
7.600.2.12XcpVerifyKeyHook()	1939
7.601hooks.c File Reference	1940
7.601.1 Detailed Description	1941
7.601.2 Function Documentation	1941
7.601.2.1 BackDoorEntryHook()	1941
7.601.2.2 BackDoorInitHook()	1941
7.601.2.3 CopInitHook()	1942
7.601.2.4 CopServiceHook()	1942
7.601.2.5 CpuUserProgramStartHook()	1942
7.601.2.6 NvmDoneHook()	1942
7.601.2.7 NvmEraseHook()	1943
7.601.2.8 NvmInitHook()	1943
7.601.2.9 NvmReinitHook()	1943
7.601.2.10NvmWriteHook()	1943
7.601.2.11XcpGetSeedHook()	1944

7.601.2.12XcpVerifyKeyHook()	1944
7.602hooks.c File Reference	1945
7.602.1 Detailed Description	1946
7.602.2 Function Documentation	1946
7.602.2.1 BackDoorEntryHook()	1946
7.602.2.2 BackDoorInitHook()	1946
7.602.2.3 CopInitHook()	1947
7.602.2.4 CopServiceHook()	1947
7.602.2.5 CpuUserProgramStartHook()	1947
7.602.2.6 NvmDoneHook()	1947
7.602.2.7 NvmEraseHook()	1948
7.602.2.8 NvmInitHook()	1948
7.602.2.9 NvmReinitHook()	1948
7.602.2.10NvmWriteHook()	1948
7.602.2.11XcpGetSeedHook()	1949
7.602.2.12XcpVerifyKeyHook()	1949
7.603hooks.c File Reference	1950
7.603.1 Detailed Description	1951
7.603.2 Function Documentation	1951
7.603.2.1 BackDoorEntryHook()	1951
7.603.2.2 BackDoorInitHook()	1951
7.603.2.3 CopInitHook()	1952
7.603.2.4 CopServiceHook()	1952
7.603.2.5 CpuUserProgramStartHook()	1952
7.603.2.6 NvmDoneHook()	1952
7.603.2.7 NvmEraseHook()	1953
7.603.2.8 NvmInitHook()	1953
7.603.2.9 NvmReinitHook()	1953
7.603.2.10NvmWriteHook()	1953
7.603.2.11UsbConnectHook()	1954

7.603.2.12	UsbEnterLowPowerModeHook()	1954
7.603.2.13	UsbLeaveLowPowerModeHook()	1954
7.603.2.14	XcpGetSeedHook()	1955
7.603.2.15	XcpVerifyKeyHook()	1955
7.604	hooks.c File Reference	1955
7.604.1	Detailed Description	1957
7.604.2	Function Documentation	1957
7.604.2.1	BackDoorEntryHook()	1957
7.604.2.2	BackDoorInitHook()	1957
7.604.2.3	CopInitHook()	1958
7.604.2.4	CopServiceHook()	1958
7.604.2.5	CpuUserProgramStartHook()	1958
7.604.2.6	NvmDoneHook()	1958
7.604.2.7	NvmEraseHook()	1959
7.604.2.8	NvmInitHook()	1959
7.604.2.9	NvmReinitHook()	1959
7.604.2.10	NvmWriteHook()	1959
7.604.2.11	UsbConnectHook()	1960
7.604.2.12	UsbEnterLowPowerModeHook()	1960
7.604.2.13	UsbLeaveLowPowerModeHook()	1960
7.604.2.14	XcpGetSeedHook()	1961
7.604.2.15	XcpVerifyKeyHook()	1961
7.605	hooks.c File Reference	1961
7.605.1	Detailed Description	1963
7.605.2	Function Documentation	1963
7.605.2.1	BackDoorEntryHook()	1963
7.605.2.2	BackDoorInitHook()	1963
7.605.2.3	CopInitHook()	1964
7.605.2.4	CopServiceHook()	1964
7.605.2.5	CpuUserProgramStartHook()	1964

7.605.2.6 NvmDoneHook()	1964
7.605.2.7 NvmEraseHook()	1965
7.605.2.8 NvmInitHook()	1965
7.605.2.9 NvmReinitHook()	1965
7.605.2.10 NvmWriteHook()	1965
7.605.2.11 UsbConnectHook()	1966
7.605.2.12 UsbEnterLowPowerModeHook()	1966
7.605.2.13 UsbLeaveLowPowerModeHook()	1966
7.605.2.14 XcpGetSeedHook()	1967
7.605.2.15 XcpVerifyKeyHook()	1967
7.606hooks.c File Reference	1967
7.606.1 Detailed Description	1969
7.606.2 Function Documentation	1969
7.606.2.1 BackDoorEntryHook()	1969
7.606.2.2 BackDoorInitHook()	1969
7.606.2.3 CopInitHook()	1970
7.606.2.4 CopServiceHook()	1970
7.606.2.5 CpuUserProgramStartHook()	1970
7.606.2.6 NvmDoneHook()	1970
7.606.2.7 NvmEraseHook()	1971
7.606.2.8 NvmInitHook()	1971
7.606.2.9 NvmReinitHook()	1971
7.606.2.10 NvmWriteHook()	1971
7.606.2.11 UsbConnectHook()	1972
7.606.2.12 UsbEnterLowPowerModeHook()	1972
7.606.2.13 UsbLeaveLowPowerModeHook()	1972
7.606.2.14 XcpGetSeedHook()	1973
7.606.2.15 XcpVerifyKeyHook()	1973
7.607hooks.c File Reference	1973
7.607.1 Detailed Description	1975

7.607.2 Function Documentation	1975
7.607.2.1 BackDoorEntryHook()	1975
7.607.2.2 BackDoorInitHook()	1976
7.607.2.3 CopInitHook()	1976
7.607.2.4 CopServiceHook()	1976
7.607.2.5 CpuUserProgramStartHook()	1976
7.607.2.6 FileFirmwareUpdateCompletedHook()	1977
7.607.2.7 FileFirmwareUpdateErrorHook()	1977
7.607.2.8 FileFirmwareUpdateLogHook()	1977
7.607.2.9 FileFirmwareUpdateStartedHook()	1977
7.607.2.10 FileGetFirmwareFilenameHook()	1978
7.607.2.11 FileIsFirmwareUpdateRequestedHook()	1978
7.607.2.12 NvmDoneHook()	1978
7.607.2.13 NvmEraseHook()	1978
7.607.2.14 NvmInitHook()	1979
7.607.2.15 NvmReinitHook()	1979
7.607.2.16 NvmWriteHook()	1979
7.607.2.17 UsbConnectHook()	1980
7.607.2.18 UsbEnterLowPowerModeHook()	1980
7.607.2.19 UsbLeaveLowPowerModeHook()	1980
7.607.2.20 XcpGetSeedHook()	1981
7.607.2.21 XcpVerifyKeyHook()	1981
7.607.3 Variable Documentation	1981
7.607.3.1 canUse	1981
7.607.3.2 handle	1982
7.608hooks.c File Reference	1982
7.608.1 Detailed Description	1983
7.608.2 Function Documentation	1984
7.608.2.1 BackDoorEntryHook()	1984
7.608.2.2 BackDoorInitHook()	1984

7.608.2.3 CopInitHook()	1984
7.608.2.4 CopServiceHook()	1985
7.608.2.5 CpuUserProgramStartHook()	1985
7.608.2.6 FileFirmwareUpdateCompletedHook()	1985
7.608.2.7 FileFirmwareUpdateErrorHook()	1985
7.608.2.8 FileFirmwareUpdateLogHook()	1985
7.608.2.9 FileFirmwareUpdateStartedHook()	1986
7.608.2.10 FileGetFirmwareFilenameHook()	1986
7.608.2.11 FileIsFirmwareUpdateRequestedHook()	1986
7.608.2.12 NvmDoneHook()	1987
7.608.2.13 NvmEraseHook()	1987
7.608.2.14 NvmInitHook()	1987
7.608.2.15 NvmReinitHook()	1988
7.608.2.16 NvmWriteHook()	1988
7.608.2.17 UsbConnectHook()	1988
7.608.2.18 UsbEnterLowPowerModeHook()	1989
7.608.2.19 UsbLeaveLowPowerModeHook()	1989
7.608.2.20 XcpGetSeedHook()	1989
7.608.2.21 XcpVerifyKeyHook()	1990
7.608.3 Variable Documentation	1990
7.608.3.1 canUse	1990
7.608.3.2 handle	1990
7.609hooks.c File Reference	1991
7.609.1 Detailed Description	1992
7.609.2 Function Documentation	1992
7.609.2.1 BackDoorEntryHook()	1993
7.609.2.2 BackDoorInitHook()	1993
7.609.2.3 CopInitHook()	1993
7.609.2.4 CopServiceHook()	1993
7.609.2.5 CpuUserProgramStartHook()	1994

7.609.2.6 FileFirmwareUpdateCompletedHook()	1994
7.609.2.7 FileFirmwareUpdateErrorHook()	1994
7.609.2.8 FileFirmwareUpdateLogHook()	1994
7.609.2.9 FileFirmwareUpdateStartedHook()	1995
7.609.2.10 FileGetFirmwareFilenameHook()	1995
7.609.2.11 FileIsFirmwareUpdateRequestedHook()	1995
7.609.2.12 NvmDoneHook()	1996
7.609.2.13 NvmEraseHook()	1996
7.609.2.14 NvmInitHook()	1996
7.609.2.15 NvmReinitHook()	1997
7.609.2.16 NvmWriteHook()	1997
7.609.2.17 UsbConnectHook()	1997
7.609.2.18 UsbEnterLowPowerModeHook()	1998
7.609.2.19 UsbLeaveLowPowerModeHook()	1998
7.609.2.20 XcpGetSeedHook()	1998
7.609.2.21 XcpVerifyKeyHook()	1999
7.609.3 Variable Documentation	1999
7.609.3.1 canUse	1999
7.609.3.2 handle	1999
7.610hooks.c File Reference	2000
7.610.1 Detailed Description	2001
7.610.2 Function Documentation	2001
7.610.2.1 BackDoorEntryHook()	2002
7.610.2.2 BackDoorInitHook()	2002
7.610.2.3 CopInitHook()	2002
7.610.2.4 CopServiceHook()	2002
7.610.2.5 CpuUserProgramStartHook()	2003
7.610.2.6 FileFirmwareUpdateCompletedHook()	2003
7.610.2.7 FileFirmwareUpdateErrorHook()	2003
7.610.2.8 FileFirmwareUpdateLogHook()	2003

7.610.2.9 FileFirmwareUpdateStartedHook()	2004
7.610.2.10FileGetFirmwareFilenameHook()	2004
7.610.2.11FileIsFirmwareUpdateRequestedHook()	2004
7.610.2.12NvmDoneHook()	2005
7.610.2.13NvmEraseHook()	2005
7.610.2.14NvmInitHook()	2005
7.610.2.15NvmReinitHook()	2006
7.610.2.16NvmWriteHook()	2006
7.610.2.17UsbConnectHook()	2006
7.610.2.18UsbEnterLowPowerModeHook()	2007
7.610.2.19UsbLeaveLowPowerModeHook()	2007
7.610.2.20XcpGetSeedHook()	2007
7.610.2.21XcpVerifyKeyHook()	2008
7.610.3 Variable Documentation	2008
7.610.3.1 canUse	2008
7.610.3.2 handle	2008
7.611 hooks.c File Reference	2009
7.611.1 Detailed Description	2010
7.611.2 Function Documentation	2010
7.611.2.1 BackDoorEntryHook()	2010
7.611.2.2 BackDoorInitHook()	2010
7.611.2.3 CopInitHook()	2011
7.611.2.4 CopServiceHook()	2011
7.611.2.5 CpuUserProgramStartHook()	2011
7.611.2.6 NvmDoneHook()	2011
7.611.2.7 NvmEraseHook()	2012
7.611.2.8 NvmInitHook()	2012
7.611.2.9 NvmReinitHook()	2012
7.611.2.10NvmWriteHook()	2012
7.611.2.11XcpGetSeedHook()	2013

7.611.2.12XcpVerifyKeyHook()	2013
7.612hooks.c File Reference	2014
7.612.1 Detailed Description	2015
7.612.2 Function Documentation	2015
7.612.2.1 BackDoorEntryHook()	2015
7.612.2.2 BackDoorInitHook()	2015
7.612.2.3 CopInitHook()	2016
7.612.2.4 CopServiceHook()	2016
7.612.2.5 CpuUserProgramStartHook()	2016
7.612.2.6 NvmDoneHook()	2016
7.612.2.7 NvmEraseHook()	2017
7.612.2.8 NvmInitHook()	2017
7.612.2.9 NvmReinitHook()	2017
7.612.2.10NvmWriteHook()	2017
7.612.2.11XcpGetSeedHook()	2018
7.612.2.12XcpVerifyKeyHook()	2018
7.613hooks.c File Reference	2019
7.613.1 Detailed Description	2020
7.613.2 Function Documentation	2020
7.613.2.1 BackDoorEntryHook()	2020
7.613.2.2 BackDoorInitHook()	2020
7.613.2.3 CopInitHook()	2021
7.613.2.4 CopServiceHook()	2021
7.613.2.5 CpuUserProgramStartHook()	2021
7.613.2.6 NvmDoneHook()	2021
7.613.2.7 NvmEraseHook()	2022
7.613.2.8 NvmInitHook()	2022
7.613.2.9 NvmReinitHook()	2022
7.613.2.10NvmWriteHook()	2022
7.613.2.11XcpGetSeedHook()	2023

7.613.2.12XcpVerifyKeyHook()	2023
7.614hooks.c File Reference	2024
7.614.1 Detailed Description	2025
7.614.2 Function Documentation	2025
7.614.2.1 BackDoorEntryHook()	2025
7.614.2.2 BackDoorInitHook()	2025
7.614.2.3 CopInitHook()	2026
7.614.2.4 CopServiceHook()	2026
7.614.2.5 CpuUserProgramStartHook()	2026
7.614.2.6 NvmDoneHook()	2026
7.614.2.7 NvmEraseHook()	2027
7.614.2.8 NvmInitHook()	2027
7.614.2.9 NvmReinitHook()	2027
7.614.2.10NvmWriteHook()	2027
7.614.2.11XcpGetSeedHook()	2028
7.614.2.12XcpVerifyKeyHook()	2028
7.615hooks.c File Reference	2029
7.615.1 Detailed Description	2030
7.615.2 Function Documentation	2031
7.615.2.1 BackDoorEntryHook()	2031
7.615.2.2 BackDoorInitHook()	2031
7.615.2.3 CopInitHook()	2031
7.615.2.4 CopServiceHook()	2032
7.615.2.5 CpuUserProgramStartHook()	2032
7.615.2.6 FileFirmwareUpdateCompletedHook()	2032
7.615.2.7 FileFirmwareUpdateErrorHook()	2032
7.615.2.8 FileFirmwareUpdateLogHook()	2032
7.615.2.9 FileFirmwareUpdateStartedHook()	2033
7.615.2.10FileGetFirmwareFilenameHook()	2033
7.615.2.11FileIsFirmwareUpdateRequestedHook()	2033

7.615.2.12NvmDoneHook()	2034
7.615.2.13NvmEraseHook()	2034
7.615.2.14NvmInitHook()	2034
7.615.2.15NvmReinitHook()	2035
7.615.2.16NvmWriteHook()	2035
7.615.2.17UsbConnectHook()	2035
7.615.2.18UsbEnterLowPowerModeHook()	2036
7.615.2.19UsbLeaveLowPowerModeHook()	2036
7.615.2.20XcpGetSeedHook()	2036
7.615.2.21XcpVerifyKeyHook()	2037
7.615.3 Variable Documentation	2037
7.615.3.1 canUse	2037
7.615.3.2 handle	2037
7.616hooks.c File Reference	2038
7.616.1 Detailed Description	2039
7.616.2 Function Documentation	2039
7.616.2.1 BackDoorEntryHook()	2039
7.616.2.2 BackDoorInitHook()	2040
7.616.2.3 CopInitHook()	2040
7.616.2.4 CopServiceHook()	2040
7.616.2.5 CpuUserProgramStartHook()	2040
7.616.2.6 FileFirmwareUpdateCompletedHook()	2041
7.616.2.7 FileFirmwareUpdateErrorHook()	2041
7.616.2.8 FileFirmwareUpdateLogHook()	2041
7.616.2.9 FileFirmwareUpdateStartedHook()	2041
7.616.2.10FileGetFirmwareFilenameHook()	2042
7.616.2.11FileIsFirmwareUpdateRequestedHook()	2042
7.616.2.12NvmDoneHook()	2042
7.616.2.13NvmEraseHook()	2042
7.616.2.14NvmInitHook()	2043

7.616.2.15NvmReinitHook()	2043
7.616.2.16NvmWriteHook()	2043
7.616.2.17XcpGetSeedHook()	2044
7.616.2.18XcpVerifyKeyHook()	2044
7.616.3 Variable Documentation	2045
7.616.3.1 canUse	2045
7.616.3.2 handle	2045
7.617hooks.c File Reference	2045
7.617.1 Detailed Description	2047
7.617.2 Function Documentation	2047
7.617.2.1 BackDoorEntryHook()	2047
7.617.2.2 BackDoorInitHook()	2047
7.617.2.3 CopInitHook()	2048
7.617.2.4 CopServiceHook()	2048
7.617.2.5 CpuUserProgramStartHook()	2048
7.617.2.6 FileFirmwareUpdateCompletedHook()	2048
7.617.2.7 FileFirmwareUpdateErrorHook()	2049
7.617.2.8 FileFirmwareUpdateLogHook()	2049
7.617.2.9 FileFirmwareUpdateStartedHook()	2049
7.617.2.10FileGetFirmwareFilenameHook()	2050
7.617.2.11FileIsFirmwareUpdateRequestedHook()	2050
7.617.2.12NvmDoneHook()	2050
7.617.2.13NvmEraseHook()	2050
7.617.2.14NvmInitHook()	2051
7.617.2.15NvmReinitHook()	2051
7.617.2.16NvmWriteHook()	2051
7.617.2.17XcpGetSeedHook()	2052
7.617.2.18XcpVerifyKeyHook()	2052
7.617.3 Variable Documentation	2053
7.617.3.1 canUse	2053

7.617.3.2 handle	2053
7.618hooks.c File Reference	2053
7.618.1 Detailed Description	2054
7.618.2 Function Documentation	2054
7.618.2.1 BackDoorEntryHook()	2055
7.618.2.2 BackDoorInitHook()	2055
7.618.2.3 CopInitHook()	2055
7.618.2.4 CopServiceHook()	2055
7.618.2.5 CpuUserProgramStartHook()	2056
7.618.2.6 NvmDoneHook()	2056
7.618.2.7 NvmEraseHook()	2056
7.618.2.8 NvmInitHook()	2057
7.618.2.9 NvmReinitHook()	2057
7.618.2.10 NvmWriteHook()	2057
7.618.2.11 UsbConnectHook()	2058
7.618.2.12 UsbEnterLowPowerModeHook()	2058
7.618.2.13 UsbLeaveLowPowerModeHook()	2058
7.618.2.14 XcpGetSeedHook()	2058
7.618.2.15 XcpVerifyKeyHook()	2059
7.619hooks.c File Reference	2059
7.619.1 Detailed Description	2061
7.619.2 Function Documentation	2061
7.619.2.1 BackDoorEntryHook()	2061
7.619.2.2 BackDoorInitHook()	2061
7.619.2.3 CopInitHook()	2061
7.619.2.4 CopServiceHook()	2062
7.619.2.5 CpuUserProgramStartHook()	2062
7.619.2.6 NvmDoneHook()	2062
7.619.2.7 NvmEraseHook()	2062
7.619.2.8 NvmInitHook()	2063

7.619.2.9 NvmReinitHook()	2063
7.619.2.10NvmWriteHook()	2063
7.619.2.11UsbConnectHook()	2064
7.619.2.12UsbEnterLowPowerModeHook()	2064
7.619.2.13UsbLeaveLowPowerModeHook()	2064
7.619.2.14XcpGetSeedHook()	2065
7.619.2.15XcpVerifyKeyHook()	2065
7.620hooks.c File Reference	2065
7.620.1 Detailed Description	2067
7.620.2 Function Documentation	2067
7.620.2.1 BackDoorEntryHook()	2067
7.620.2.2 BackDoorInitHook()	2067
7.620.2.3 CopInitHook()	2067
7.620.2.4 CopServiceHook()	2068
7.620.2.5 CpuUserProgramStartHook()	2068
7.620.2.6 NvmDoneHook()	2068
7.620.2.7 NvmEraseHook()	2068
7.620.2.8 NvmInitHook()	2069
7.620.2.9 NvmReinitHook()	2069
7.620.2.10NvmWriteHook()	2069
7.620.2.11UsbConnectHook()	2070
7.620.2.12UsbEnterLowPowerModeHook()	2070
7.620.2.13UsbLeaveLowPowerModeHook()	2070
7.620.2.14XcpGetSeedHook()	2071
7.620.2.15XcpVerifyKeyHook()	2071
7.621hooks.c File Reference	2071
7.621.1 Detailed Description	2073
7.621.2 Function Documentation	2073
7.621.2.1 BackDoorEntryHook()	2073
7.621.2.2 BackDoorInitHook()	2073

7.621.2.3 CopInitHook()	2073
7.621.2.4 CopServiceHook()	2074
7.621.2.5 CpuUserProgramStartHook()	2074
7.621.2.6 NvmDoneHook()	2074
7.621.2.7 NvmEraseHook()	2074
7.621.2.8 NvmInitHook()	2075
7.621.2.9 NvmReinitHook()	2075
7.621.2.10NvmWriteHook()	2075
7.621.2.11UsbConnectHook()	2076
7.621.2.12UsbEnterLowPowerModeHook()	2076
7.621.2.13UsbLeaveLowPowerModeHook()	2076
7.621.2.14XcpGetSeedHook()	2077
7.621.2.15XcpVerifyKeyHook()	2077
7.622hooks.c File Reference	2077
7.622.1 Detailed Description	2079
7.622.2 Function Documentation	2079
7.622.2.1 BackDoorEntryHook()	2079
7.622.2.2 BackDoorInitHook()	2079
7.622.2.3 CopInitHook()	2079
7.622.2.4 CopServiceHook()	2080
7.622.2.5 CpuUserProgramStartHook()	2080
7.622.2.6 NvmDoneHook()	2080
7.622.2.7 NvmEraseHook()	2080
7.622.2.8 NvmInitHook()	2081
7.622.2.9 NvmReinitHook()	2081
7.622.2.10NvmWriteHook()	2081
7.622.2.11XcpGetSeedHook()	2082
7.622.2.12XcpVerifyKeyHook()	2082
7.623hooks.c File Reference	2083
7.623.1 Detailed Description	2084

7.623.2 Function Documentation	2084
7.623.2.1 BackDoorEntryHook()	2084
7.623.2.2 BackDoorInitHook()	2084
7.623.2.3 CopInitHook()	2085
7.623.2.4 CopServiceHook()	2085
7.623.2.5 CpuUserProgramStartHook()	2085
7.623.2.6 NvmDoneHook()	2085
7.623.2.7 NvmEraseHook()	2086
7.623.2.8 NvmInitHook()	2086
7.623.2.9 NvmReinitHook()	2086
7.623.2.10 NvmWriteHook()	2086
7.623.2.11 XcpGetSeedHook()	2087
7.623.2.12 XcpVerifyKeyHook()	2087
7.624hooks.c File Reference	2088
7.624.1 Detailed Description	2089
7.624.2 Function Documentation	2089
7.624.2.1 BackDoorEntryHook()	2089
7.624.2.2 BackDoorInitHook()	2089
7.624.2.3 CopInitHook()	2090
7.624.2.4 CopServiceHook()	2090
7.624.2.5 CpuUserProgramStartHook()	2090
7.624.2.6 NvmDoneHook()	2090
7.624.2.7 NvmEraseHook()	2091
7.624.2.8 NvmInitHook()	2091
7.624.2.9 NvmReinitHook()	2091
7.624.2.10 NvmWriteHook()	2091
7.624.2.11 XcpGetSeedHook()	2092
7.624.2.12 XcpVerifyKeyHook()	2092
7.625hooks.c File Reference	2093
7.625.1 Detailed Description	2094

7.625.2 Function Documentation	2094
7.625.2.1 BackDoorEntryHook()	2094
7.625.2.2 BackDoorInitHook()	2094
7.625.2.3 CopInitHook()	2095
7.625.2.4 CopServiceHook()	2095
7.625.2.5 CpuUserProgramStartHook()	2095
7.625.2.6 NvmDoneHook()	2095
7.625.2.7 NvmEraseHook()	2096
7.625.2.8 NvmInitHook()	2096
7.625.2.9 NvmReinitHook()	2096
7.625.2.10 NvmWriteHook()	2096
7.625.2.11 XcpGetSeedHook()	2097
7.625.2.12 XcpVerifyKeyHook()	2097
7.626hooks.c File Reference	2098
7.626.1 Detailed Description	2099
7.626.2 Function Documentation	2099
7.626.2.1 BackDoorEntryHook()	2099
7.626.2.2 BackDoorInitHook()	2099
7.626.2.3 CopInitHook()	2100
7.626.2.4 CopServiceHook()	2100
7.626.2.5 CpuUserProgramStartHook()	2100
7.626.2.6 NvmDoneHook()	2100
7.626.2.7 NvmEraseHook()	2101
7.626.2.8 NvmInitHook()	2101
7.626.2.9 NvmReinitHook()	2101
7.626.2.10 NvmWriteHook()	2101
7.626.2.11 UsbConnectHook()	2102
7.626.2.12 UsbEnterLowPowerModeHook()	2102
7.626.2.13 UsbLeaveLowPowerModeHook()	2102
7.626.2.14 XcpGetSeedHook()	2103

7.626.2.15XcpVerifyKeyHook()	2103
7.627hooks.c File Reference	2103
7.627.1 Detailed Description	2105
7.627.2 Function Documentation	2105
7.627.2.1 BackDoorEntryHook()	2105
7.627.2.2 BackDoorInitHook()	2105
7.627.2.3 CopInitHook()	2105
7.627.2.4 CopServiceHook()	2106
7.627.2.5 CpuUserProgramStartHook()	2106
7.627.2.6 NvmDoneHook()	2106
7.627.2.7 NvmEraseHook()	2106
7.627.2.8 NvmInitHook()	2107
7.627.2.9 NvmReinitHook()	2107
7.627.2.10NvmWriteHook()	2107
7.627.2.11UsbConnectHook()	2108
7.627.2.12UsbEnterLowPowerModeHook()	2108
7.627.2.13UsbLeaveLowPowerModeHook()	2108
7.627.2.14XcpGetSeedHook()	2109
7.627.2.15XcpVerifyKeyHook()	2109
7.628hooks.c File Reference	2109
7.628.1 Detailed Description	2111
7.628.2 Function Documentation	2111
7.628.2.1 BackDoorEntryHook()	2111
7.628.2.2 BackDoorInitHook()	2111
7.628.2.3 CopInitHook()	2111
7.628.2.4 CopServiceHook()	2112
7.628.2.5 CpuUserProgramStartHook()	2112
7.628.2.6 NvmDoneHook()	2112
7.628.2.7 NvmEraseHook()	2112
7.628.2.8 NvmInitHook()	2113

7.628.2.9 NvmReinitHook()	2113
7.628.2.10 NvmWriteHook()	2113
7.628.2.11 UsbConnectHook()	2114
7.628.2.12 UsbEnterLowPowerModeHook()	2114
7.628.2.13 UsbLeaveLowPowerModeHook()	2114
7.628.2.14 XcpGetSeedHook()	2115
7.628.2.15 XcpVerifyKeyHook()	2115
7.629hooks.c File Reference	2115
7.629.1 Detailed Description	2117
7.629.2 Function Documentation	2117
7.629.2.1 BackDoorEntryHook()	2117
7.629.2.2 BackDoorInitHook()	2117
7.629.2.3 CopInitHook()	2117
7.629.2.4 CopServiceHook()	2118
7.629.2.5 CpuUserProgramStartHook()	2118
7.629.2.6 NvmDoneHook()	2118
7.629.2.7 NvmEraseHook()	2118
7.629.2.8 NvmInitHook()	2119
7.629.2.9 NvmReinitHook()	2119
7.629.2.10 NvmWriteHook()	2119
7.629.2.11 UsbConnectHook()	2120
7.629.2.12 UsbEnterLowPowerModeHook()	2120
7.629.2.13 UsbLeaveLowPowerModeHook()	2120
7.629.2.14 XcpGetSeedHook()	2121
7.629.2.15 XcpVerifyKeyHook()	2121
7.630hooks.c File Reference	2121
7.630.1 Detailed Description	2123
7.630.2 Function Documentation	2123
7.630.2.1 BackDoorEntryHook()	2123
7.630.2.2 BackDoorInitHook()	2123

7.630.2.3 CopInitHook()	2123
7.630.2.4 CopServiceHook()	2124
7.630.2.5 CpuUserProgramStartHook()	2124
7.630.2.6 NvmDoneHook()	2124
7.630.2.7 NvmEraseHook()	2124
7.630.2.8 NvmInitHook()	2125
7.630.2.9 NvmReinitHook()	2125
7.630.2.10NvmWriteHook()	2125
7.630.2.11XcpGetSeedHook()	2126
7.630.2.12XcpVerifyKeyHook()	2126
7.631 hooks.c File Reference	2127
7.631.1 Detailed Description	2128
7.631.2 Function Documentation	2128
7.631.2.1 BackDoorEntryHook()	2128
7.631.2.2 BackDoorInitHook()	2128
7.631.2.3 CopInitHook()	2129
7.631.2.4 CopServiceHook()	2129
7.631.2.5 CpuUserProgramStartHook()	2129
7.631.2.6 NvmDoneHook()	2130
7.631.2.7 NvmEraseHook()	2130
7.631.2.8 NvmInitHook()	2130
7.631.2.9 NvmReinitHook()	2131
7.631.2.10NvmWriteHook()	2131
7.631.2.11XcpGetSeedHook()	2131
7.631.2.12XcpVerifyKeyHook()	2132
7.632irq.c File Reference	2132
7.632.1 Detailed Description	2133
7.632.2 Function Documentation	2133
7.632.2.1 IrqInterruptDisable()	2134
7.632.2.2 IrqInterruptEnable()	2134

7.632.2.3 IrqInterruptRestore()	2134
7.633irq.h File Reference	2134
7.633.1 Detailed Description	2135
7.633.2 Function Documentation	2135
7.633.2.1 IrqInterruptDisable()	2135
7.633.2.2 IrqInterruptEnable()	2135
7.633.2.3 IrqInterruptRestore()	2136
7.634led.c File Reference	2136
7.634.1 Detailed Description	2137
7.634.2 Function Documentation	2137
7.634.2.1 LedBlinkExit()	2137
7.634.2.2 LedBlinkInit()	2137
7.634.2.3 LedBlinkTask()	2137
7.635led.c File Reference	2138
7.635.1 Detailed Description	2138
7.635.2 Function Documentation	2139
7.635.2.1 LedInit()	2139
7.635.2.2 LedToggle()	2139
7.636led.c File Reference	2139
7.636.1 Detailed Description	2140
7.636.2 Function Documentation	2140
7.636.2.1 LedBlinkExit()	2140
7.636.2.2 LedBlinkInit()	2140
7.636.2.3 LedBlinkTask()	2141
7.637led.c File Reference	2141
7.637.1 Detailed Description	2142
7.637.2 Function Documentation	2142
7.637.2.1 LedInit()	2142
7.637.2.2 LedToggle()	2142
7.638led.c File Reference	2142

7.638.1 Detailed Description	2143
7.638.2 Function Documentation	2143
7.638.2.1 LedBlinkExit()	2143
7.638.2.2 LedBlinkInit()	2143
7.638.2.3 LedBlinkTask()	2144
7.639led.c File Reference	2144
7.639.1 Detailed Description	2145
7.639.2 Function Documentation	2145
7.639.2.1 LedInit()	2145
7.639.2.2 LedToggle()	2145
7.640led.c File Reference	2145
7.640.1 Detailed Description	2146
7.640.2 Function Documentation	2146
7.640.2.1 LedBlinkExit()	2146
7.640.2.2 LedBlinkInit()	2146
7.640.2.3 LedBlinkTask()	2147
7.641led.c File Reference	2147
7.641.1 Detailed Description	2148
7.641.2 Function Documentation	2148
7.641.2.1 LedInit()	2148
7.641.2.2 LedToggle()	2148
7.642led.c File Reference	2148
7.642.1 Detailed Description	2149
7.642.2 Function Documentation	2149
7.642.2.1 LedBlinkExit()	2149
7.642.2.2 LedBlinkInit()	2149
7.642.2.3 LedBlinkTask()	2150
7.643led.c File Reference	2150
7.643.1 Detailed Description	2151
7.643.2 Function Documentation	2151

7.643.2.1 LedInit()	2151
7.643.2.2 LedToggle()	2151
7.644led.c File Reference	2151
7.644.1 Detailed Description	2152
7.644.2 Function Documentation	2152
7.644.2.1 LedBlinkExit()	2152
7.644.2.2 LedBlinkInit()	2152
7.644.2.3 LedBlinkTask()	2153
7.645led.c File Reference	2153
7.645.1 Detailed Description	2154
7.645.2 Function Documentation	2154
7.645.2.1 LedInit()	2154
7.645.2.2 LedToggle()	2154
7.646led.c File Reference	2154
7.646.1 Detailed Description	2155
7.646.2 Function Documentation	2155
7.646.2.1 LedBlinkExit()	2155
7.646.2.2 LedBlinkInit()	2155
7.646.2.3 LedBlinkTask()	2156
7.647led.c File Reference	2156
7.647.1 Detailed Description	2157
7.647.2 Function Documentation	2157
7.647.2.1 LedInit()	2157
7.647.2.2 LedToggle()	2157
7.648led.c File Reference	2157
7.648.1 Detailed Description	2158
7.648.2 Function Documentation	2158
7.648.2.1 LedBlinkExit()	2158
7.648.2.2 LedBlinkInit()	2158
7.648.2.3 LedBlinkTask()	2159

7.649led.c File Reference	2159
7.649.1 Detailed Description	2160
7.649.2 Function Documentation	2160
7.649.2.1 LedInit()	2160
7.649.2.2 LedToggle()	2160
7.650led.c File Reference	2160
7.650.1 Detailed Description	2161
7.650.2 Function Documentation	2161
7.650.2.1 LedBlinkExit()	2161
7.650.2.2 LedBlinkInit()	2161
7.650.2.3 LedBlinkTask()	2162
7.651led.c File Reference	2162
7.651.1 Detailed Description	2163
7.651.2 Function Documentation	2163
7.651.2.1 LedInit()	2163
7.651.2.2 LedToggle()	2163
7.652led.c File Reference	2163
7.652.1 Detailed Description	2164
7.652.2 Function Documentation	2164
7.652.2.1 LedBlinkExit()	2164
7.652.2.2 LedBlinkInit()	2164
7.652.2.3 LedBlinkTask()	2165
7.653led.c File Reference	2165
7.653.1 Detailed Description	2166
7.653.2 Function Documentation	2166
7.653.2.1 LedInit()	2166
7.653.2.2 LedToggle()	2166
7.654led.c File Reference	2166
7.654.1 Detailed Description	2167
7.654.2 Function Documentation	2167

7.654.2.1 LedBlinkExit()	2167
7.654.2.2 LedBlinkInit()	2167
7.654.2.3 LedBlinkTask()	2168
7.655led.c File Reference	2168
7.655.1 Detailed Description	2169
7.655.2 Function Documentation	2169
7.655.2.1 LedInit()	2169
7.655.2.2 LedToggle()	2169
7.656led.c File Reference	2169
7.656.1 Detailed Description	2170
7.656.2 Function Documentation	2170
7.656.2.1 LedBlinkExit()	2170
7.656.2.2 LedBlinkInit()	2170
7.656.2.3 LedBlinkTask()	2171
7.657led.c File Reference	2171
7.657.1 Detailed Description	2172
7.657.2 Function Documentation	2172
7.657.2.1 LedInit()	2172
7.657.2.2 LedToggle()	2172
7.658led.c File Reference	2172
7.658.1 Detailed Description	2173
7.658.2 Function Documentation	2173
7.658.2.1 LedBlinkExit()	2173
7.658.2.2 LedBlinkInit()	2173
7.658.2.3 LedBlinkTask()	2174
7.659led.c File Reference	2174
7.659.1 Detailed Description	2175
7.659.2 Function Documentation	2175
7.659.2.1 LedInit()	2175
7.659.2.2 LedToggle()	2175

7.660led.c File Reference	2175
7.660.1 Detailed Description	2176
7.660.2 Function Documentation	2176
7.660.2.1 LedBlinkExit()	2176
7.660.2.2 LedBlinkInit()	2176
7.660.2.3 LedBlinkTask()	2177
7.661led.c File Reference	2177
7.661.1 Detailed Description	2178
7.661.2 Function Documentation	2178
7.661.2.1 LedInit()	2178
7.661.2.2 LedToggle()	2178
7.662led.c File Reference	2178
7.662.1 Detailed Description	2179
7.662.2 Function Documentation	2179
7.662.2.1 LedBlinkExit()	2179
7.662.2.2 LedBlinkInit()	2179
7.662.2.3 LedBlinkTask()	2180
7.663led.c File Reference	2180
7.663.1 Detailed Description	2181
7.663.2 Function Documentation	2181
7.663.2.1 LedInit()	2181
7.663.2.2 LedToggle()	2181
7.664led.c File Reference	2181
7.664.1 Detailed Description	2182
7.664.2 Function Documentation	2182
7.664.2.1 LedBlinkExit()	2182
7.664.2.2 LedBlinkInit()	2182
7.664.2.3 LedBlinkTask()	2183
7.665led.c File Reference	2183
7.665.1 Detailed Description	2184

7.665.2 Function Documentation	2184
7.665.2.1 LedInit()	2184
7.665.2.2 LedToggle()	2184
7.666led.c File Reference	2184
7.666.1 Detailed Description	2185
7.666.2 Function Documentation	2185
7.666.2.1 LedBlinkExit()	2185
7.666.2.2 LedBlinkInit()	2185
7.666.2.3 LedBlinkTask()	2186
7.667led.c File Reference	2186
7.667.1 Detailed Description	2187
7.667.2 Function Documentation	2187
7.667.2.1 LedInit()	2187
7.667.2.2 LedToggle()	2187
7.668led.c File Reference	2187
7.668.1 Detailed Description	2188
7.668.2 Function Documentation	2188
7.668.2.1 LedBlinkExit()	2188
7.668.2.2 LedBlinkInit()	2188
7.668.2.3 LedBlinkTask()	2189
7.669led.c File Reference	2189
7.669.1 Detailed Description	2190
7.669.2 Function Documentation	2190
7.669.2.1 LedInit()	2190
7.669.2.2 LedToggle()	2190
7.670led.c File Reference	2190
7.670.1 Detailed Description	2191
7.670.2 Function Documentation	2191
7.670.2.1 LedBlinkExit()	2191
7.670.2.2 LedBlinkInit()	2191

7.670.2.3 LedBlinkTask()	2192
7.671led.c File Reference	2192
7.671.1 Detailed Description	2193
7.671.2 Function Documentation	2193
7.671.2.1 LedInit()	2193
7.671.2.2 LedToggle()	2193
7.672led.c File Reference	2193
7.672.1 Detailed Description	2194
7.672.2 Function Documentation	2194
7.672.2.1 LedInit()	2194
7.672.2.2 LedToggle()	2194
7.673led.c File Reference	2195
7.673.1 Detailed Description	2195
7.673.2 Function Documentation	2195
7.673.2.1 LedInit()	2195
7.673.2.2 LedToggle()	2196
7.674led.c File Reference	2196
7.674.1 Detailed Description	2196
7.674.2 Function Documentation	2196
7.674.2.1 LedInit()	2197
7.674.2.2 LedToggle()	2197
7.675led.c File Reference	2197
7.675.1 Detailed Description	2198
7.675.2 Function Documentation	2198
7.675.2.1 LedInit()	2198
7.675.2.2 LedToggle()	2198
7.676led.c File Reference	2198
7.676.1 Detailed Description	2199
7.676.2 Function Documentation	2199
7.676.2.1 LedInit()	2199

7.676.2.2 LedToggle()	2199
7.677led.c File Reference	2200
7.677.1 Detailed Description	2200
7.677.2 Function Documentation	2200
7.677.2.1 LedInit()	2200
7.677.2.2 LedToggle()	2201
7.678led.c File Reference	2201
7.678.1 Detailed Description	2202
7.678.2 Function Documentation	2202
7.678.2.1 LedBlinkExit()	2202
7.678.2.2 LedBlinkInit()	2202
7.678.2.3 LedBlinkTask()	2202
7.679led.c File Reference	2203
7.679.1 Detailed Description	2203
7.679.2 Function Documentation	2203
7.679.2.1 LedInit()	2203
7.679.2.2 LedToggle()	2204
7.680led.c File Reference	2204
7.680.1 Detailed Description	2205
7.680.2 Function Documentation	2205
7.680.2.1 LedBlinkExit()	2205
7.680.2.2 LedBlinkInit()	2205
7.680.2.3 LedBlinkTask()	2205
7.681led.c File Reference	2206
7.681.1 Detailed Description	2206
7.681.2 Function Documentation	2206
7.681.2.1 LedInit()	2206
7.681.2.2 LedToggle()	2207
7.682led.c File Reference	2207
7.682.1 Detailed Description	2208

7.682.2 Function Documentation	2208
7.682.2.1 LedBlinkExit()	2208
7.682.2.2 LedBlinkInit()	2208
7.682.2.3 LedBlinkTask()	2208
7.683led.c File Reference	2209
7.683.1 Detailed Description	2209
7.683.2 Function Documentation	2209
7.683.2.1 LedInit()	2209
7.683.2.2 LedToggle()	2210
7.684led.c File Reference	2210
7.684.1 Detailed Description	2211
7.684.2 Function Documentation	2211
7.684.2.1 LedBlinkExit()	2211
7.684.2.2 LedBlinkInit()	2211
7.684.2.3 LedBlinkTask()	2211
7.685led.c File Reference	2212
7.685.1 Detailed Description	2212
7.685.2 Function Documentation	2212
7.685.2.1 LedInit()	2212
7.685.2.2 LedToggle()	2213
7.686led.c File Reference	2213
7.686.1 Detailed Description	2214
7.686.2 Function Documentation	2214
7.686.2.1 LedBlinkExit()	2214
7.686.2.2 LedBlinkInit()	2214
7.686.2.3 LedBlinkTask()	2214
7.687led.c File Reference	2215
7.687.1 Detailed Description	2215
7.687.2 Function Documentation	2215
7.687.2.1 LedInit()	2216

7.687.2.2 LedToggle()	2216
7.688led.c File Reference	2216
7.688.1 Detailed Description	2217
7.688.2 Function Documentation	2217
7.688.2.1 LedBlinkExit()	2217
7.688.2.2 LedBlinkInit()	2217
7.688.2.3 LedBlinkTask()	2218
7.689led.c File Reference	2218
7.689.1 Detailed Description	2219
7.689.2 Function Documentation	2219
7.689.2.1 LedInit()	2219
7.689.2.2 LedToggle()	2219
7.690led.c File Reference	2219
7.690.1 Detailed Description	2220
7.690.2 Function Documentation	2220
7.690.2.1 LedBlinkExit()	2220
7.690.2.2 LedBlinkInit()	2220
7.690.2.3 LedBlinkTask()	2221
7.691led.c File Reference	2221
7.691.1 Detailed Description	2222
7.691.2 Function Documentation	2222
7.691.2.1 LedInit()	2222
7.691.2.2 LedToggle()	2222
7.692led.c File Reference	2222
7.692.1 Detailed Description	2223
7.692.2 Function Documentation	2223
7.692.2.1 LedBlinkExit()	2223
7.692.2.2 LedBlinkInit()	2223
7.692.2.3 LedBlinkTask()	2224
7.693led.c File Reference	2224

7.693.1 Detailed Description	2225
7.693.2 Function Documentation	2225
7.693.2.1 LedInit()	2225
7.693.2.2 LedToggle()	2225
7.694led.c File Reference	2225
7.694.1 Detailed Description	2226
7.694.2 Function Documentation	2226
7.694.2.1 LedBlinkExit()	2226
7.694.2.2 LedBlinkInit()	2226
7.694.2.3 LedBlinkTask()	2227
7.695led.c File Reference	2227
7.695.1 Detailed Description	2228
7.695.2 Function Documentation	2228
7.695.2.1 LedInit()	2228
7.695.2.2 LedToggle()	2228
7.696led.c File Reference	2228
7.696.1 Detailed Description	2229
7.696.2 Function Documentation	2229
7.696.2.1 LedBlinkExit()	2229
7.696.2.2 LedBlinkInit()	2229
7.696.2.3 LedBlinkTask()	2230
7.697led.c File Reference	2230
7.697.1 Detailed Description	2231
7.697.2 Function Documentation	2231
7.697.2.1 LedInit()	2231
7.697.2.2 LedToggle()	2231
7.698led.c File Reference	2231
7.698.1 Detailed Description	2232
7.698.2 Function Documentation	2232
7.698.2.1 LedBlinkExit()	2232

7.698.2.2 LedBlinkInit()	2232
7.698.2.3 LedBlinkTask()	2233
7.699led.c File Reference	2233
7.699.1 Detailed Description	2234
7.699.2 Function Documentation	2234
7.699.2.1 LedInit()	2234
7.699.2.2 LedToggle()	2234
7.700led.c File Reference	2234
7.700.1 Detailed Description	2235
7.700.2 Function Documentation	2235
7.700.2.1 LedBlinkExit()	2235
7.700.2.2 LedBlinkInit()	2235
7.700.2.3 LedBlinkTask()	2236
7.701led.c File Reference	2236
7.701.1 Detailed Description	2237
7.701.2 Function Documentation	2237
7.701.2.1 LedInit()	2237
7.701.2.2 LedToggle()	2237
7.702led.c File Reference	2237
7.702.1 Detailed Description	2238
7.702.2 Function Documentation	2238
7.702.2.1 LedBlinkExit()	2238
7.702.2.2 LedBlinkInit()	2238
7.702.2.3 LedBlinkTask()	2239
7.703led.c File Reference	2239
7.703.1 Detailed Description	2240
7.703.2 Function Documentation	2240
7.703.2.1 LedInit()	2240
7.703.2.2 LedToggle()	2240
7.704led.c File Reference	2240

7.704.1 Detailed Description	2241
7.704.2 Function Documentation	2241
7.704.2.1 LedBlinkExit()	2241
7.704.2.2 LedBlinkInit()	2241
7.704.2.3 LedBlinkTask()	2242
7.705led.c File Reference	2242
7.705.1 Detailed Description	2243
7.705.2 Function Documentation	2243
7.705.2.1 LedInit()	2243
7.705.2.2 LedToggle()	2243
7.706led.c File Reference	2243
7.706.1 Detailed Description	2244
7.706.2 Function Documentation	2244
7.706.2.1 LedBlinkExit()	2244
7.706.2.2 LedBlinkInit()	2244
7.706.2.3 LedBlinkTask()	2245
7.707led.c File Reference	2245
7.707.1 Detailed Description	2246
7.707.2 Function Documentation	2246
7.707.2.1 LedInit()	2246
7.707.2.2 LedToggle()	2246
7.708led.c File Reference	2246
7.708.1 Detailed Description	2247
7.708.2 Function Documentation	2247
7.708.2.1 LedBlinkExit()	2247
7.708.2.2 LedBlinkInit()	2247
7.708.2.3 LedBlinkTask()	2248
7.709led.c File Reference	2248
7.709.1 Detailed Description	2249
7.709.2 Function Documentation	2249

7.709.2.1 LedInit()	2249
7.709.2.2 LedToggle()	2249
7.710led.c File Reference	2249
7.710.1 Detailed Description	2250
7.710.2 Function Documentation	2250
7.710.2.1 LedBlinkExit()	2250
7.710.2.2 LedBlinkInit()	2250
7.710.2.3 LedBlinkTask()	2251
7.711led.c File Reference	2251
7.711.1 Detailed Description	2252
7.711.2 Function Documentation	2252
7.711.2.1 LedInit()	2252
7.711.2.2 LedToggle()	2252
7.712led.c File Reference	2252
7.712.1 Detailed Description	2253
7.712.2 Function Documentation	2253
7.712.2.1 LedBlinkExit()	2253
7.712.2.2 LedBlinkInit()	2253
7.712.2.3 LedBlinkTask()	2254
7.713led.c File Reference	2254
7.713.1 Detailed Description	2255
7.713.2 Function Documentation	2255
7.713.2.1 LedInit()	2255
7.713.2.2 LedToggle()	2255
7.714led.c File Reference	2255
7.714.1 Detailed Description	2256
7.714.2 Function Documentation	2256
7.714.2.1 LedBlinkExit()	2256
7.714.2.2 LedBlinkInit()	2256
7.714.2.3 LedBlinkTask()	2257

7.715led.c File Reference	2257
7.715.1 Detailed Description	2258
7.715.2 Function Documentation	2258
7.715.2.1 LedInit()	2258
7.715.2.2 LedToggle()	2258
7.716led.c File Reference	2258
7.716.1 Detailed Description	2259
7.716.2 Function Documentation	2259
7.716.2.1 LedBlinkExit()	2259
7.716.2.2 LedBlinkInit()	2259
7.716.2.3 LedBlinkTask()	2260
7.717led.c File Reference	2260
7.717.1 Detailed Description	2261
7.717.2 Function Documentation	2261
7.717.2.1 LedInit()	2261
7.717.2.2 LedToggle()	2261
7.718led.c File Reference	2261
7.718.1 Detailed Description	2262
7.718.2 Function Documentation	2262
7.718.2.1 LedBlinkExit()	2262
7.718.2.2 LedBlinkInit()	2262
7.718.2.3 LedBlinkTask()	2263
7.719led.c File Reference	2263
7.719.1 Detailed Description	2264
7.719.2 Function Documentation	2264
7.719.2.1 LedInit()	2264
7.719.2.2 LedToggle()	2264
7.720led.c File Reference	2264
7.720.1 Detailed Description	2265
7.720.2 Function Documentation	2265

7.720.2.1 LedBlinkExit()	2265
7.720.2.2 LedBlinkInit()	2265
7.720.2.3 LedBlinkTask()	2266
7.721led.c File Reference	2266
7.721.1 Detailed Description	2267
7.721.2 Function Documentation	2267
7.721.2.1 LedInit()	2267
7.721.2.2 LedToggle()	2267
7.722led.c File Reference	2267
7.722.1 Detailed Description	2268
7.722.2 Function Documentation	2268
7.722.2.1 LedBlinkExit()	2268
7.722.2.2 LedBlinkInit()	2268
7.722.2.3 LedBlinkTask()	2269
7.723led.c File Reference	2269
7.723.1 Detailed Description	2270
7.723.2 Function Documentation	2270
7.723.2.1 LedInit()	2270
7.723.2.2 LedToggle()	2270
7.724led.c File Reference	2271
7.724.1 Detailed Description	2271
7.724.2 Function Documentation	2271
7.724.2.1 LedBlinkExit()	2272
7.724.2.2 LedBlinkInit()	2272
7.724.2.3 LedBlinkTask()	2272
7.725led.c File Reference	2272
7.725.1 Detailed Description	2273
7.725.2 Function Documentation	2273
7.725.2.1 LedInit()	2273
7.725.2.2 LedToggle()	2274

7.726led.c File Reference	2274
7.726.1 Detailed Description	2275
7.726.2 Function Documentation	2275
7.726.2.1 LedBlinkExit()	2275
7.726.2.2 LedBlinkInit()	2275
7.726.2.3 LedBlinkTask()	2275
7.727led.c File Reference	2276
7.727.1 Detailed Description	2276
7.727.2 Function Documentation	2276
7.727.2.1 LedInit()	2277
7.727.2.2 LedToggle()	2277
7.728led.c File Reference	2277
7.728.1 Detailed Description	2278
7.728.2 Function Documentation	2278
7.728.2.1 LedBlinkExit()	2278
7.728.2.2 LedBlinkInit()	2278
7.728.2.3 LedBlinkTask()	2279
7.729led.c File Reference	2279
7.729.1 Detailed Description	2280
7.729.2 Function Documentation	2280
7.729.2.1 LedInit()	2280
7.729.2.2 LedToggle()	2280
7.730led.c File Reference	2281
7.730.1 Detailed Description	2281
7.730.2 Function Documentation	2281
7.730.2.1 LedBlinkExit()	2282
7.730.2.2 LedBlinkInit()	2282
7.730.2.3 LedBlinkTask()	2282
7.731led.c File Reference	2282
7.731.1 Detailed Description	2283

7.731.2 Function Documentation	2283
7.731.2.1 LedInit()	2283
7.731.2.2 LedToggle()	2284
7.732led.c File Reference	2284
7.732.1 Detailed Description	2285
7.732.2 Function Documentation	2285
7.732.2.1 LedBlinkExit()	2285
7.732.2.2 LedBlinkInit()	2285
7.732.2.3 LedBlinkTask()	2285
7.733led.c File Reference	2286
7.733.1 Detailed Description	2286
7.733.2 Function Documentation	2286
7.733.2.1 LedInit()	2286
7.733.2.2 LedToggle()	2287
7.734led.c File Reference	2287
7.734.1 Detailed Description	2288
7.734.2 Function Documentation	2288
7.734.2.1 LedBlinkExit()	2288
7.734.2.2 LedBlinkInit()	2288
7.734.2.3 LedBlinkTask()	2288
7.735led.c File Reference	2289
7.735.1 Detailed Description	2289
7.735.2 Function Documentation	2289
7.735.2.1 LedInit()	2289
7.735.2.2 LedToggle()	2290
7.736led.c File Reference	2290
7.736.1 Detailed Description	2291
7.736.2 Function Documentation	2291
7.736.2.1 LedBlinkExit()	2291
7.736.2.2 LedBlinkInit()	2291

7.736.2.3 LedBlinkTask()	2291
7.737led.c File Reference	2292
7.737.1 Detailed Description	2292
7.737.2 Function Documentation	2292
7.737.2.1 LedInit()	2292
7.737.2.2 LedToggle()	2293
7.738led.c File Reference	2293
7.738.1 Detailed Description	2294
7.738.2 Function Documentation	2294
7.738.2.1 LedBlinkExit()	2294
7.738.2.2 LedBlinkInit()	2294
7.738.2.3 LedBlinkTask()	2294
7.739led.c File Reference	2295
7.739.1 Detailed Description	2295
7.739.2 Function Documentation	2295
7.739.2.1 LedInit()	2295
7.739.2.2 LedToggle()	2296
7.740led.c File Reference	2296
7.740.1 Detailed Description	2296
7.740.2 Function Documentation	2297
7.740.2.1 LedInit()	2297
7.740.2.2 LedToggle()	2297
7.741led.c File Reference	2297
7.741.1 Detailed Description	2298
7.741.2 Function Documentation	2298
7.741.2.1 LedInit()	2298
7.741.2.2 LedToggle()	2298
7.742led.c File Reference	2299
7.742.1 Detailed Description	2299
7.742.2 Function Documentation	2299

7.742.2.1 LedInit()	2299
7.742.2.2 LedToggle()	2300
7.743led.c File Reference	2300
7.743.1 Detailed Description	2301
7.743.2 Function Documentation	2301
7.743.2.1 LedBlinkExit()	2301
7.743.2.2 LedBlinkInit()	2301
7.743.2.3 LedBlinkTask()	2301
7.744led.c File Reference	2302
7.744.1 Detailed Description	2302
7.744.2 Function Documentation	2302
7.744.2.1 LedInit()	2302
7.744.2.2 LedToggle()	2303
7.745led.c File Reference	2303
7.745.1 Detailed Description	2304
7.745.2 Function Documentation	2304
7.745.2.1 LedBlinkExit()	2304
7.745.2.2 LedBlinkInit()	2304
7.745.2.3 LedBlinkTask()	2304
7.746led.c File Reference	2305
7.746.1 Detailed Description	2305
7.746.2 Function Documentation	2305
7.746.2.1 LedInit()	2305
7.746.2.2 LedToggle()	2306
7.747led.c File Reference	2306
7.747.1 Detailed Description	2307
7.747.2 Function Documentation	2307
7.747.2.1 LedBlinkExit()	2307
7.747.2.2 LedBlinkInit()	2307
7.747.2.3 LedBlinkTask()	2307

7.748led.c File Reference	2308
7.748.1 Detailed Description	2308
7.748.2 Function Documentation	2308
7.748.2.1 LedInit()	2308
7.748.2.2 LedToggle()	2309
7.749led.c File Reference	2309
7.749.1 Detailed Description	2310
7.749.2 Function Documentation	2310
7.749.2.1 LedBlinkExit()	2310
7.749.2.2 LedBlinkInit()	2310
7.749.2.3 LedBlinkTask()	2310
7.750led.c File Reference	2311
7.750.1 Detailed Description	2311
7.750.2 Function Documentation	2311
7.750.2.1 LedInit()	2311
7.750.2.2 LedToggle()	2312
7.751led.c File Reference	2312
7.751.1 Detailed Description	2313
7.751.2 Function Documentation	2313
7.751.2.1 LedBlinkExit()	2313
7.751.2.2 LedBlinkInit()	2313
7.751.2.3 LedBlinkTask()	2313
7.752led.c File Reference	2314
7.752.1 Detailed Description	2314
7.752.2 Function Documentation	2314
7.752.2.1 LedInit()	2314
7.752.2.2 LedToggle()	2315
7.753led.c File Reference	2315
7.753.1 Detailed Description	2316
7.753.2 Function Documentation	2316

7.753.2.1 LedBlinkExit()	2316
7.753.2.2 LedBlinkInit()	2316
7.753.2.3 LedBlinkTask()	2316
7.754led.c File Reference	2317
7.754.1 Detailed Description	2317
7.754.2 Function Documentation	2317
7.754.2.1 LedInit()	2317
7.754.2.2 LedToggle()	2318
7.755led.c File Reference	2318
7.755.1 Detailed Description	2319
7.755.2 Function Documentation	2319
7.755.2.1 LedBlinkExit()	2319
7.755.2.2 LedBlinkInit()	2319
7.755.2.3 LedBlinkTask()	2319
7.756led.c File Reference	2320
7.756.1 Detailed Description	2320
7.756.2 Function Documentation	2320
7.756.2.1 LedInit()	2320
7.756.2.2 LedToggle()	2321
7.757led.c File Reference	2321
7.757.1 Detailed Description	2322
7.757.2 Function Documentation	2322
7.757.2.1 LedBlinkExit()	2322
7.757.2.2 LedBlinkInit()	2322
7.757.2.3 LedBlinkTask()	2322
7.758led.c File Reference	2323
7.758.1 Detailed Description	2323
7.758.2 Function Documentation	2323
7.758.2.1 LedInit()	2323
7.758.2.2 LedToggle()	2324

7.759led.c File Reference	2324
7.759.1 Detailed Description	2324
7.759.2 Function Documentation	2324
7.759.2.1 LedInit()	2325
7.759.2.2 LedToggle()	2325
7.760led.c File Reference	2325
7.760.1 Detailed Description	2326
7.760.2 Function Documentation	2326
7.760.2.1 LedBlinkExit()	2326
7.760.2.2 LedBlinkInit()	2326
7.760.2.3 LedBlinkTask()	2327
7.761led.c File Reference	2327
7.761.1 Detailed Description	2328
7.761.2 Function Documentation	2328
7.761.2.1 LedInit()	2328
7.761.2.2 LedToggle()	2328
7.762led.c File Reference	2328
7.762.1 Detailed Description	2329
7.762.2 Function Documentation	2329
7.762.2.1 LedBlinkExit()	2329
7.762.2.2 LedBlinkInit()	2329
7.762.2.3 LedBlinkTask()	2330
7.763led.c File Reference	2330
7.763.1 Detailed Description	2331
7.763.2 Function Documentation	2331
7.763.2.1 LedInit()	2331
7.763.2.2 LedToggle()	2331
7.764led.c File Reference	2331
7.764.1 Detailed Description	2332
7.764.2 Function Documentation	2332

7.764.2.1 LedBlinkExit()	2332
7.764.2.2 LedBlinkInit()	2332
7.764.2.3 LedBlinkTask()	2333
7.765led.c File Reference	2333
7.765.1 Detailed Description	2334
7.765.2 Function Documentation	2334
7.765.2.1 LedInit()	2334
7.765.2.2 LedToggle()	2334
7.766led.c File Reference	2334
7.766.1 Detailed Description	2335
7.766.2 Function Documentation	2335
7.766.2.1 LedBlinkExit()	2335
7.766.2.2 LedBlinkInit()	2335
7.766.2.3 LedBlinkTask()	2336
7.767led.c File Reference	2336
7.767.1 Detailed Description	2337
7.767.2 Function Documentation	2337
7.767.2.1 LedInit()	2337
7.767.2.2 LedToggle()	2337
7.768led.c File Reference	2337
7.768.1 Detailed Description	2338
7.768.2 Function Documentation	2338
7.768.2.1 LedBlinkExit()	2338
7.768.2.2 LedBlinkInit()	2338
7.768.2.3 LedBlinkTask()	2339
7.769led.c File Reference	2339
7.769.1 Detailed Description	2340
7.769.2 Function Documentation	2340
7.769.2.1 LedInit()	2340
7.769.2.2 LedToggle()	2340

7.770led.c File Reference	2340
7.770.1 Detailed Description	2341
7.770.2 Function Documentation	2341
7.770.2.1 LedBlinkExit()	2341
7.770.2.2 LedBlinkInit()	2341
7.770.2.3 LedBlinkTask()	2342
7.771led.c File Reference	2342
7.771.1 Detailed Description	2343
7.771.2 Function Documentation	2343
7.771.2.1 LedInit()	2343
7.771.2.2 LedToggle()	2343
7.772led.c File Reference	2343
7.772.1 Detailed Description	2344
7.772.2 Function Documentation	2344
7.772.2.1 LedBlinkExit()	2344
7.772.2.2 LedBlinkInit()	2344
7.772.2.3 LedBlinkTask()	2345
7.773led.c File Reference	2345
7.773.1 Detailed Description	2346
7.773.2 Function Documentation	2346
7.773.2.1 LedInit()	2346
7.773.2.2 LedToggle()	2346
7.774led.c File Reference	2346
7.774.1 Detailed Description	2347
7.774.2 Function Documentation	2347
7.774.2.1 LedBlinkExit()	2347
7.774.2.2 LedBlinkInit()	2347
7.774.2.3 LedBlinkTask()	2348
7.775led.c File Reference	2348
7.775.1 Detailed Description	2349

7.775.2 Function Documentation	2349
7.775.2.1 LedInit()	2349
7.775.2.2 LedToggle()	2349
7.776led.c File Reference	2349
7.776.1 Detailed Description	2350
7.776.2 Function Documentation	2350
7.776.2.1 LedBlinkExit()	2350
7.776.2.2 LedBlinkInit()	2350
7.776.2.3 LedBlinkTask()	2351
7.777led.c File Reference	2351
7.777.1 Detailed Description	2352
7.777.2 Function Documentation	2352
7.777.2.1 LedInit()	2352
7.777.2.2 LedToggle()	2352
7.778led.c File Reference	2352
7.778.1 Detailed Description	2353
7.778.2 Function Documentation	2353
7.778.2.1 LedBlinkExit()	2353
7.778.2.2 LedBlinkInit()	2353
7.778.2.3 LedBlinkTask()	2354
7.779led.c File Reference	2354
7.779.1 Detailed Description	2355
7.779.2 Function Documentation	2355
7.779.2.1 LedInit()	2355
7.779.2.2 LedToggle()	2355
7.780led.c File Reference	2355
7.780.1 Detailed Description	2356
7.780.2 Function Documentation	2356
7.780.2.1 LedBlinkExit()	2356
7.780.2.2 LedBlinkInit()	2356

7.780.2.3 LedBlinkTask()	2357
7.781led.c File Reference	2357
7.781.1 Detailed Description	2358
7.781.2 Function Documentation	2358
7.781.2.1 LedInit()	2358
7.781.2.2 LedToggle()	2358
7.782led.c File Reference	2358
7.782.1 Detailed Description	2359
7.782.2 Function Documentation	2359
7.782.2.1 LedBlinkExit()	2359
7.782.2.2 LedBlinkInit()	2359
7.782.2.3 LedBlinkTask()	2360
7.783led.c File Reference	2360
7.783.1 Detailed Description	2361
7.783.2 Function Documentation	2361
7.783.2.1 LedInit()	2361
7.783.2.2 LedToggle()	2361
7.784led.c File Reference	2361
7.784.1 Detailed Description	2362
7.784.2 Function Documentation	2362
7.784.2.1 LedBlinkExit()	2362
7.784.2.2 LedBlinkInit()	2362
7.784.2.3 LedBlinkTask()	2363
7.785led.c File Reference	2363
7.785.1 Detailed Description	2364
7.785.2 Function Documentation	2364
7.785.2.1 LedInit()	2364
7.785.2.2 LedToggle()	2364
7.786led.c File Reference	2364
7.786.1 Detailed Description	2365

7.786.2 Function Documentation	2365
7.786.2.1 LedBlinkExit()	2365
7.786.2.2 LedBlinkInit()	2365
7.786.2.3 LedBlinkTask()	2366
7.787led.c File Reference	2366
7.787.1 Detailed Description	2367
7.787.2 Function Documentation	2367
7.787.2.1 LedInit()	2367
7.787.2.2 LedToggle()	2367
7.788led.c File Reference	2367
7.788.1 Detailed Description	2368
7.788.2 Function Documentation	2368
7.788.2.1 LedBlinkExit()	2368
7.788.2.2 LedBlinkInit()	2368
7.788.2.3 LedBlinkTask()	2369
7.789led.c File Reference	2369
7.789.1 Detailed Description	2370
7.789.2 Function Documentation	2370
7.789.2.1 LedInit()	2370
7.789.2.2 LedToggle()	2370
7.790led.c File Reference	2370
7.790.1 Detailed Description	2371
7.790.2 Function Documentation	2371
7.790.2.1 LedInit()	2371
7.790.2.2 LedToggle()	2371
7.791led.h File Reference	2372
7.791.1 Detailed Description	2372
7.791.2 Function Documentation	2372
7.791.2.1 LedBlinkExit()	2372
7.791.2.2 LedBlinkInit()	2372

7.791.2.3 LedBlinkTask()	2373
7.792led.h File Reference	2373
7.792.1 Detailed Description	2374
7.792.2 Function Documentation	2374
7.792.2.1 LedInit()	2374
7.792.2.2 LedToggle()	2374
7.793led.h File Reference	2374
7.793.1 Detailed Description	2375
7.793.2 Function Documentation	2375
7.793.2.1 LedBlinkExit()	2375
7.793.2.2 LedBlinkInit()	2375
7.793.2.3 LedBlinkTask()	2376
7.794led.h File Reference	2376
7.794.1 Detailed Description	2376
7.794.2 Function Documentation	2376
7.794.2.1 LedInit()	2377
7.794.2.2 LedToggle()	2377
7.795led.h File Reference	2377
7.795.1 Detailed Description	2378
7.795.2 Function Documentation	2378
7.795.2.1 LedBlinkExit()	2378
7.795.2.2 LedBlinkInit()	2378
7.795.2.3 LedBlinkTask()	2379
7.796led.h File Reference	2379
7.796.1 Detailed Description	2379
7.796.2 Function Documentation	2379
7.796.2.1 LedInit()	2380
7.796.2.2 LedToggle()	2380
7.797led.h File Reference	2380
7.797.1 Detailed Description	2381

7.797.2 Function Documentation	2381
7.797.2.1 LedBlinkExit()	2381
7.797.2.2 LedBlinkInit()	2381
7.797.2.3 LedBlinkTask()	2382
7.798led.h File Reference	2382
7.798.1 Detailed Description	2382
7.798.2 Function Documentation	2382
7.798.2.1 LedInit()	2383
7.798.2.2 LedToggle()	2383
7.799led.h File Reference	2383
7.799.1 Detailed Description	2384
7.799.2 Function Documentation	2384
7.799.2.1 LedBlinkExit()	2384
7.799.2.2 LedBlinkInit()	2384
7.799.2.3 LedBlinkTask()	2385
7.800led.h File Reference	2385
7.800.1 Detailed Description	2385
7.800.2 Function Documentation	2385
7.800.2.1 LedInit()	2386
7.800.2.2 LedToggle()	2386
7.801led.h File Reference	2386
7.801.1 Detailed Description	2387
7.801.2 Function Documentation	2387
7.801.2.1 LedBlinkExit()	2387
7.801.2.2 LedBlinkInit()	2387
7.801.2.3 LedBlinkTask()	2388
7.802led.h File Reference	2388
7.802.1 Detailed Description	2388
7.802.2 Function Documentation	2388
7.802.2.1 LedInit()	2389

7.802.2.2 LedToggle()	2389
7.803led.h File Reference	2389
7.803.1 Detailed Description	2390
7.803.2 Function Documentation	2390
7.803.2.1 LedBlinkExit()	2390
7.803.2.2 LedBlinkInit()	2390
7.803.2.3 LedBlinkTask()	2391
7.804led.h File Reference	2391
7.804.1 Detailed Description	2391
7.804.2 Function Documentation	2391
7.804.2.1 LedInit()	2392
7.804.2.2 LedToggle()	2392
7.805led.h File Reference	2392
7.805.1 Detailed Description	2393
7.805.2 Function Documentation	2393
7.805.2.1 LedBlinkExit()	2393
7.805.2.2 LedBlinkInit()	2393
7.805.2.3 LedBlinkTask()	2394
7.806led.h File Reference	2394
7.806.1 Detailed Description	2394
7.806.2 Function Documentation	2394
7.806.2.1 LedInit()	2395
7.806.2.2 LedToggle()	2395
7.807led.h File Reference	2395
7.807.1 Detailed Description	2396
7.807.2 Function Documentation	2396
7.807.2.1 LedBlinkExit()	2396
7.807.2.2 LedBlinkInit()	2396
7.807.2.3 LedBlinkTask()	2397
7.808led.h File Reference	2397

7.808.1 Detailed Description	2397
7.808.2 Function Documentation	2397
7.808.2.1 LedInit()	2398
7.808.2.2 LedToggle()	2398
7.809led.h File Reference	2398
7.809.1 Detailed Description	2399
7.809.2 Function Documentation	2399
7.809.2.1 LedBlinkExit()	2399
7.809.2.2 LedBlinkInit()	2399
7.809.2.3 LedBlinkTask()	2400
7.810led.h File Reference	2400
7.810.1 Detailed Description	2400
7.810.2 Function Documentation	2400
7.810.2.1 LedInit()	2401
7.810.2.2 LedToggle()	2401
7.811led.h File Reference	2401
7.811.1 Detailed Description	2402
7.811.2 Function Documentation	2402
7.811.2.1 LedBlinkExit()	2402
7.811.2.2 LedBlinkInit()	2402
7.811.2.3 LedBlinkTask()	2403
7.812led.h File Reference	2403
7.812.1 Detailed Description	2403
7.812.2 Function Documentation	2403
7.812.2.1 LedInit()	2404
7.812.2.2 LedToggle()	2404
7.813led.h File Reference	2404
7.813.1 Detailed Description	2405
7.813.2 Function Documentation	2405
7.813.2.1 LedBlinkExit()	2405

7.813.2.2 LedBlinkInit()	2405
7.813.2.3 LedBlinkTask()	2406
7.814led.h File Reference	2406
7.814.1 Detailed Description	2406
7.814.2 Function Documentation	2406
7.814.2.1 LedInit()	2407
7.814.2.2 LedToggle()	2407
7.815led.h File Reference	2407
7.815.1 Detailed Description	2408
7.815.2 Function Documentation	2408
7.815.2.1 LedBlinkExit()	2408
7.815.2.2 LedBlinkInit()	2408
7.815.2.3 LedBlinkTask()	2409
7.816led.h File Reference	2409
7.816.1 Detailed Description	2409
7.816.2 Function Documentation	2409
7.816.2.1 LedInit()	2410
7.816.2.2 LedToggle()	2410
7.817led.h File Reference	2410
7.817.1 Detailed Description	2411
7.817.2 Function Documentation	2411
7.817.2.1 LedBlinkExit()	2411
7.817.2.2 LedBlinkInit()	2411
7.817.2.3 LedBlinkTask()	2412
7.818led.h File Reference	2412
7.818.1 Detailed Description	2412
7.818.2 Function Documentation	2412
7.818.2.1 LedInit()	2413
7.818.2.2 LedToggle()	2413
7.819led.h File Reference	2413

7.819.1 Detailed Description	2414
7.819.2 Function Documentation	2414
7.819.2.1 LedBlinkExit()	2414
7.819.2.2 LedBlinkInit()	2414
7.819.2.3 LedBlinkTask()	2415
7.820led.h File Reference	2415
7.820.1 Detailed Description	2415
7.820.2 Function Documentation	2415
7.820.2.1 LedInit()	2416
7.820.2.2 LedToggle()	2416
7.821led.h File Reference	2416
7.821.1 Detailed Description	2417
7.821.2 Function Documentation	2417
7.821.2.1 LedBlinkExit()	2417
7.821.2.2 LedBlinkInit()	2417
7.821.2.3 LedBlinkTask()	2418
7.822led.h File Reference	2418
7.822.1 Detailed Description	2418
7.822.2 Function Documentation	2418
7.822.2.1 LedInit()	2419
7.822.2.2 LedToggle()	2419
7.823led.h File Reference	2419
7.823.1 Detailed Description	2420
7.823.2 Function Documentation	2420
7.823.2.1 LedBlinkExit()	2420
7.823.2.2 LedBlinkInit()	2420
7.823.2.3 LedBlinkTask()	2421
7.824led.h File Reference	2421
7.824.1 Detailed Description	2421
7.824.2 Function Documentation	2421

7.824.2.1 LedInit()	2422
7.824.2.2 LedToggle()	2422
7.825led.h File Reference	2422
7.825.1 Detailed Description	2423
7.825.2 Function Documentation	2423
7.825.2.1 LedBlinkExit()	2423
7.825.2.2 LedBlinkInit()	2423
7.825.2.3 LedBlinkTask()	2424
7.826led.h File Reference	2424
7.826.1 Detailed Description	2424
7.826.2 Function Documentation	2424
7.826.2.1 LedInit()	2425
7.826.2.2 LedToggle()	2425
7.827led.h File Reference	2425
7.827.1 Detailed Description	2426
7.827.2 Function Documentation	2426
7.827.2.1 LedBlinkExit()	2426
7.827.2.2 LedBlinkInit()	2426
7.827.2.3 LedBlinkTask()	2427
7.828led.h File Reference	2427
7.828.1 Detailed Description	2427
7.828.2 Function Documentation	2427
7.828.2.1 LedInit()	2428
7.828.2.2 LedToggle()	2428
7.829led.h File Reference	2428
7.829.1 Detailed Description	2429
7.829.2 Function Documentation	2429
7.829.2.1 LedInit()	2429
7.829.2.2 LedToggle()	2430
7.830led.h File Reference	2430

7.830.1 Detailed Description	2431
7.830.2 Function Documentation	2431
7.830.2.1 LedInit()	2431
7.830.2.2 LedToggle()	2431
7.831led.h File Reference	2432
7.831.1 Detailed Description	2432
7.831.2 Function Documentation	2432
7.831.2.1 LedInit()	2432
7.831.2.2 LedToggle()	2433
7.832led.h File Reference	2433
7.832.1 Detailed Description	2433
7.832.2 Function Documentation	2433
7.832.2.1 LedInit()	2434
7.832.2.2 LedToggle()	2434
7.833led.h File Reference	2434
7.833.1 Detailed Description	2435
7.833.2 Function Documentation	2435
7.833.2.1 LedInit()	2435
7.833.2.2 LedToggle()	2435
7.834led.h File Reference	2436
7.834.1 Detailed Description	2436
7.834.2 Function Documentation	2436
7.834.2.1 LedInit()	2436
7.834.2.2 LedToggle()	2437
7.835led.h File Reference	2437
7.835.1 Detailed Description	2437
7.835.2 Function Documentation	2437
7.835.2.1 LedBlinkExit()	2438
7.835.2.2 LedBlinkInit()	2438
7.835.2.3 LedBlinkTask()	2438

7.836led.h File Reference	2438
7.836.1 Detailed Description	2439
7.836.2 Function Documentation	2439
7.836.2.1 LedInit()	2439
7.836.2.2 LedToggle()	2440
7.837led.h File Reference	2440
7.837.1 Detailed Description	2440
7.837.2 Function Documentation	2440
7.837.2.1 LedBlinkExit()	2441
7.837.2.2 LedBlinkInit()	2441
7.837.2.3 LedBlinkTask()	2441
7.838led.h File Reference	2441
7.838.1 Detailed Description	2442
7.838.2 Function Documentation	2442
7.838.2.1 LedInit()	2442
7.838.2.2 LedToggle()	2443
7.839led.h File Reference	2443
7.839.1 Detailed Description	2443
7.839.2 Function Documentation	2443
7.839.2.1 LedBlinkExit()	2444
7.839.2.2 LedBlinkInit()	2444
7.839.2.3 LedBlinkTask()	2444
7.840led.h File Reference	2444
7.840.1 Detailed Description	2445
7.840.2 Function Documentation	2445
7.840.2.1 LedInit()	2445
7.840.2.2 LedToggle()	2446
7.841led.h File Reference	2446
7.841.1 Detailed Description	2446
7.841.2 Function Documentation	2446

7.841.2.1 LedBlinkExit()	2447
7.841.2.2 LedBlinkInit()	2447
7.841.2.3 LedBlinkTask()	2447
7.842led.h File Reference	2447
7.842.1 Detailed Description	2448
7.842.2 Function Documentation	2448
7.842.2.1 LedInit()	2448
7.842.2.2 LedToggle()	2449
7.843led.h File Reference	2449
7.843.1 Detailed Description	2449
7.843.2 Function Documentation	2449
7.843.2.1 LedBlinkExit()	2450
7.843.2.2 LedBlinkInit()	2450
7.843.2.3 LedBlinkTask()	2450
7.844led.h File Reference	2450
7.844.1 Detailed Description	2451
7.844.2 Function Documentation	2451
7.844.2.1 LedInit()	2451
7.844.2.2 LedToggle()	2452
7.845led.h File Reference	2452
7.845.1 Detailed Description	2452
7.845.2 Function Documentation	2452
7.845.2.1 LedBlinkExit()	2453
7.845.2.2 LedBlinkInit()	2453
7.845.2.3 LedBlinkTask()	2453
7.846led.h File Reference	2453
7.846.1 Detailed Description	2454
7.846.2 Function Documentation	2454
7.846.2.1 LedInit()	2454
7.846.2.2 LedToggle()	2455

7.847led.h File Reference	2455
7.847.1 Detailed Description	2455
7.847.2 Function Documentation	2455
7.847.2.1 LedBlinkExit()	2456
7.847.2.2 LedBlinkInit()	2456
7.847.2.3 LedBlinkTask()	2456
7.848led.h File Reference	2456
7.848.1 Detailed Description	2457
7.848.2 Function Documentation	2457
7.848.2.1 LedInit()	2457
7.848.2.2 LedToggle()	2458
7.849led.h File Reference	2458
7.849.1 Detailed Description	2458
7.849.2 Function Documentation	2458
7.849.2.1 LedBlinkExit()	2459
7.849.2.2 LedBlinkInit()	2459
7.849.2.3 LedBlinkTask()	2459
7.850led.h File Reference	2459
7.850.1 Detailed Description	2460
7.850.2 Function Documentation	2460
7.850.2.1 LedInit()	2460
7.850.2.2 LedToggle()	2461
7.851led.h File Reference	2461
7.851.1 Detailed Description	2461
7.851.2 Function Documentation	2461
7.851.2.1 LedBlinkExit()	2462
7.851.2.2 LedBlinkInit()	2462
7.851.2.3 LedBlinkTask()	2462
7.852led.h File Reference	2462
7.852.1 Detailed Description	2463

7.852.2 Function Documentation	2463
7.852.2.1 LedInit()	2463
7.852.2.2 LedToggle()	2464
7.853led.h File Reference	2464
7.853.1 Detailed Description	2464
7.853.2 Function Documentation	2464
7.853.2.1 LedBlinkExit()	2465
7.853.2.2 LedBlinkInit()	2465
7.853.2.3 LedBlinkTask()	2465
7.854led.h File Reference	2465
7.854.1 Detailed Description	2466
7.854.2 Function Documentation	2466
7.854.2.1 LedInit()	2466
7.854.2.2 LedToggle()	2467
7.855led.h File Reference	2467
7.855.1 Detailed Description	2467
7.855.2 Function Documentation	2467
7.855.2.1 LedBlinkExit()	2468
7.855.2.2 LedBlinkInit()	2468
7.855.2.3 LedBlinkTask()	2468
7.856led.h File Reference	2468
7.856.1 Detailed Description	2469
7.856.2 Function Documentation	2469
7.856.2.1 LedInit()	2469
7.856.2.2 LedToggle()	2470
7.857led.h File Reference	2470
7.857.1 Detailed Description	2470
7.857.2 Function Documentation	2470
7.857.2.1 LedBlinkExit()	2471
7.857.2.2 LedBlinkInit()	2471

7.857.2.3 LedBlinkTask()	2471
7.858led.h File Reference	2471
7.858.1 Detailed Description	2472
7.858.2 Function Documentation	2472
7.858.2.1 LedInit()	2472
7.858.2.2 LedToggle()	2473
7.859led.h File Reference	2473
7.859.1 Detailed Description	2473
7.859.2 Function Documentation	2473
7.859.2.1 LedBlinkExit()	2474
7.859.2.2 LedBlinkInit()	2474
7.859.2.3 LedBlinkTask()	2474
7.860led.h File Reference	2474
7.860.1 Detailed Description	2475
7.860.2 Function Documentation	2475
7.860.2.1 LedInit()	2475
7.860.2.2 LedToggle()	2476
7.861led.h File Reference	2476
7.861.1 Detailed Description	2476
7.861.2 Function Documentation	2476
7.861.2.1 LedBlinkExit()	2477
7.861.2.2 LedBlinkInit()	2477
7.861.2.3 LedBlinkTask()	2477
7.862led.h File Reference	2477
7.862.1 Detailed Description	2478
7.862.2 Function Documentation	2478
7.862.2.1 LedInit()	2478
7.862.2.2 LedToggle()	2479
7.863led.h File Reference	2479
7.863.1 Detailed Description	2479

7.863.2 Function Documentation	2479
7.863.2.1 LedBlinkExit()	2480
7.863.2.2 LedBlinkInit()	2480
7.863.2.3 LedBlinkTask()	2480
7.864led.h File Reference	2480
7.864.1 Detailed Description	2481
7.864.2 Function Documentation	2481
7.864.2.1 LedInit()	2481
7.864.2.2 LedToggle()	2482
7.865led.h File Reference	2482
7.865.1 Detailed Description	2482
7.865.2 Function Documentation	2482
7.865.2.1 LedBlinkExit()	2483
7.865.2.2 LedBlinkInit()	2483
7.865.2.3 LedBlinkTask()	2483
7.866led.h File Reference	2483
7.866.1 Detailed Description	2484
7.866.2 Function Documentation	2484
7.866.2.1 LedInit()	2484
7.866.2.2 LedToggle()	2485
7.867led.h File Reference	2485
7.867.1 Detailed Description	2485
7.867.2 Function Documentation	2485
7.867.2.1 LedBlinkExit()	2486
7.867.2.2 LedBlinkInit()	2486
7.867.2.3 LedBlinkTask()	2486
7.868led.h File Reference	2486
7.868.1 Detailed Description	2487
7.868.2 Function Documentation	2487
7.868.2.1 LedInit()	2487

7.868.2.2 LedToggle()	2488
7.869led.h File Reference	2488
7.869.1 Detailed Description	2488
7.869.2 Function Documentation	2488
7.869.2.1 LedBlinkExit()	2489
7.869.2.2 LedBlinkInit()	2489
7.869.2.3 LedBlinkTask()	2489
7.870led.h File Reference	2489
7.870.1 Detailed Description	2490
7.870.2 Function Documentation	2490
7.870.2.1 LedInit()	2490
7.870.2.2 LedToggle()	2491
7.871led.h File Reference	2491
7.871.1 Detailed Description	2491
7.871.2 Function Documentation	2491
7.871.2.1 LedBlinkExit()	2492
7.871.2.2 LedBlinkInit()	2492
7.871.2.3 LedBlinkTask()	2492
7.872led.h File Reference	2492
7.872.1 Detailed Description	2493
7.872.2 Function Documentation	2493
7.872.2.1 LedInit()	2493
7.872.2.2 LedToggle()	2494
7.873led.h File Reference	2494
7.873.1 Detailed Description	2494
7.873.2 Function Documentation	2494
7.873.2.1 LedBlinkExit()	2495
7.873.2.2 LedBlinkInit()	2495
7.873.2.3 LedBlinkTask()	2495
7.874led.h File Reference	2495

7.874.1 Detailed Description	2496
7.874.2 Function Documentation	2496
7.874.2.1 LedInit()	2496
7.874.2.2 LedToggle()	2497
7.875led.h File Reference	2497
7.875.1 Detailed Description	2497
7.875.2 Function Documentation	2497
7.875.2.1 LedBlinkExit()	2498
7.875.2.2 LedBlinkInit()	2498
7.875.2.3 LedBlinkTask()	2498
7.876led.h File Reference	2498
7.876.1 Detailed Description	2499
7.876.2 Function Documentation	2499
7.876.2.1 LedInit()	2499
7.876.2.2 LedToggle()	2500
7.877led.h File Reference	2500
7.877.1 Detailed Description	2500
7.877.2 Function Documentation	2500
7.877.2.1 LedBlinkExit()	2501
7.877.2.2 LedBlinkInit()	2501
7.877.2.3 LedBlinkTask()	2501
7.878led.h File Reference	2501
7.878.1 Detailed Description	2502
7.878.2 Function Documentation	2502
7.878.2.1 LedInit()	2502
7.878.2.2 LedToggle()	2503
7.879led.h File Reference	2503
7.879.1 Detailed Description	2503
7.879.2 Function Documentation	2503
7.879.2.1 LedBlinkExit()	2504

7.879.2.2 LedBlinkInit()	2504
7.879.2.3 LedBlinkTask()	2504
7.880led.h File Reference	2504
7.880.1 Detailed Description	2505
7.880.2 Function Documentation	2505
7.880.2.1 LedInit()	2505
7.880.2.2 LedToggle()	2506
7.881led.h File Reference	2506
7.881.1 Detailed Description	2506
7.881.2 Function Documentation	2506
7.881.2.1 LedBlinkExit()	2507
7.881.2.2 LedBlinkInit()	2507
7.881.2.3 LedBlinkTask()	2507
7.882led.h File Reference	2507
7.882.1 Detailed Description	2508
7.882.2 Function Documentation	2508
7.882.2.1 LedInit()	2508
7.882.2.2 LedToggle()	2509
7.883led.h File Reference	2509
7.883.1 Detailed Description	2509
7.883.2 Function Documentation	2509
7.883.2.1 LedBlinkExit()	2510
7.883.2.2 LedBlinkInit()	2510
7.883.2.3 LedBlinkTask()	2510
7.884led.h File Reference	2510
7.884.1 Detailed Description	2511
7.884.2 Function Documentation	2511
7.884.2.1 LedInit()	2511
7.884.2.2 LedToggle()	2512
7.885led.h File Reference	2512

7.885.1 Detailed Description	2512
7.885.2 Function Documentation	2512
7.885.2.1 LedBlinkExit()	2513
7.885.2.2 LedBlinkInit()	2513
7.885.2.3 LedBlinkTask()	2513
7.886led.h File Reference	2513
7.886.1 Detailed Description	2514
7.886.2 Function Documentation	2514
7.886.2.1 LedInit()	2514
7.886.2.2 LedToggle()	2515
7.887led.h File Reference	2515
7.887.1 Detailed Description	2515
7.887.2 Function Documentation	2515
7.887.2.1 LedBlinkExit()	2516
7.887.2.2 LedBlinkInit()	2516
7.887.2.3 LedBlinkTask()	2516
7.888led.h File Reference	2516
7.888.1 Detailed Description	2517
7.888.2 Function Documentation	2517
7.888.2.1 LedInit()	2517
7.888.2.2 LedToggle()	2518
7.889led.h File Reference	2518
7.889.1 Detailed Description	2518
7.889.2 Function Documentation	2518
7.889.2.1 LedBlinkExit()	2519
7.889.2.2 LedBlinkInit()	2519
7.889.2.3 LedBlinkTask()	2519
7.890led.h File Reference	2519
7.890.1 Detailed Description	2520
7.890.2 Function Documentation	2520

7.890.2.1 LedInit()	2520
7.890.2.2 LedToggle()	2521
7.891led.h File Reference	2521
7.891.1 Detailed Description	2521
7.891.2 Function Documentation	2521
7.891.2.1 LedBlinkExit()	2522
7.891.2.2 LedBlinkInit()	2522
7.891.2.3 LedBlinkTask()	2522
7.892led.h File Reference	2522
7.892.1 Detailed Description	2523
7.892.2 Function Documentation	2523
7.892.2.1 LedInit()	2523
7.892.2.2 LedToggle()	2524
7.893led.h File Reference	2524
7.893.1 Detailed Description	2524
7.893.2 Function Documentation	2524
7.893.2.1 LedBlinkExit()	2525
7.893.2.2 LedBlinkInit()	2525
7.893.2.3 LedBlinkTask()	2525
7.894led.h File Reference	2525
7.894.1 Detailed Description	2526
7.894.2 Function Documentation	2526
7.894.2.1 LedInit()	2526
7.894.2.2 LedToggle()	2527
7.895led.h File Reference	2527
7.895.1 Detailed Description	2527
7.895.2 Function Documentation	2527
7.895.2.1 LedBlinkExit()	2528
7.895.2.2 LedBlinkInit()	2528
7.895.2.3 LedBlinkTask()	2528

7.896led.h File Reference	2528
7.896.1 Detailed Description	2529
7.896.2 Function Documentation	2529
7.896.2.1 LedInit()	2529
7.896.2.2 LedToggle()	2530
7.897led.h File Reference	2530
7.897.1 Detailed Description	2530
7.897.2 Function Documentation	2530
7.897.2.1 LedBlinkExit()	2531
7.897.2.2 LedBlinkInit()	2531
7.897.2.3 LedBlinkTask()	2531
7.898led.h File Reference	2531
7.898.1 Detailed Description	2532
7.898.2 Function Documentation	2532
7.898.2.1 LedBlinkExit()	2532
7.898.2.2 LedBlinkInit()	2532
7.898.2.3 LedBlinkTask()	2533
7.899led.h File Reference	2533
7.899.1 Detailed Description	2534
7.899.2 Function Documentation	2534
7.899.2.1 LedBlinkExit()	2534
7.899.2.2 LedBlinkInit()	2534
7.899.2.3 LedBlinkTask()	2534
7.900led.h File Reference	2535
7.900.1 Detailed Description	2535
7.900.2 Function Documentation	2535
7.900.2.1 LedBlinkExit()	2535
7.900.2.2 LedBlinkInit()	2535
7.900.2.3 LedBlinkTask()	2536
7.901led.h File Reference	2536

7.901.1 Detailed Description	2537
7.901.2 Function Documentation	2537
7.901.2.1 LedInit()	2537
7.901.2.2 LedToggle()	2537
7.902led.h File Reference	2537
7.902.1 Detailed Description	2538
7.902.2 Function Documentation	2538
7.902.2.1 LedBlinkExit()	2538
7.902.2.2 LedBlinkInit()	2538
7.902.2.3 LedBlinkTask()	2539
7.903led.h File Reference	2539
7.903.1 Detailed Description	2539
7.903.2 Function Documentation	2539
7.903.2.1 LedInit()	2540
7.903.2.2 LedToggle()	2540
7.904led.h File Reference	2540
7.904.1 Detailed Description	2541
7.904.2 Function Documentation	2541
7.904.2.1 LedBlinkExit()	2541
7.904.2.2 LedBlinkInit()	2541
7.904.2.3 LedBlinkTask()	2542
7.905led.h File Reference	2542
7.905.1 Detailed Description	2542
7.905.2 Function Documentation	2542
7.905.2.1 LedInit()	2543
7.905.2.2 LedToggle()	2543
7.906led.h File Reference	2543
7.906.1 Detailed Description	2544
7.906.2 Function Documentation	2544
7.906.2.1 LedBlinkExit()	2544

7.906.2.2 LedBlinkInit()	2544
7.906.2.3 LedBlinkTask()	2545
7.907led.h File Reference	2545
7.907.1 Detailed Description	2545
7.907.2 Function Documentation	2545
7.907.2.1 LedInit()	2546
7.907.2.2 LedToggle()	2546
7.908led.h File Reference	2546
7.908.1 Detailed Description	2547
7.908.2 Function Documentation	2547
7.908.2.1 LedBlinkExit()	2547
7.908.2.2 LedBlinkInit()	2547
7.908.2.3 LedBlinkTask()	2548
7.909led.h File Reference	2548
7.909.1 Detailed Description	2548
7.909.2 Function Documentation	2548
7.909.2.1 LedInit()	2549
7.909.2.2 LedToggle()	2549
7.910led.h File Reference	2549
7.910.1 Detailed Description	2550
7.910.2 Function Documentation	2550
7.910.2.1 LedBlinkExit()	2550
7.910.2.2 LedBlinkInit()	2550
7.910.2.3 LedBlinkTask()	2551
7.911led.h File Reference	2551
7.911.1 Detailed Description	2551
7.911.2 Function Documentation	2551
7.911.2.1 LedInit()	2552
7.911.2.2 LedToggle()	2552
7.912led.h File Reference	2552

7.912.1 Detailed Description	2553
7.912.2 Function Documentation	2553
7.912.2.1 LedBlinkExit()	2553
7.912.2.2 LedBlinkInit()	2553
7.912.2.3 LedBlinkTask()	2554
7.913led.h File Reference	2554
7.913.1 Detailed Description	2554
7.913.2 Function Documentation	2554
7.913.2.1 LedInit()	2555
7.913.2.2 LedToggle()	2555
7.914led.h File Reference	2555
7.914.1 Detailed Description	2556
7.914.2 Function Documentation	2556
7.914.2.1 LedBlinkExit()	2556
7.914.2.2 LedBlinkInit()	2556
7.914.2.3 LedBlinkTask()	2557
7.915led.h File Reference	2557
7.915.1 Detailed Description	2557
7.915.2 Function Documentation	2557
7.915.2.1 LedInit()	2558
7.915.2.2 LedToggle()	2558
7.916led.h File Reference	2558
7.916.1 Detailed Description	2559
7.916.2 Function Documentation	2559
7.916.2.1 LedInit()	2559
7.916.2.2 LedToggle()	2559
7.917led.h File Reference	2560
7.917.1 Detailed Description	2560
7.917.2 Function Documentation	2560
7.917.2.1 LedBlinkExit()	2560

7.917.2.2 LedBlinkInit()	2560
7.917.2.3 LedBlinkTask()	2561
7.918led.h File Reference	2561
7.918.1 Detailed Description	2562
7.918.2 Function Documentation	2562
7.918.2.1 LedInit()	2562
7.918.2.2 LedToggle()	2562
7.919led.h File Reference	2562
7.919.1 Detailed Description	2563
7.919.2 Function Documentation	2563
7.919.2.1 LedBlinkExit()	2563
7.919.2.2 LedBlinkInit()	2563
7.919.2.3 LedBlinkTask()	2564
7.920led.h File Reference	2564
7.920.1 Detailed Description	2564
7.920.2 Function Documentation	2564
7.920.2.1 LedInit()	2565
7.920.2.2 LedToggle()	2565
7.921led.h File Reference	2565
7.921.1 Detailed Description	2566
7.921.2 Function Documentation	2566
7.921.2.1 LedBlinkExit()	2566
7.921.2.2 LedBlinkInit()	2566
7.921.2.3 LedBlinkTask()	2567
7.922led.h File Reference	2567
7.922.1 Detailed Description	2567
7.922.2 Function Documentation	2567
7.922.2.1 LedInit()	2568
7.922.2.2 LedToggle()	2568
7.923led.h File Reference	2568

7.923.1 Detailed Description	2569
7.923.2 Function Documentation	2569
7.923.2.1 LedBlinkExit()	2569
7.923.2.2 LedBlinkInit()	2569
7.923.2.3 LedBlinkTask()	2570
7.924led.h File Reference	2570
7.924.1 Detailed Description	2570
7.924.2 Function Documentation	2570
7.924.2.1 LedInit()	2571
7.924.2.2 LedToggle()	2571
7.925led.h File Reference	2571
7.925.1 Detailed Description	2572
7.925.2 Function Documentation	2572
7.925.2.1 LedBlinkExit()	2572
7.925.2.2 LedBlinkInit()	2572
7.925.2.3 LedBlinkTask()	2573
7.926led.h File Reference	2573
7.926.1 Detailed Description	2573
7.926.2 Function Documentation	2573
7.926.2.1 LedInit()	2574
7.926.2.2 LedToggle()	2574
7.927led.h File Reference	2574
7.927.1 Detailed Description	2575
7.927.2 Function Documentation	2575
7.927.2.1 LedBlinkExit()	2575
7.927.2.2 LedBlinkInit()	2575
7.927.2.3 LedBlinkTask()	2576
7.928led.h File Reference	2576
7.928.1 Detailed Description	2576
7.928.2 Function Documentation	2576

7.928.2.1 LedInit()	2577
7.928.2.2 LedToggle()	2577
7.929led.h File Reference	2577
7.929.1 Detailed Description	2578
7.929.2 Function Documentation	2578
7.929.2.1 LedBlinkExit()	2578
7.929.2.2 LedBlinkInit()	2578
7.929.2.3 LedBlinkTask()	2579
7.930led.h File Reference	2579
7.930.1 Detailed Description	2579
7.930.2 Function Documentation	2579
7.930.2.1 LedInit()	2580
7.930.2.2 LedToggle()	2580
7.931led.h File Reference	2580
7.931.1 Detailed Description	2581
7.931.2 Function Documentation	2581
7.931.2.1 LedBlinkExit()	2581
7.931.2.2 LedBlinkInit()	2581
7.931.2.3 LedBlinkTask()	2582
7.932led.h File Reference	2582
7.932.1 Detailed Description	2582
7.932.2 Function Documentation	2582
7.932.2.1 LedInit()	2583
7.932.2.2 LedToggle()	2583
7.933led.h File Reference	2583
7.933.1 Detailed Description	2584
7.933.2 Function Documentation	2584
7.933.2.1 LedBlinkExit()	2584
7.933.2.2 LedBlinkInit()	2584
7.933.2.3 LedBlinkTask()	2585

7.934led.h File Reference	2585
7.934.1 Detailed Description	2585
7.934.2 Function Documentation	2585
7.934.2.1 LedInit()	2586
7.934.2.2 LedToggle()	2586
7.935led.h File Reference	2586
7.935.1 Detailed Description	2587
7.935.2 Function Documentation	2587
7.935.2.1 LedBlinkExit()	2587
7.935.2.2 LedBlinkInit()	2587
7.935.2.3 LedBlinkTask()	2588
7.936led.h File Reference	2588
7.936.1 Detailed Description	2588
7.936.2 Function Documentation	2588
7.936.2.1 LedInit()	2589
7.936.2.2 LedToggle()	2589
7.937led.h File Reference	2589
7.937.1 Detailed Description	2590
7.937.2 Function Documentation	2590
7.937.2.1 LedBlinkExit()	2590
7.937.2.2 LedBlinkInit()	2590
7.937.2.3 LedBlinkTask()	2591
7.938led.h File Reference	2591
7.938.1 Detailed Description	2591
7.938.2 Function Documentation	2591
7.938.2.1 LedInit()	2592
7.938.2.2 LedToggle()	2592
7.939led.h File Reference	2592
7.939.1 Detailed Description	2593
7.939.2 Function Documentation	2593

7.939.2.1 LedBlinkExit()	2593
7.939.2.2 LedBlinkInit()	2593
7.939.2.3 LedBlinkTask()	2594
7.940led.h File Reference	2594
7.940.1 Detailed Description	2594
7.940.2 Function Documentation	2594
7.940.2.1 LedInit()	2595
7.940.2.2 LedToggle()	2595
7.941led.h File Reference	2595
7.941.1 Detailed Description	2596
7.941.2 Function Documentation	2596
7.941.2.1 LedBlinkExit()	2596
7.941.2.2 LedBlinkInit()	2596
7.941.2.3 LedBlinkTask()	2597
7.942led.h File Reference	2597
7.942.1 Detailed Description	2597
7.942.2 Function Documentation	2597
7.942.2.1 LedInit()	2598
7.942.2.2 LedToggle()	2598
7.943led.h File Reference	2598
7.943.1 Detailed Description	2599
7.943.2 Function Documentation	2599
7.943.2.1 LedBlinkExit()	2599
7.943.2.2 LedBlinkInit()	2599
7.943.2.3 LedBlinkTask()	2600
7.944led.h File Reference	2600
7.944.1 Detailed Description	2600
7.944.2 Function Documentation	2600
7.944.2.1 LedInit()	2601
7.944.2.2 LedToggle()	2601

7.945led.h File Reference	2601
7.945.1 Detailed Description	2602
7.945.2 Function Documentation	2602
7.945.2.1 LedBlinkExit()	2602
7.945.2.2 LedBlinkInit()	2602
7.945.2.3 LedBlinkTask()	2603
7.946led.h File Reference	2603
7.946.1 Detailed Description	2604
7.946.2 Function Documentation	2604
7.946.2.1 LedInit()	2604
7.946.2.2 LedToggle()	2604
7.947led.h File Reference	2604
7.947.1 Detailed Description	2605
7.947.2 Function Documentation	2605
7.947.2.1 LedInit()	2605
7.947.2.2 LedToggle()	2605
7.948main.c File Reference	2606
7.948.1 Detailed Description	2606
7.948.2 Function Documentation	2606
7.948.2.1 Init()	2606
7.948.2.2 main()	2607
7.949main.c File Reference	2607
7.949.1 Detailed Description	2607
7.949.2 Function Documentation	2608
7.949.2.1 Init()	2608
7.949.2.2 main()	2608
7.950main.c File Reference	2608
7.950.1 Detailed Description	2609
7.950.2 Function Documentation	2609
7.950.2.1 HAL_MspDeInit()	2609

7.950.2.2 HAL_Msplnit()	2609
7.950.2.3 Init()	2610
7.950.2.4 main()	2610
7.950.2.5 SystemClock_Config()	2610
7.951main.c File Reference	2611
7.951.1 Detailed Description	2611
7.951.2 Function Documentation	2611
7.951.2.1 HAL_MspDeInit()	2612
7.951.2.2 HAL_Msplnit()	2612
7.951.2.3 Init()	2612
7.951.2.4 main()	2612
7.951.2.5 SystemClock_Config()	2613
7.952main.c File Reference	2613
7.952.1 Detailed Description	2614
7.952.2 Function Documentation	2614
7.952.2.1 HAL_MspDeInit()	2614
7.952.2.2 HAL_Msplnit()	2614
7.952.2.3 Init()	2614
7.952.2.4 main()	2615
7.952.2.5 SystemClock_Config()	2615
7.953main.c File Reference	2615
7.953.1 Detailed Description	2616
7.953.2 Function Documentation	2616
7.953.2.1 HAL_MspDeInit()	2616
7.953.2.2 HAL_Msplnit()	2616
7.953.2.3 Init()	2617
7.953.2.4 main()	2617
7.953.2.5 SystemClock_Config()	2617
7.954main.c File Reference	2618
7.954.1 Detailed Description	2618

7.954.2 Function Documentation	2618
7.954.2.1 HAL_MspDeInit()	2619
7.954.2.2 HAL_MspInit()	2619
7.954.2.3 Init()	2619
7.954.2.4 main()	2619
7.954.2.5 SystemClock_Config()	2620
7.955main.c File Reference	2620
7.955.1 Detailed Description	2621
7.955.2 Function Documentation	2621
7.955.2.1 HAL_MspDeInit()	2621
7.955.2.2 HAL_MspInit()	2621
7.955.2.3 Init()	2621
7.955.2.4 main()	2622
7.955.2.5 SystemClock_Config()	2622
7.956main.c File Reference	2622
7.956.1 Detailed Description	2623
7.956.2 Function Documentation	2623
7.956.2.1 HAL_MspDeInit()	2623
7.956.2.2 HAL_MspInit()	2623
7.956.2.3 Init()	2624
7.956.2.4 main()	2624
7.956.2.5 SystemClock_Config()	2624
7.957main.c File Reference	2625
7.957.1 Detailed Description	2625
7.957.2 Function Documentation	2625
7.957.2.1 HAL_MspDeInit()	2626
7.957.2.2 HAL_MspInit()	2626
7.957.2.3 Init()	2626
7.957.2.4 main()	2626
7.957.2.5 SystemClock_Config()	2627

7.958main.c File Reference	2627
7.958.1 Detailed Description	2628
7.958.2 Function Documentation	2628
7.958.2.1 HAL_MspDeInit()	2628
7.958.2.2 HAL_MspInit()	2628
7.958.2.3 Init()	2628
7.958.2.4 main()	2629
7.958.2.5 SystemClock_Config()	2629
7.959main.c File Reference	2629
7.959.1 Detailed Description	2630
7.959.2 Function Documentation	2630
7.959.2.1 HAL_MspDeInit()	2630
7.959.2.2 HAL_MspInit()	2630
7.959.2.3 Init()	2631
7.959.2.4 main()	2631
7.959.2.5 SystemClock_Config()	2631
7.960main.c File Reference	2632
7.960.1 Detailed Description	2632
7.960.2 Function Documentation	2632
7.960.2.1 HAL_MspDeInit()	2633
7.960.2.2 HAL_MspInit()	2633
7.960.2.3 Init()	2633
7.960.2.4 main()	2633
7.960.2.5 SystemClock_Config()	2634
7.961main.c File Reference	2634
7.961.1 Detailed Description	2635
7.961.2 Function Documentation	2635
7.961.2.1 HAL_MspDeInit()	2635
7.961.2.2 HAL_MspInit()	2635
7.961.2.3 Init()	2635

7.961.2.4 main()	2636
7.961.2.5 SystemClock_Config()	2636
7.962main.c File Reference	2636
7.962.1 Detailed Description	2637
7.962.2 Function Documentation	2637
7.962.2.1 HAL_MspDeInit()	2637
7.962.2.2 HAL_MspInit()	2637
7.962.2.3 Init()	2638
7.962.2.4 main()	2638
7.962.2.5 SystemClock_Config()	2638
7.963main.c File Reference	2639
7.963.1 Detailed Description	2639
7.963.2 Function Documentation	2639
7.963.2.1 HAL_MspDeInit()	2640
7.963.2.2 HAL_MspInit()	2640
7.963.2.3 Init()	2640
7.963.2.4 main()	2640
7.963.2.5 SystemClock_Config()	2641
7.963.2.6 VectorBase_Config()	2641
7.964main.c File Reference	2641
7.964.1 Detailed Description	2642
7.964.2 Function Documentation	2642
7.964.2.1 HAL_MspDeInit()	2642
7.964.2.2 HAL_MspInit()	2642
7.964.2.3 Init()	2643
7.964.2.4 main()	2643
7.964.2.5 SystemClock_Config()	2643
7.965main.c File Reference	2644
7.965.1 Detailed Description	2644
7.965.2 Function Documentation	2644

7.965.2.1 HAL_MspDeInit()	2645
7.965.2.2 HAL_MspInit()	2645
7.965.2.3 Init()	2645
7.965.2.4 main()	2645
7.965.2.5 SystemClock_Config()	2646
7.965.2.6 VectorBase_Config()	2646
7.966main.c File Reference	2646
7.966.1 Detailed Description	2647
7.966.2 Function Documentation	2647
7.966.2.1 HAL_MspDeInit()	2647
7.966.2.2 HAL_MspInit()	2647
7.966.2.3 Init()	2648
7.966.2.4 main()	2648
7.966.2.5 SystemClock_Config()	2648
7.967main.c File Reference	2649
7.967.1 Detailed Description	2649
7.967.2 Function Documentation	2649
7.967.2.1 HAL_MspDeInit()	2650
7.967.2.2 HAL_MspInit()	2650
7.967.2.3 Init()	2650
7.967.2.4 main()	2650
7.967.2.5 SystemClock_Config()	2651
7.967.2.6 VectorBase_Config()	2651
7.968main.c File Reference	2651
7.968.1 Detailed Description	2652
7.968.2 Function Documentation	2652
7.968.2.1 main()	2652
7.968.2.2 PostInit()	2652
7.969main.c File Reference	2653
7.969.1 Detailed Description	2653

7.969.2 Function Documentation	2653
7.969.2.1 Init()	2653
7.969.2.2 main()	2654
7.970main.c File Reference	2654
7.970.1 Detailed Description	2654
7.970.2 Function Documentation	2655
7.970.2.1 Init()	2655
7.970.2.2 main()	2655
7.970.2.3 PostInit()	2655
7.971main.c File Reference	2656
7.971.1 Detailed Description	2656
7.971.2 Function Documentation	2656
7.971.2.1 Init()	2656
7.971.2.2 main()	2657
7.972main.c File Reference	2657
7.972.1 Detailed Description	2658
7.972.2 Function Documentation	2658
7.972.2.1 HAL_MspDeInit()	2658
7.972.2.2 HAL_MspInit()	2658
7.972.2.3 Init()	2658
7.972.2.4 main()	2659
7.972.2.5 SystemClock_Config()	2659
7.973main.c File Reference	2659
7.973.1 Detailed Description	2660
7.973.2 Function Documentation	2660
7.973.2.1 HAL_MspDeInit()	2660
7.973.2.2 HAL_MspInit()	2660
7.973.2.3 Init()	2661
7.973.2.4 main()	2661
7.973.2.5 SystemClock_Config()	2661

7.973.2.6 VectorBase_Config()	2662
7.974main.c File Reference	2662
7.974.1 Detailed Description	2663
7.974.2 Function Documentation	2663
7.974.2.1 HAL_MspDeInit()	2663
7.974.2.2 HAL_Msplnit()	2663
7.974.2.3 Init()	2663
7.974.2.4 main()	2664
7.974.2.5 SystemClock_Config()	2664
7.975main.c File Reference	2664
7.975.1 Detailed Description	2665
7.975.2 Function Documentation	2665
7.975.2.1 HAL_MspDeInit()	2665
7.975.2.2 HAL_Msplnit()	2665
7.975.2.3 Init()	2666
7.975.2.4 main()	2666
7.975.2.5 SystemClock_Config()	2666
7.975.2.6 VectorBase_Config()	2667
7.976main.c File Reference	2667
7.976.1 Detailed Description	2668
7.976.2 Function Documentation	2668
7.976.2.1 HAL_MspDeInit()	2668
7.976.2.2 HAL_Msplnit()	2668
7.976.2.3 Init()	2668
7.976.2.4 main()	2669
7.976.2.5 SystemClock_Config()	2669
7.977main.c File Reference	2669
7.977.1 Detailed Description	2670
7.977.2 Function Documentation	2670
7.977.2.1 HAL_MspDeInit()	2670

7.977.2.2 HAL_Msplnit()	2670
7.977.2.3 Init()	2671
7.977.2.4 main()	2671
7.977.2.5 SystemClock_Config()	2671
7.977.2.6 VectorBase_Config()	2672
7.978main.c File Reference	2672
7.978.1 Detailed Description	2672
7.978.2 Function Documentation	2673
7.978.2.1 Init()	2673
7.978.2.2 main()	2673
7.979main.c File Reference	2673
7.979.1 Detailed Description	2674
7.979.2 Function Documentation	2674
7.979.2.1 Init()	2674
7.979.2.2 main()	2674
7.980main.c File Reference	2675
7.980.1 Detailed Description	2675
7.980.2 Function Documentation	2675
7.980.2.1 Init()	2675
7.980.2.2 main()	2676
7.981main.c File Reference	2676
7.981.1 Detailed Description	2676
7.981.2 Function Documentation	2676
7.981.2.1 Init()	2677
7.981.2.2 main()	2677
7.982main.c File Reference	2677
7.982.1 Detailed Description	2678
7.982.2 Function Documentation	2678
7.982.2.1 Init()	2678
7.982.2.2 main()	2678

7.983main.c File Reference	2679
7.983.1 Detailed Description	2679
7.983.2 Function Documentation	2679
7.983.2.1 Init()	2679
7.983.2.2 main()	2680
7.984main.c File Reference	2680
7.984.1 Detailed Description	2680
7.984.2 Function Documentation	2680
7.984.2.1 Init()	2681
7.984.2.2 main()	2681
7.985main.c File Reference	2681
7.985.1 Detailed Description	2682
7.985.2 Function Documentation	2682
7.985.2.1 Init()	2682
7.985.2.2 main()	2682
7.986main.c File Reference	2682
7.986.1 Detailed Description	2683
7.986.2 Function Documentation	2683
7.986.2.1 Init()	2683
7.986.2.2 main()	2683
7.987main.c File Reference	2684
7.987.1 Detailed Description	2684
7.987.2 Function Documentation	2684
7.987.2.1 Init()	2684
7.987.2.2 main()	2685
7.988main.c File Reference	2685
7.988.1 Detailed Description	2685
7.988.2 Function Documentation	2685
7.988.2.1 Init()	2686
7.988.2.2 main()	2686

7.989main.c File Reference	2686
7.989.1 Detailed Description	2687
7.989.2 Function Documentation	2687
7.989.2.1 Init()	2687
7.989.2.2 main()	2687
7.990main.c File Reference	2687
7.990.1 Detailed Description	2688
7.990.2 Function Documentation	2688
7.990.2.1 HAL_MspDeInit()	2688
7.990.2.2 HAL_MspInit()	2688
7.990.2.3 Init()	2689
7.990.2.4 main()	2689
7.990.2.5 SystemClock_Config()	2689
7.991main.c File Reference	2690
7.991.1 Detailed Description	2690
7.991.2 Function Documentation	2690
7.991.2.1 HAL_MspDeInit()	2691
7.991.2.2 HAL_MspInit()	2691
7.991.2.3 Init()	2691
7.991.2.4 main()	2691
7.991.2.5 SystemClock_Config()	2692
7.991.2.6 VectorBase_Config()	2692
7.992main.c File Reference	2692
7.992.1 Detailed Description	2693
7.992.2 Function Documentation	2693
7.992.2.1 HAL_MspDeInit()	2693
7.992.2.2 HAL_MspInit()	2693
7.992.2.3 Init()	2694
7.992.2.4 main()	2694
7.992.2.5 SystemClock_Config()	2694

7.993main.c File Reference	2695
7.993.1 Detailed Description	2695
7.993.2 Function Documentation	2695
7.993.2.1 HAL_MspDeInit()	2696
7.993.2.2 HAL_MspInit()	2696
7.993.2.3 Init()	2696
7.993.2.4 main()	2696
7.993.2.5 SystemClock_Config()	2697
7.993.2.6 VectorBase_Config()	2697
7.994main.c File Reference	2697
7.994.1 Detailed Description	2698
7.994.2 Function Documentation	2698
7.994.2.1 HAL_MspDeInit()	2698
7.994.2.2 HAL_MspInit()	2698
7.994.2.3 Init()	2699
7.994.2.4 main()	2699
7.994.2.5 SystemClock_Config()	2699
7.995main.c File Reference	2700
7.995.1 Detailed Description	2700
7.995.2 Function Documentation	2700
7.995.2.1 HAL_MspDeInit()	2701
7.995.2.2 HAL_MspInit()	2701
7.995.2.3 Init()	2701
7.995.2.4 main()	2701
7.995.2.5 SystemClock_Config()	2702
7.995.2.6 VectorBase_Config()	2702
7.996main.c File Reference	2702
7.996.1 Detailed Description	2703
7.996.2 Function Documentation	2703
7.996.2.1 HAL_MspDeInit()	2703

7.996.2.2 HAL_Msplnit()	2703
7.996.2.3 Init()	2704
7.996.2.4 main()	2704
7.996.2.5 SystemClock_Config()	2704
7.997main.c File Reference	2705
7.997.1 Detailed Description	2705
7.997.2 Function Documentation	2705
7.997.2.1 HAL_MspDeInit()	2706
7.997.2.2 HAL_Msplnit()	2706
7.997.2.3 Init()	2706
7.997.2.4 main()	2706
7.997.2.5 SystemClock_Config()	2707
7.997.2.6 VectorBase_Config()	2707
7.998main.c File Reference	2707
7.998.1 Detailed Description	2708
7.998.2 Function Documentation	2708
7.998.2.1 HAL_MspDeInit()	2708
7.998.2.2 HAL_Msplnit()	2708
7.998.2.3 Init()	2709
7.998.2.4 main()	2709
7.998.2.5 SystemClock_Config()	2709
7.999main.c File Reference	2710
7.999.1 Detailed Description	2710
7.999.2 Function Documentation	2710
7.999.2.1 HAL_MspDeInit()	2711
7.999.2.2 HAL_Msplnit()	2711
7.999.2.3 Init()	2711
7.999.2.4 main()	2711
7.999.2.5 SystemClock_Config()	2712
7.999.2.6 VectorBase_Config()	2712

7.1000	main.c File Reference	2712
7.1000.1	Detailed Description	2713
7.1000.2	Function Documentation	2713
7.1000.2.1	HAL_MspDeInit()	2713
7.1000.2.2	HAL_MspInit()	2713
7.1000.2.3	init()	2714
7.1000.2.4	main()	2714
7.1000.2.5	SystemClock_Config()	2714
7.1001	main.c File Reference	2715
7.1001.1	Detailed Description	2715
7.1001.2	Function Documentation	2715
7.1001.2.1	HAL_MspDeInit()	2716
7.1001.2.2	HAL_MspInit()	2716
7.1001.2.3	init()	2716
7.1001.2.4	main()	2716
7.1001.2.5	SystemClock_Config()	2717
7.1001.2.6	VectorBase_Config()	2717
7.1002	main.c File Reference	2717
7.1002.1	Detailed Description	2718
7.1002.2	Function Documentation	2718
7.1002.2.1	HAL_MspDeInit()	2718
7.1002.2.2	HAL_MspInit()	2718
7.1002.2.3	init()	2719
7.1002.2.4	main()	2719
7.1002.2.5	SystemClock_Config()	2719
7.1003	main.c File Reference	2720
7.1003.1	Detailed Description	2720
7.1003.2	Function Documentation	2720
7.1003.2.1	HAL_MspDeInit()	2721
7.1003.2.2	HAL_MspInit()	2721

7.1003.2.3	init()	2721
7.1003.2.4	main()	2721
7.1003.2.5	SystemClock_Config()	2722
7.1003.2.6	VectorBase_Config()	2722
7.1004	main.c File Reference	2722
7.1004.1	Detailed Description	2723
7.1004.2	Function Documentation	2723
7.1004.2.1	HAL_MspDeInit()	2723
7.1004.2.2	HAL_MspInit()	2723
7.1004.2.3	init()	2724
7.1004.2.4	main()	2724
7.1004.2.5	SystemClock_Config()	2724
7.1005	main.c File Reference	2725
7.1005.1	Detailed Description	2725
7.1005.2	Function Documentation	2725
7.1005.2.1	HAL_MspDeInit()	2726
7.1005.2.2	HAL_MspInit()	2726
7.1005.2.3	init()	2726
7.1005.2.4	main()	2726
7.1005.2.5	SystemClock_Config()	2727
7.1005.2.6	VectorBase_Config()	2727
7.1006	main.c File Reference	2727
7.1006.1	Detailed Description	2728
7.1006.2	Function Documentation	2728
7.1006.2.1	HAL_MspDeInit()	2728
7.1006.2.2	HAL_MspInit()	2728
7.1006.2.3	init()	2729
7.1006.2.4	main()	2729
7.1006.2.5	SystemClock_Config()	2729
7.1007	main.c File Reference	2730

7.1007.1Detailed Description	2730
7.1007.2Function Documentation	2730
7.1007.2.1HAL_MspDeInit()	2731
7.1007.2.2HAL_MspInit()	2731
7.1007.2.3nit()	2731
7.1007.2.4main()	2731
7.1007.2.5SystemClock_Config()	2732
7.1007.2.6VectorBase_Config()	2732
7.1008main.c File Reference	2732
7.1008.1Detailed Description	2733
7.1008.2Function Documentation	2733
7.1008.2.1HAL_MspDeInit()	2733
7.1008.2.2HAL_MspInit()	2733
7.1008.2.3nit()	2734
7.1008.2.4main()	2734
7.1008.2.5SystemClock_Config()	2734
7.1009main.c File Reference	2735
7.1009.1Detailed Description	2735
7.1009.2Function Documentation	2735
7.1009.2.1HAL_MspDeInit()	2736
7.1009.2.2HAL_MspInit()	2736
7.1009.2.3nit()	2736
7.1009.2.4main()	2736
7.1009.2.5SystemClock_Config()	2737
7.1009.2.6VectorBase_Config()	2737
7.1010main.c File Reference	2737
7.1010.1Detailed Description	2738
7.1010.2Function Documentation	2738
7.1010.2.1HAL_MspDeInit()	2738
7.1010.2.2HAL_MspInit()	2738

7.1010.2.3	init()	2739
7.1010.2.4	main()	2739
7.1010.2.5	SystemClock_Config()	2739
7.1011	main.c File Reference	2740
7.1011.1	Detailed Description	2740
7.1011.2	Function Documentation	2740
7.1011.2.1	HAL_MspDeInit()	2741
7.1011.2.2	HAL_MspInit()	2741
7.1011.2.3	init()	2741
7.1011.2.4	main()	2741
7.1011.2.5	SystemClock_Config()	2742
7.1011.2.6	VectorBase_Config()	2742
7.1012	main.c File Reference	2742
7.1012.1	Detailed Description	2743
7.1012.2	Function Documentation	2743
7.1012.2.1	HAL_MspDeInit()	2743
7.1012.2.2	HAL_MspInit()	2743
7.1012.2.3	init()	2744
7.1012.2.4	main()	2744
7.1012.2.5	SystemClock_Config()	2744
7.1013	main.c File Reference	2745
7.1013.1	Detailed Description	2745
7.1013.2	Function Documentation	2745
7.1013.2.1	HAL_MspDeInit()	2746
7.1013.2.2	HAL_MspInit()	2746
7.1013.2.3	init()	2746
7.1013.2.4	main()	2746
7.1013.2.5	SystemClock_Config()	2747
7.1013.2.6	VectorBase_Config()	2747
7.1014	main.c File Reference	2747

7.1014.1Detailed Description	2748
7.1014.2Function Documentation	2748
7.1014.2.1HAL_MspDeInit()	2748
7.1014.2.2HAL_MspInit()	2748
7.1014.2.3nit()	2749
7.1014.2.4main()	2749
7.1014.2.5SystemClock_Config()	2749
7.1015main.c File Reference	2750
7.1015.1Detailed Description	2750
7.1015.2Function Documentation	2750
7.1015.2.1HAL_MspDeInit()	2751
7.1015.2.2HAL_MspInit()	2751
7.1015.2.3nit()	2751
7.1015.2.4main()	2751
7.1015.2.5SystemClock_Config()	2752
7.1015.2.6VectorBase_Config()	2752
7.1016main.c File Reference	2752
7.1016.1Detailed Description	2753
7.1016.2Function Documentation	2753
7.1016.2.1HAL_MspDeInit()	2753
7.1016.2.2HAL_MspInit()	2753
7.1016.2.3nit()	2754
7.1016.2.4main()	2754
7.1016.2.5SystemClock_Config()	2754
7.1017main.c File Reference	2755
7.1017.1Detailed Description	2755
7.1017.2Function Documentation	2755
7.1017.2.1HAL_MspDeInit()	2756
7.1017.2.2HAL_MspInit()	2756
7.1017.2.3nit()	2756

7.1017.2.4	main()	2756
7.1017.2.5	SystemClock_Config()	2757
7.1017.2.6	VectorBase_Config()	2757
7.1018	main.c File Reference	2757
7.1018.1	Detailed Description	2758
7.1018.2	Function Documentation	2758
7.1018.2.1	HAL_MspDeInit()	2758
7.1018.2.2	HAL_MspInit()	2758
7.1018.2.3	Init()	2759
7.1018.2.4	main()	2759
7.1018.2.5	SystemClock_Config()	2759
7.1019	main.c File Reference	2760
7.1019.1	Detailed Description	2760
7.1019.2	Function Documentation	2760
7.1019.2.1	HAL_MspDeInit()	2761
7.1019.2.2	HAL_MspInit()	2761
7.1019.2.3	Init()	2761
7.1019.2.4	main()	2761
7.1019.2.5	SystemClock_Config()	2762
7.1019.2.6	VectorBase_Config()	2762
7.1020	main.c File Reference	2762
7.1020.1	Detailed Description	2763
7.1020.2	Function Documentation	2763
7.1020.2.1	Init()	2763
7.1020.2.2	main()	2763
7.1020.2.3	SystemClockConfig()	2764
7.1021	main.c File Reference	2764
7.1021.1	Detailed Description	2765
7.1021.2	Function Documentation	2765
7.1021.2.1	Init()	2765

7.1021.2.2main()	2765
7.1021.2.3SystemClockConfig()	2765
7.1022main.c File Reference	2766
7.1022.1Detailed Description	2766
7.1022.2Function Documentation	2766
7.1022.2.1Init()	2766
7.1022.2.2main()	2767
7.1022.2.3SystemClockConfig()	2767
7.1023main.c File Reference	2767
7.1023.1Detailed Description	2768
7.1023.2Function Documentation	2768
7.1023.2.1Init()	2768
7.1023.2.2main()	2768
7.1023.2.3SystemClockConfig()	2769
7.1024main.c File Reference	2769
7.1024.1Detailed Description	2770
7.1024.2Function Documentation	2770
7.1024.2.1HAL_MspDeInit()	2770
7.1024.2.2HAL_MspInit()	2770
7.1024.2.3Init()	2771
7.1024.2.4main()	2771
7.1024.2.5SystemClock_Config()	2771
7.1025main.c File Reference	2772
7.1025.1Detailed Description	2772
7.1025.2Function Documentation	2773
7.1025.2.1HAL_MspDeInit()	2773
7.1025.2.2HAL_MspInit()	2773
7.1025.2.3Init()	2773
7.1025.2.4main()	2774
7.1025.2.5SystemClock_Config()	2774

7.1025.2.6VectorBase_Config()	2774
7.1026main.c File Reference	2775
7.1026.1Detailed Description	2775
7.1026.2Function Documentation	2775
7.1026.2.1HAL_MspDeInit()	2776
7.1026.2.2HAL_MspInit()	2776
7.1026.2.3Init()	2776
7.1026.2.4main()	2776
7.1026.2.5SystemClock_Config()	2777
7.1027main.c File Reference	2777
7.1027.1Detailed Description	2778
7.1027.2Function Documentation	2778
7.1027.2.1HAL_MspDeInit()	2778
7.1027.2.2HAL_MspInit()	2778
7.1027.2.3Init()	2779
7.1027.2.4main()	2779
7.1027.2.5SystemClock_Config()	2779
7.1027.2.6VectorBase_Config()	2780
7.1028main.c File Reference	2780
7.1028.1Detailed Description	2781
7.1028.2Function Documentation	2781
7.1028.2.1HAL_MspDeInit()	2781
7.1028.2.2HAL_MspInit()	2781
7.1028.2.3Init()	2781
7.1028.2.4main()	2782
7.1028.2.5SystemClock_Config()	2782
7.1029main.c File Reference	2782
7.1029.1Detailed Description	2783
7.1029.2Function Documentation	2783
7.1029.2.1HAL_MspDeInit()	2783

7.1029.2.2	HAL_MspInit()	2783
7.1029.2.3	init()	2784
7.1029.2.4	main()	2784
7.1029.2.5	SystemClock_Config()	2784
7.1029.2.6	VectorBase_Config()	2785
7.1030	main.c File Reference	2785
7.1030.1	Detailed Description	2786
7.1030.2	Function Documentation	2786
7.1030.2.1	HAL_MspDeInit()	2786
7.1030.2.2	HAL_MspInit()	2786
7.1030.2.3	init()	2786
7.1030.2.4	main()	2787
7.1030.2.5	SystemClock_Config()	2787
7.1031	main.c File Reference	2787
7.1031.1	Detailed Description	2788
7.1031.2	Function Documentation	2788
7.1031.2.1	HAL_MspDeInit()	2788
7.1031.2.2	HAL_MspInit()	2788
7.1031.2.3	init()	2789
7.1031.2.4	main()	2789
7.1031.2.5	SystemClock_Config()	2789
7.1031.2.6	VectorBase_Config()	2790
7.1032	main.c File Reference	2790
7.1032.1	Detailed Description	2791
7.1032.2	Function Documentation	2791
7.1032.2.1	HAL_MspDeInit()	2791
7.1032.2.2	HAL_MspInit()	2791
7.1032.2.3	init()	2791
7.1032.2.4	main()	2792
7.1032.2.5	SystemClock_Config()	2792

7.1033	main.c File Reference	2792
7.1033.1	Detailed Description	2793
7.1033.2	Function Documentation	2793
7.1033.2.1	HAL_MspDeInit()	2793
7.1033.2.2	HAL_MspInit()	2793
7.1033.2.3	init()	2794
7.1033.2.4	main()	2794
7.1033.2.5	SystemClock_Config()	2794
7.1033.2.6	VectorBase_Config()	2795
7.1034	main.c File Reference	2795
7.1034.1	Detailed Description	2796
7.1034.2	Function Documentation	2796
7.1034.2.1	HAL_MspDeInit()	2796
7.1034.2.2	HAL_MspInit()	2796
7.1034.2.3	init()	2796
7.1034.2.4	main()	2797
7.1034.2.5	SystemClock_Config()	2797
7.1035	main.c File Reference	2797
7.1035.1	Detailed Description	2798
7.1035.2	Function Documentation	2798
7.1035.2.1	HAL_MspDeInit()	2798
7.1035.2.2	HAL_MspInit()	2798
7.1035.2.3	init()	2799
7.1035.2.4	main()	2799
7.1035.2.5	SystemClock_Config()	2799
7.1035.2.6	VectorBase_Config()	2800
7.1036	main.c File Reference	2800
7.1036.1	Detailed Description	2801
7.1036.2	Function Documentation	2801
7.1036.2.1	HAL_MspDeInit()	2801

7.1036.2.2HAL_Msplnit()	2801
7.1036.2.3nit()	2801
7.1036.2.4main()	2802
7.1036.2.5SystemClock_Config()	2802
7.1037main.c File Reference	2802
7.1037.1Detailed Description	2803
7.1037.2Function Documentation	2803
7.1037.2.1HAL_MspDeInit()	2803
7.1037.2.2HAL_Msplnit()	2803
7.1037.2.3nit()	2804
7.1037.2.4main()	2804
7.1037.2.5SystemClock_Config()	2804
7.1037.2.6VectorBase_Config()	2805
7.1038main.c File Reference	2805
7.1038.1Detailed Description	2806
7.1038.2Function Documentation	2806
7.1038.2.1HAL_MspDeInit()	2806
7.1038.2.2HAL_Msplnit()	2806
7.1038.2.3nit()	2806
7.1038.2.4main()	2807
7.1038.2.5SystemClock_Config()	2807
7.1039main.c File Reference	2807
7.1039.1Detailed Description	2808
7.1039.2Function Documentation	2808
7.1039.2.1HAL_MspDeInit()	2808
7.1039.2.2HAL_Msplnit()	2808
7.1039.2.3nit()	2809
7.1039.2.4main()	2809
7.1039.2.5SystemClock_Config()	2809
7.1039.2.6VectorBase_Config()	2810

7.1040	main.c File Reference	2810
7.1040.1	Detailed Description	2811
7.1040.2	Function Documentation	2811
7.1040.2.1	HAL_MspDeInit()	2811
7.1040.2.2	HAL_MspInit()	2811
7.1040.2.3	init()	2811
7.1040.2.4	main()	2812
7.1040.2.5	SystemClock_Config()	2812
7.1041	main.c File Reference	2812
7.1041.1	Detailed Description	2813
7.1041.2	Function Documentation	2813
7.1041.2.1	HAL_MspDeInit()	2813
7.1041.2.2	HAL_MspInit()	2813
7.1041.2.3	init()	2814
7.1041.2.4	main()	2814
7.1041.2.5	SystemClock_Config()	2814
7.1041.2.6	VectorBase_Config()	2815
7.1042	main.c File Reference	2815
7.1042.1	Detailed Description	2816
7.1042.2	Function Documentation	2816
7.1042.2.1	HAL_MspDeInit()	2816
7.1042.2.2	HAL_MspInit()	2816
7.1042.2.3	init()	2816
7.1042.2.4	main()	2817
7.1042.2.5	SystemClock_Config()	2817
7.1043	main.c File Reference	2817
7.1043.1	Detailed Description	2818
7.1043.2	Function Documentation	2818
7.1043.2.1	HAL_MspDeInit()	2818
7.1043.2.2	HAL_MspInit()	2818

7.1043.2.3	init()	2819
7.1043.2.4	main()	2819
7.1043.2.5	SystemClock_Config()	2819
7.1043.2.6	VectorBase_Config()	2820
7.1044	main.c File Reference	2820
7.1044.1	Detailed Description	2821
7.1044.2	Function Documentation	2821
7.1044.2.1	HAL_MspDeInit()	2821
7.1044.2.2	HAL_MspInit()	2821
7.1044.2.3	init()	2821
7.1044.2.4	main()	2822
7.1044.2.5	SystemClock_Config()	2822
7.1045	main.c File Reference	2822
7.1045.1	Detailed Description	2823
7.1045.2	Function Documentation	2823
7.1045.2.1	HAL_MspDeInit()	2823
7.1045.2.2	HAL_MspInit()	2823
7.1045.2.3	init()	2824
7.1045.2.4	main()	2824
7.1045.2.5	SystemClock_Config()	2824
7.1045.2.6	VectorBase_Config()	2825
7.1046	main.c File Reference	2825
7.1046.1	Detailed Description	2826
7.1046.2	Function Documentation	2826
7.1046.2.1	HAL_MspDeInit()	2826
7.1046.2.2	HAL_MspInit()	2826
7.1046.2.3	init()	2826
7.1046.2.4	main()	2827
7.1046.2.5	SystemClock_Config()	2827
7.1047	main.c File Reference	2827

7.1047.1Detailed Description	2828
7.1047.2Function Documentation	2828
7.1047.2.1HAL_MspDeInit()	2828
7.1047.2.2HAL_MspInit()	2828
7.1047.2.3init()	2829
7.1047.2.4main()	2829
7.1047.2.5SystemClock_Config()	2829
7.1047.2.6VectorBase_Config()	2830
7.1048main.c File Reference	2830
7.1048.1Detailed Description	2831
7.1048.2Function Documentation	2831
7.1048.2.1HAL_MspDeInit()	2831
7.1048.2.2HAL_MspInit()	2831
7.1048.2.3init()	2831
7.1048.2.4main()	2832
7.1048.2.5SystemClock_Config()	2832
7.1049main.c File Reference	2832
7.1049.1Detailed Description	2833
7.1049.2Function Documentation	2833
7.1049.2.1HAL_MspDeInit()	2833
7.1049.2.2HAL_MspInit()	2833
7.1049.2.3init()	2834
7.1049.2.4main()	2834
7.1049.2.5SystemClock_Config()	2834
7.1049.2.6VectorBase_Config()	2835
7.1050main.c File Reference	2835
7.1050.1Detailed Description	2836
7.1050.2Function Documentation	2836
7.1050.2.1HAL_MspDeInit()	2836
7.1050.2.2HAL_MspInit()	2836

7.1050.2.3init()	2836
7.1050.2.4main()	2837
7.1050.2.5SystemClock_Config()	2837
7.1051main.c File Reference	2837
7.1051.1Detailed Description	2838
7.1051.2Function Documentation	2838
7.1051.2.1HAL_MspDeInit()	2838
7.1051.2.2HAL_MspInit()	2838
7.1051.2.3init()	2839
7.1051.2.4main()	2839
7.1051.2.5SystemClock_Config()	2839
7.1051.2.6VectorBase_Config()	2840
7.1052main.c File Reference	2840
7.1052.1Detailed Description	2841
7.1052.2Function Documentation	2841
7.1052.2.1HAL_MspDeInit()	2841
7.1052.2.2HAL_MspInit()	2841
7.1052.2.3init()	2841
7.1052.2.4main()	2842
7.1052.2.5SystemClock_Config()	2842
7.1053main.c File Reference	2842
7.1053.1Detailed Description	2843
7.1053.2Function Documentation	2843
7.1053.2.1HAL_MspDeInit()	2843
7.1053.2.2HAL_MspInit()	2843
7.1053.2.3init()	2844
7.1053.2.4main()	2844
7.1053.2.5SystemClock_Config()	2844
7.1053.2.6VectorBase_Config()	2845
7.1054main.c File Reference	2845

7.1054.1Detailed Description	2845
7.1054.2Function Documentation	2846
7.1054.2.1Init()	2846
7.1054.2.2main()	2846
7.1055main.c File Reference	2846
7.1055.1Detailed Description	2847
7.1055.2Function Documentation	2847
7.1055.2.1Init()	2847
7.1055.2.2main()	2847
7.1056main.c File Reference	2847
7.1056.1Detailed Description	2848
7.1056.2Function Documentation	2848
7.1056.2.1Init()	2848
7.1056.2.2main()	2848
7.1056.2.3PostInit()	2849
7.1057main.c File Reference	2849
7.1057.1Detailed Description	2849
7.1057.2Function Documentation	2849
7.1057.2.1Init()	2850
7.1057.2.2main()	2850
7.1058main.c File Reference	2850
7.1058.1Detailed Description	2851
7.1058.2Function Documentation	2851
7.1058.2.1Init()	2851
7.1058.2.2main()	2851
7.1058.2.3PostInit()	2852
7.1059main.c File Reference	2852
7.1059.1Detailed Description	2852
7.1059.2Function Documentation	2852
7.1059.2.1Init()	2853

7.1059.2.2main()	2853
7.1060main.c File Reference	2853
7.1060.1Detailed Description	2854
7.1060.2Function Documentation	2854
7.1060.2.1HAL_MspDeInit()	2854
7.1060.2.2HAL_MspInit()	2854
7.1060.2.3init()	2855
7.1060.2.4main()	2855
7.1060.2.5SystemClock_Config()	2855
7.1061main.c File Reference	2856
7.1061.1Detailed Description	2856
7.1061.2Function Documentation	2856
7.1061.2.1HAL_MspDeInit()	2857
7.1061.2.2HAL_MspInit()	2857
7.1061.2.3init()	2857
7.1061.2.4main()	2857
7.1061.2.5SystemClock_Config()	2858
7.1061.2.6VectorBase_Config()	2858
7.1062main.c File Reference	2858
7.1062.1Detailed Description	2859
7.1062.2Function Documentation	2859
7.1062.2.1HAL_MspDeInit()	2859
7.1062.2.2HAL_MspInit()	2859
7.1062.2.3init()	2860
7.1062.2.4main()	2860
7.1062.2.5SystemClock_Config()	2860
7.1063main.c File Reference	2861
7.1063.1Detailed Description	2861
7.1063.2Function Documentation	2861
7.1063.2.1HAL_MspDeInit()	2862

7.1063.2.2	HAL_MspInit()	2862
7.1063.2.3	init()	2862
7.1063.2.4	main()	2862
7.1063.2.5	SystemClock_Config()	2863
7.1063.2.6	VectorBase_Config()	2863
7.1064	main.c File Reference	2863
7.1064.1	Detailed Description	2864
7.1064.2	Function Documentation	2864
7.1064.2.1	HAL_MspDeInit()	2864
7.1064.2.2	HAL_MspInit()	2864
7.1064.2.3	init()	2865
7.1064.2.4	main()	2865
7.1064.2.5	SystemClock_Config()	2865
7.1065	main.c File Reference	2866
7.1065.1	Detailed Description	2866
7.1065.2	Function Documentation	2866
7.1065.2.1	HAL_MspDeInit()	2867
7.1065.2.2	HAL_MspInit()	2867
7.1065.2.3	init()	2867
7.1065.2.4	main()	2867
7.1065.2.5	SystemClock_Config()	2868
7.1065.2.6	VectorBase_Config()	2868
7.1066	main.c File Reference	2868
7.1066.1	Detailed Description	2869
7.1066.2	Function Documentation	2869
7.1066.2.1	HAL_MspDeInit()	2869
7.1066.2.2	HAL_MspInit()	2869
7.1066.2.3	init()	2870
7.1066.2.4	main()	2870
7.1066.2.5	SystemClock_Config()	2870

7.1067	main.c File Reference	2871
7.1067.1	Detailed Description	2871
7.1067.2	Function Documentation	2871
7.1067.2.1	HAL_MspDeInit()	2872
7.1067.2.2	HAL_MspInit()	2872
7.1067.2.3	init()	2872
7.1067.2.4	main()	2872
7.1067.2.5	SystemClock_Config()	2873
7.1067.2.6	VectorBase_Config()	2873
7.1068	main.c File Reference	2873
7.1068.1	Detailed Description	2874
7.1068.2	Function Documentation	2874
7.1068.2.1	HAL_MspDeInit()	2874
7.1068.2.2	HAL_MspInit()	2874
7.1068.2.3	init()	2875
7.1068.2.4	main()	2875
7.1068.2.5	SystemClock_Config()	2875
7.1069	main.c File Reference	2876
7.1069.1	Detailed Description	2876
7.1069.2	Function Documentation	2876
7.1069.2.1	HAL_MspDeInit()	2877
7.1069.2.2	HAL_MspInit()	2877
7.1069.2.3	init()	2877
7.1069.2.4	main()	2877
7.1069.2.5	SystemClock_Config()	2878
7.1069.2.6	VectorBase_Config()	2878
7.1070	main.c File Reference	2878
7.1070.1	Detailed Description	2879
7.1070.2	Function Documentation	2879
7.1070.2.1	HAL_MspDeInit()	2879

7.1070.2.2HAL_Msplnit()	2879
7.1070.2.3nit()	2880
7.1070.2.4main()	2880
7.1070.2.5SystemClock_Config()	2880
7.1070main.c File Reference	2881
7.1071.1Detailed Description	2881
7.1071.2Function Documentation	2881
7.1071.2.1HAL_MspDeInit()	2882
7.1071.2.2HAL_Msplnit()	2882
7.1071.2.3nit()	2882
7.1071.2.4main()	2882
7.1071.2.5SystemClock_Config()	2883
7.1071.2.6VectorBase_Config()	2883
7.1070main.c File Reference	2883
7.1072.1Detailed Description	2884
7.1072.2Function Documentation	2884
7.1072.2.1HAL_MspDeInit()	2884
7.1072.2.2HAL_Msplnit()	2884
7.1072.2.3nit()	2885
7.1072.2.4main()	2885
7.1072.2.5SystemClock_Config()	2885
7.1070main.c File Reference	2886
7.1073.1Detailed Description	2886
7.1073.2Function Documentation	2886
7.1073.2.1HAL_MspDeInit()	2887
7.1073.2.2HAL_Msplnit()	2887
7.1073.2.3nit()	2887
7.1073.2.4main()	2887
7.1073.2.5SystemClock_Config()	2888
7.1073.2.6VectorBase_Config()	2888

7.1074	main.c File Reference	2888
7.1074.1	Detailed Description	2889
7.1074.2	Function Documentation	2889
7.1074.2.1	HAL_MspDeInit()	2889
7.1074.2.2	HAL_MspInit()	2889
7.1074.2.3	init()	2890
7.1074.2.4	main()	2890
7.1074.2.5	SystemClock_Config()	2890
7.1075	main.c File Reference	2891
7.1075.1	Detailed Description	2891
7.1075.2	Function Documentation	2891
7.1075.2.1	HAL_MspDeInit()	2892
7.1075.2.2	HAL_MspInit()	2892
7.1075.2.3	init()	2892
7.1075.2.4	main()	2892
7.1075.2.5	SystemClock_Config()	2893
7.1075.2.6	VectorBase_Config()	2893
7.1076	main.c File Reference	2893
7.1076.1	Detailed Description	2894
7.1076.2	Function Documentation	2894
7.1076.2.1	HAL_MspDeInit()	2894
7.1076.2.2	HAL_MspInit()	2894
7.1076.2.3	init()	2895
7.1076.2.4	main()	2895
7.1076.2.5	SystemClock_Config()	2895
7.1077	main.c File Reference	2896
7.1077.1	Detailed Description	2896
7.1077.2	Function Documentation	2896
7.1077.2.1	HAL_MspDeInit()	2897
7.1077.2.2	HAL_MspInit()	2897

7.1077.2.3Init()	2897
7.1077.2.4main()	2897
7.1077.2.5SystemClock_Config()	2898
7.1077.2.6VectorBase_Config()	2898
7.1078main.c File Reference	2898
7.1078.1Detailed Description	2899
7.1078.2Function Documentation	2899
7.1078.2.1Init()	2899
7.1078.2.2main()	2899
7.1078.2.3SystemClockInit()	2900
7.1079main.c File Reference	2900
7.1079.1Detailed Description	2900
7.1079.2Function Documentation	2901
7.1079.2.1Init()	2901
7.1079.2.2main()	2901
7.1079.2.3SystemClockInit()	2901
7.1080main.c File Reference	2902
7.1080.1Detailed Description	2902
7.1080.2Function Documentation	2902
7.1080.2.1Init()	2902
7.1080.2.2main()	2903
7.1081main.c File Reference	2903
7.1081.1Detailed Description	2903
7.1081.2Function Documentation	2904
7.1081.2.1Init()	2904
7.1081.2.2main()	2904
7.1081.2.3SysClockInit()	2904
7.1082et.c File Reference	2905
7.1082.1Detailed Description	2905
7.1082.2Function Documentation	2906

7.1082.2.1NetApp()	2906
7.1082.2.2NetInit()	2906
7.1082.2.3NetTask()	2906
7.1083.net.c File Reference	2907
7.1083.1Detailed Description	2907
7.1083.2Function Documentation	2908
7.1083.2.1NetApp()	2908
7.1083.2.2NetInit()	2908
7.1083.2.3NetTask()	2908
7.1084.net.c File Reference	2909
7.1084.1Detailed Description	2910
7.1084.2Function Documentation	2910
7.1084.2.1NetApp()	2910
7.1084.2.2NetInit()	2910
7.1084.2.3NetTask()	2910
7.1085.net.c File Reference	2911
7.1085.1Detailed Description	2912
7.1085.2Function Documentation	2912
7.1085.2.1NetApp()	2912
7.1085.2.2NetInit()	2912
7.1085.2.3NetTask()	2912
7.1086.net.c File Reference	2913
7.1086.1Detailed Description	2914
7.1086.2Function Documentation	2914
7.1086.2.1NetApp()	2914
7.1086.2.2NetInit()	2914
7.1086.2.3NetTask()	2914
7.1087.net.c File Reference	2915
7.1087.1Detailed Description	2916
7.1087.2Function Documentation	2916

7.1087.2.1NetApp()	2916
7.1087.2.2NetInit()	2916
7.1087.2.3NetTask()	2916
7.1088et.c File Reference	2917
7.1088.1Detailed Description	2918
7.1088.2Function Documentation	2918
7.1088.2.1NetApp()	2918
7.1088.2.2NetInit()	2918
7.1088.2.3NetTask()	2918
7.1089et.c File Reference	2919
7.1089.1Detailed Description	2920
7.1089.2Function Documentation	2920
7.1089.2.1NetApp()	2920
7.1089.2.2NetInit()	2920
7.1089.2.3NetTask()	2920
7.1090et.c File Reference	2921
7.1090.1Detailed Description	2922
7.1090.2Function Documentation	2922
7.1090.2.1NetApp()	2922
7.1090.2.2NetInit()	2922
7.1090.2.3NetTask()	2922
7.1091et.c File Reference	2923
7.1091.1Detailed Description	2924
7.1091.2Function Documentation	2924
7.1091.2.1NetApp()	2924
7.1091.2.2NetInit()	2924
7.1091.2.3NetTask()	2924
7.1092et.c File Reference	2925
7.1092.1Detailed Description	2926
7.1092.2Function Documentation	2926

7.1092.2.1NetApp()	2926
7.1092.2.2NetInit()	2926
7.1092.2.3NetTask()	2926
7.1093.net.c File Reference	2927
7.1093.1Detailed Description	2928
7.1093.2Function Documentation	2928
7.1093.2.1NetApp()	2928
7.1093.2.2NetInit()	2928
7.1093.2.3NetTask()	2928
7.1094.net.c File Reference	2929
7.1094.1Detailed Description	2930
7.1094.2Function Documentation	2930
7.1094.2.1NetApp()	2930
7.1094.2.2NetInit()	2930
7.1094.2.3NetReceivePacket()	2930
7.1094.2.4NetServerTask()	2931
7.1094.2.5NetTransmitPacket()	2931
7.1095.net.h File Reference	2932
7.1095.1Detailed Description	2932
7.1095.2Function Documentation	2932
7.1095.2.1NetApp()	2933
7.1095.2.2NetInit()	2933
7.1095.2.3NetTask()	2933
7.1096.net.h File Reference	2934
7.1096.1Detailed Description	2934
7.1096.2Function Documentation	2934
7.1096.2.1NetApp()	2935
7.1096.2.2NetInit()	2935
7.1096.2.3NetTask()	2935
7.1097.net.h File Reference	2936

7.1097.1Detailed Description	2936
7.1097.2Function Documentation	2936
7.1097.2.1NetApp()	2937
7.1097.2.2NetInit()	2937
7.1097.2.3NetTask()	2937
7.1098.net.h File Reference	2938
7.1098.1Detailed Description	2938
7.1098.2Function Documentation	2938
7.1098.2.1NetApp()	2939
7.1098.2.2NetInit()	2939
7.1098.2.3NetTask()	2939
7.1099.net.h File Reference	2940
7.1099.1Detailed Description	2940
7.1099.2Function Documentation	2940
7.1099.2.1NetApp()	2941
7.1099.2.2NetInit()	2941
7.1099.2.3NetTask()	2941
7.1100.net.h File Reference	2942
7.1100.1Detailed Description	2942
7.1100.2Function Documentation	2942
7.1100.2.1NetApp()	2943
7.1100.2.2NetInit()	2943
7.1100.2.3NetTask()	2943
7.1101.net.h File Reference	2944
7.1101.1Detailed Description	2944
7.1101.2Function Documentation	2944
7.1101.2.1NetApp()	2945
7.1101.2.2NetInit()	2945
7.1101.2.3NetTask()	2945
7.1102.net.h File Reference	2946

Generated by Doxygen

7.1107.1Detailed Description	2956
7.1107.2Function Documentation	2956
7.1107.2.1NetApp()	2957
7.1107.2.2NetInit()	2957
7.1107.2.3NetReceivePacket()	2957
7.1107.2.4NetTransmitPacket()	2958
7.1108vm.c File Reference	2958
7.1108.1Detailed Description	2959
7.1108.2Function Documentation	2959
7.1108.2.1NvmDone()	2960
7.1108.2.2NvmDoneHook()	2960
7.1108.2.3NvmErase()	2960
7.1108.2.4NvmEraseHook()	2961
7.1108.2.5NvmGetUserProgBaseAddress()	2961
7.1108.2.6NvmInit()	2961
7.1108.2.7NvmInitHook()	2962
7.1108.2.8NvmReinit()	2962
7.1108.2.9NvmReinitHook()	2962
7.1108.2.10NvmVerifyChecksum()	2963
7.1108.2.11NvmWrite()	2963
7.1108.2.12NvmWriteHook()	2963
7.1109vm.c File Reference	2964
7.1109.1Detailed Description	2965
7.1109.2Function Documentation	2965
7.1109.2.1NvmDone()	2965
7.1109.2.2NvmDoneHook()	2966
7.1109.2.3NvmErase()	2966
7.1109.2.4NvmEraseHook()	2966
7.1109.2.5NvmGetUserProgBaseAddress()	2967
7.1109.2.6NvmInit()	2967

7.1109.2.7NvmInitHook()	2967
7.1109.2.8NvmReinit()	2968
7.1109.2.9NvmReinitHook()	2968
7.1109.2.10NvmVerifyChecksum()	2968
7.1109.2.11NvmWrite()	2968
7.1109.2.12NvmWriteHook()	2969
7.1110Nvm.c File Reference	2969
7.1110.1Detailed Description	2971
7.1110.2Function Documentation	2971
7.1110.2.1NvmDone()	2971
7.1110.2.2NvmDoneHook()	2971
7.1110.2.3NvmErase()	2971
7.1110.2.4NvmEraseHook()	2972
7.1110.2.5NvmGetUserProgBaseAddress()	2972
7.1110.2.6NvmInit()	2973
7.1110.2.7NvmInitHook()	2973
7.1110.2.8NvmReinit()	2973
7.1110.2.9NvmReinitHook()	2973
7.1110.2.10NvmVerifyChecksum()	2974
7.1110.2.11NvmWrite()	2974
7.1110.2.12NvmWriteHook()	2974
7.1111Nvm.c File Reference	2975
7.1111.1Detailed Description	2976
7.1111.2Function Documentation	2976
7.1111.2.1NvmDone()	2976
7.1111.2.2NvmDoneHook()	2976
7.1111.2.3NvmErase()	2976
7.1111.2.4NvmEraseHook()	2977
7.1111.2.5NvmGetUserProgBaseAddress()	2977
7.1111.2.6NvmInit()	2978

7.1111.2.7NvmInitHook()	2978
7.1111.2.8NvmReinit()	2978
7.1111.2.9NvmReinitHook()	2978
7.1111.2.10NvmVerifyChecksum()	2979
7.1111.2.11NvmWrite()	2979
7.1111.2.12NvmWriteHook()	2979
7.1112Nvm.c File Reference	2980
7.1112.1Detailed Description	2981
7.1112.2Function Documentation	2981
7.1112.2.1NvmDone()	2981
7.1112.2.2NvmDoneHook()	2981
7.1112.2.3NvmErase()	2981
7.1112.2.4NvmEraseHook()	2982
7.1112.2.5NvmGetUserProgBaseAddress()	2982
7.1112.2.6NvmInit()	2983
7.1112.2.7NvmInitHook()	2983
7.1112.2.8NvmReinit()	2983
7.1112.2.9NvmReinitHook()	2983
7.1112.2.10NvmVerifyChecksum()	2984
7.1112.2.11NvmWrite()	2984
7.1112.2.12NvmWriteHook()	2984
7.1113Nvm.c File Reference	2985
7.1113.1Detailed Description	2986
7.1113.2Function Documentation	2986
7.1113.2.1NvmDone()	2986
7.1113.2.2NvmDoneHook()	2986
7.1113.2.3NvmErase()	2986
7.1113.2.4NvmEraseHook()	2987
7.1113.2.5NvmGetUserProgBaseAddress()	2987
7.1113.2.6NvmInit()	2988

7.1113.2.7NvmInitHook()	2988
7.1113.2.8NvmReinit()	2988
7.1113.2.9NvmReinitHook()	2988
7.1113.2.10NvmVerifyChecksum()	2989
7.1113.2.11NvmWrite()	2989
7.1113.2.12NvmWriteHook()	2989
7.1114Nvm.c File Reference	2990
7.1114.1Detailed Description	2991
7.1114.2Function Documentation	2991
7.1114.2.1NvmDone()	2991
7.1114.2.2NvmDoneHook()	2991
7.1114.2.3NvmErase()	2991
7.1114.2.4NvmEraseHook()	2992
7.1114.2.5NvmGetUserProgBaseAddress()	2992
7.1114.2.6NvmInit()	2993
7.1114.2.7NvmInitHook()	2993
7.1114.2.8NvmReinit()	2993
7.1114.2.9NvmReinitHook()	2993
7.1114.2.10NvmVerifyChecksum()	2994
7.1114.2.11NvmWrite()	2994
7.1114.2.12NvmWriteHook()	2994
7.1115vm.c File Reference	2995
7.1115.1Detailed Description	2996
7.1115.2Function Documentation	2996
7.1115.2.1NvmDone()	2996
7.1115.2.2NvmDoneHook()	2996
7.1115.2.3NvmErase()	2996
7.1115.2.4NvmEraseHook()	2997
7.1115.2.5NvmGetUserProgBaseAddress()	2997
7.1115.2.6NvmInit()	2998

7.1115.2.7NvmInitHook()	2998
7.1115.2.8NvmReinit()	2998
7.1115.2.9NvmReinitHook()	2998
7.1115.2.10NvmVerifyChecksum()	2999
7.1115.2.11NvmWrite()	2999
7.1115.2.12NvmWriteHook()	2999
7.1116Nvm.c File Reference	3000
7.1116.1Detailed Description	3001
7.1116.2Function Documentation	3001
7.1116.2.1NvmDone()	3001
7.1116.2.2NvmDoneHook()	3001
7.1116.2.3NvmErase()	3001
7.1116.2.4NvmEraseHook()	3002
7.1116.2.5NvmGetUserProgBaseAddress()	3002
7.1116.2.6NvmInit()	3003
7.1116.2.7NvmInitHook()	3003
7.1116.2.8NvmReinit()	3003
7.1116.2.9NvmReinitHook()	3003
7.1116.2.10NvmVerifyChecksum()	3004
7.1116.2.11NvmWrite()	3004
7.1116.2.12NvmWriteHook()	3004
7.1117Nvm.c File Reference	3005
7.1117.1Detailed Description	3006
7.1117.2Function Documentation	3006
7.1117.2.1NvmDone()	3006
7.1117.2.2NvmDoneHook()	3006
7.1117.2.3NvmErase()	3006
7.1117.2.4NvmEraseHook()	3007
7.1117.2.5NvmGetUserProgBaseAddress()	3007
7.1117.2.6NvmInit()	3008

7.1117.2.7NvmInitHook()	3008
7.1117.2.8NvmReinit()	3008
7.1117.2.9NvmReinitHook()	3008
7.1117.2.10NvmVerifyChecksum()	3009
7.1117.2.11NvmWrite()	3009
7.1117.2.12NvmWriteHook()	3009
7.1118Nvm.c File Reference	3010
7.1118.1Detailed Description	3011
7.1118.2Function Documentation	3011
7.1118.2.1NvmDone()	3011
7.1118.2.2NvmDoneHook()	3011
7.1118.2.3NvmErase()	3011
7.1118.2.4NvmEraseHook()	3012
7.1118.2.5NvmGetUserProgBaseAddress()	3012
7.1118.2.6NvmInit()	3013
7.1118.2.7NvmInitHook()	3013
7.1118.2.8NvmReinit()	3013
7.1118.2.9NvmReinitHook()	3013
7.1118.2.10NvmVerifyChecksum()	3014
7.1118.2.11NvmWrite()	3014
7.1118.2.12NvmWriteHook()	3014
7.1119Nvm.c File Reference	3015
7.1119.1Detailed Description	3016
7.1119.2Function Documentation	3016
7.1119.2.1NvmDone()	3016
7.1119.2.2NvmDoneHook()	3016
7.1119.2.3NvmErase()	3016
7.1119.2.4NvmEraseHook()	3017
7.1119.2.5NvmGetUserProgBaseAddress()	3017
7.1119.2.6NvmInit()	3018

7.1119.2.7NvmInitHook()	3018
7.1119.2.8NvmReinit()	3018
7.1119.2.9NvmReinitHook()	3018
7.1119.2.10NvmVerifyChecksum()	3019
7.1119.2.11NvmWrite()	3019
7.1119.2.12NvmWriteHook()	3019
7.1120Nvm.c File Reference	3020
7.1120.1Detailed Description	3021
7.1120.2Function Documentation	3021
7.1120.2.1NvmDone()	3021
7.1120.2.2NvmDoneHook()	3021
7.1120.2.3NvmErase()	3021
7.1120.2.4NvmEraseHook()	3022
7.1120.2.5NvmGetUserProgBaseAddress()	3022
7.1120.2.6NvmInit()	3023
7.1120.2.7NvmInitHook()	3023
7.1120.2.8NvmReinit()	3023
7.1120.2.9NvmReinitHook()	3023
7.1120.2.10NvmVerifyChecksum()	3024
7.1120.2.11NvmWrite()	3024
7.1120.2.12NvmWriteHook()	3024
7.1121Nvm.c File Reference	3025
7.1121.1Detailed Description	3026
7.1121.2Function Documentation	3026
7.1121.2.1NvmDone()	3026
7.1121.2.2NvmDoneHook()	3026
7.1121.2.3NvmErase()	3026
7.1121.2.4NvmEraseHook()	3027
7.1121.2.5NvmGetUserProgBaseAddress()	3027
7.1121.2.6NvmInit()	3028

7.1121.2.7NvmInitHook()	3028
7.1121.2.8NvmReinit()	3028
7.1121.2.9NvmReinitHook()	3028
7.1121.2.10NvmVerifyChecksum()	3029
7.1121.2.11NvmWrite()	3029
7.1121.2.12NvmWriteHook()	3029
7.1122Nvm.c File Reference	3030
7.1122.1Detailed Description	3031
7.1122.2Function Documentation	3031
7.1122.2.1NvmDone()	3031
7.1122.2.2NvmDoneHook()	3031
7.1122.2.3NvmErase()	3031
7.1122.2.4NvmEraseHook()	3032
7.1122.2.5NvmGetUserProgBaseAddress()	3032
7.1122.2.6NvmInit()	3033
7.1122.2.7NvmInitHook()	3033
7.1122.2.8NvmReinit()	3033
7.1122.2.9NvmReinitHook()	3033
7.1122.2.10NvmVerifyChecksum()	3034
7.1122.2.11NvmWrite()	3034
7.1122.2.12NvmWriteHook()	3034
7.1123Nvm.c File Reference	3035
7.1123.1Detailed Description	3036
7.1123.2Function Documentation	3036
7.1123.2.1NvmDone()	3036
7.1123.2.2NvmDoneHook()	3036
7.1123.2.3NvmErase()	3036
7.1123.2.4NvmEraseHook()	3037
7.1123.2.5NvmGetUserProgBaseAddress()	3037
7.1123.2.6NvmInit()	3038

7.1123.2.7NvmInitHook()	3038
7.1123.2.8NvmReinit()	3038
7.1123.2.9NvmReinitHook()	3038
7.1123.2.10NvmVerifyChecksum()	3039
7.1123.2.11NvmWrite()	3039
7.1123.2.12NvmWriteHook()	3039
7.1124Nvm.c File Reference	3040
7.1124.1Detailed Description	3041
7.1124.2Function Documentation	3041
7.1124.2.1NvmDone()	3041
7.1124.2.2NvmDoneHook()	3041
7.1124.2.3NvmErase()	3041
7.1124.2.4NvmEraseHook()	3042
7.1124.2.5NvmGetUserProgBaseAddress()	3042
7.1124.2.6NvmInit()	3043
7.1124.2.7NvmInitHook()	3043
7.1124.2.8NvmReinit()	3043
7.1124.2.9NvmReinitHook()	3043
7.1124.2.10NvmVerifyChecksum()	3044
7.1124.2.11NvmWrite()	3044
7.1124.2.12NvmWriteHook()	3044
7.1125Nvm.c File Reference	3045
7.1125.1Detailed Description	3046
7.1125.2Function Documentation	3046
7.1125.2.1NvmDone()	3046
7.1125.2.2NvmDoneHook()	3046
7.1125.2.3NvmErase()	3046
7.1125.2.4NvmEraseHook()	3047
7.1125.2.5NvmGetUserProgBaseAddress()	3047
7.1125.2.6NvmInit()	3048

7.1125.2.7NvmInitHook()	3048
7.1125.2.8NvmReinit()	3048
7.1125.2.9NvmReinitHook()	3048
7.1125.2.10NvmVerifyChecksum()	3049
7.1125.2.11NvmWrite()	3049
7.1125.2.12NvmWriteHook()	3049
7.1126Nvm.c File Reference	3050
7.1126.1Detailed Description	3051
7.1126.2Function Documentation	3051
7.1126.2.1NvmDone()	3051
7.1126.2.2NvmDoneHook()	3051
7.1126.2.3NvmErase()	3051
7.1126.2.4NvmEraseHook()	3052
7.1126.2.5NvmGetUserProgBaseAddress()	3052
7.1126.2.6NvmInit()	3053
7.1126.2.7NvmInitHook()	3053
7.1126.2.8NvmReinit()	3053
7.1126.2.9NvmReinitHook()	3053
7.1126.2.10NvmVerifyChecksum()	3054
7.1126.2.11NvmWrite()	3054
7.1126.2.12NvmWriteHook()	3054
7.1127Nvm.h File Reference	3055
7.1127.1Detailed Description	3055
7.1127.2Function Documentation	3056
7.1127.2.1NvmDone()	3056
7.1127.2.2NvmErase()	3056
7.1127.2.3NvmGetUserProgBaseAddress()	3056
7.1127.2.4NvmInit()	3057
7.1127.2.5NvmReinit()	3057
7.1127.2.6NvmVerifyChecksum()	3057

7.1127.2.7 NvmWrite()	3057
7.1128.1 plausibility.h File Reference	3058
7.1128.1 Detailed Description	3058
7.1128.2 232.c File Reference	3058
7.1129.1 Detailed Description	3059
7.1130.2 232.c File Reference	3059
7.1130.1 Detailed Description	3059
7.1131.2 232.c File Reference	3059
7.1131.1 Detailed Description	3060
7.1132.2 232.c File Reference	3060
7.1132.1 Detailed Description	3060
7.1133.2 232.c File Reference	3060
7.1133.1 Detailed Description	3061
7.1134.2 232.c File Reference	3061
7.1134.1 Detailed Description	3061
7.1135.2 232.c File Reference	3061
7.1135.1 Detailed Description	3062
7.1136.2 232.c File Reference	3062
7.1136.1 Detailed Description	3062
7.1137.2 232.c File Reference	3062
7.1137.1 Detailed Description	3063
7.1138.2 232.c File Reference	3063
7.1138.1 Detailed Description	3063
7.1139.2 232.c File Reference	3063
7.1139.1 Detailed Description	3064
7.1140.2 232.c File Reference	3064
7.1140.1 Detailed Description	3064
7.1141.2 232.c File Reference	3064
7.1141.1 Detailed Description	3065
7.1142.2 232.c File Reference	3065

7.1142.1 Detailed Description	3065
7.1142.232.c File Reference	3065
7.1143.1 Detailed Description	3066
7.1143.232.c File Reference	3066
7.1144.1 Detailed Description	3066
7.1144.232.c File Reference	3066
7.1145.1 Detailed Description	3067
7.1145.232.h File Reference	3067
7.1146.1 Detailed Description	3067
7.1146.232.h File Reference	3067
7.1147.1 Detailed Description	3068
7.1147.2 Function Documentation	3068
7.1147.2.1 __attribute__((aligned(1)))	3068
7.1147.2.2 SharedParamsCalculateChecksum()	3069
7.1147.2.3 SharedParamsInit()	3069
7.1147.2.4 SharedParamsReadByIndex()	3069
7.1147.2.5 SharedParamsValidateBuffer()	3070
7.1147.2.6 SharedParamsVerifyChecksum()	3070
7.1147.2.7 SharedParamsWriteByIndex()	3070
7.1147.2.8 SharedParamsWriteChecksum()	3071
7.1148.1 Detailed Description	3071
7.1148.232.c File Reference	3071
7.1148.2 Function Documentation	3072
7.1148.2.1 __attribute__((aligned(1)))	3073
7.1148.2.2 SharedParamsCalculateChecksum()	3073
7.1148.2.3 SharedParamsInit()	3074
7.1148.2.4 SharedParamsReadByIndex()	3074
7.1148.2.5 SharedParamsValidateBuffer()	3074
7.1148.2.6 SharedParamsVerifyChecksum()	3075
7.1148.2.7 SharedParamsWriteByIndex()	3075

7.1148.2.8SharedParamsWriteChecksum()	3075
7.1149Shared_params.c File Reference	3076
7.1149.1Detailed Description	3077
7.1149.2Function Documentation	3077
7.1149.2.1SharedParamsCalculateChecksum()	3077
7.1149.2.2SharedParamsInit()	3077
7.1149.2.3SharedParamsReadByIndex()	3077
7.1149.2.4SharedParamsValidateBuffer()	3078
7.1149.2.5SharedParamsVerifyChecksum()	3078
7.1149.2.6SharedParamsWriteByIndex()	3078
7.1149.2.7SharedParamsWriteChecksum()	3079
7.1149.3Variable Documentation	3079
7.1149.3.1shared	3079
7.1150Shared_params.c File Reference	3080
7.1150.1Detailed Description	3081
7.1150.2Function Documentation	3081
7.1150.2.1SharedParamsCalculateChecksum()	3081
7.1150.2.2SharedParamsInit()	3081
7.1150.2.3SharedParamsReadByIndex()	3081
7.1150.2.4SharedParamsValidateBuffer()	3082
7.1150.2.5SharedParamsVerifyChecksum()	3082
7.1150.2.6SharedParamsWriteByIndex()	3082
7.1150.2.7SharedParamsWriteChecksum()	3083
7.1150.3Variable Documentation	3083
7.1150.3.1shared	3083
7.1151Shared_params.c File Reference	3084
7.1151.1Detailed Description	3085
7.1151.2Function Documentation	3085
7.1151.2.1__attribute__()	3085
7.1151.2.2SharedParamsCalculateChecksum()	3086

7.1151.2.3SharedParamsInit()	3086
7.1151.2.4SharedParamsReadByIndex()	3086
7.1151.2.5SharedParamsValidateBuffer()	3087
7.1151.2.6SharedParamsVerifyChecksum()	3087
7.1151.2.7SharedParamsWriteByIndex()	3087
7.1151.2.8SharedParamsWriteChecksum()	3088
7.1152Shared_params.c File Reference	3088
7.1152.1Detailed Description	3089
7.1152.2Function Documentation	3089
7.1152.2.1__attribute__()	3089
7.1152.2.2SharedParamsCalculateChecksum()	3090
7.1152.2.3SharedParamsInit()	3090
7.1152.2.4SharedParamsReadByIndex()	3090
7.1152.2.5SharedParamsValidateBuffer()	3091
7.1152.2.6SharedParamsVerifyChecksum()	3091
7.1152.2.7SharedParamsWriteByIndex()	3091
7.1152.2.8SharedParamsWriteChecksum()	3092
7.1153Shared_params.c File Reference	3092
7.1153.1Detailed Description	3093
7.1153.2Function Documentation	3093
7.1153.2.1__attribute__()	3093
7.1153.2.2SharedParamsCalculateChecksum()	3094
7.1153.2.3SharedParamsInit()	3095
7.1153.2.4SharedParamsReadByIndex()	3095
7.1153.2.5SharedParamsValidateBuffer()	3095
7.1153.2.6SharedParamsVerifyChecksum()	3096
7.1153.2.7SharedParamsWriteByIndex()	3096
7.1153.2.8SharedParamsWriteChecksum()	3096
7.1154Shared_params.c File Reference	3097
7.1154.1Detailed Description	3098

7.1154.2Function Documentation	3098
7.1154.2.1__attribute__()	3098
7.1154.2.2SharedParamsCalculateChecksum()	3099
7.1154.2.3SharedParamsInit()	3099
7.1154.2.4SharedParamsReadByIndex()	3099
7.1154.2.5SharedParamsValidateBuffer()	3100
7.1154.2.6SharedParamsVerifyChecksum()	3100
7.1154.2.7SharedParamsWriteByIndex()	3100
7.1154.2.8SharedParamsWriteChecksum()	3101
7.1155shared_params.c File Reference	3101
7.1155.1Detailed Description	3102
7.1155.2Function Documentation	3102
7.1155.2.1SharedParamsCalculateChecksum()	3102
7.1155.2.2SharedParamsInit()	3103
7.1155.2.3SharedParamsReadByIndex()	3103
7.1155.2.4SharedParamsValidateBuffer()	3103
7.1155.2.5SharedParamsVerifyChecksum()	3104
7.1155.2.6SharedParamsWriteByIndex()	3104
7.1155.2.7SharedParamsWriteChecksum()	3104
7.1155.3Variable Documentation	3105
7.1155.3.1shared	3105
7.1156shared_params.c File Reference	3105
7.1156.1Detailed Description	3107
7.1156.2Function Documentation	3107
7.1156.2.1SharedParamsCalculateChecksum()	3107
7.1156.2.2SharedParamsInit()	3107
7.1156.2.3SharedParamsReadByIndex()	3107
7.1156.2.4SharedParamsValidateBuffer()	3108
7.1156.2.5SharedParamsVerifyChecksum()	3108
7.1156.2.6SharedParamsWriteByIndex()	3108

7.1156.2.7SharedParamsWriteChecksum()	3109
7.1156.3/variable Documentation	3109
7.1156.3.1shared	3109
7.1157shared_params.c File Reference	3110
7.1157.1Detailed Description	3111
7.1157.2Function Documentation	3111
7.1157.2.1__attribute__()	3111
7.1157.2.2SharedParamsCalculateChecksum()	3112
7.1157.2.3SharedParamsInit()	3112
7.1157.2.4SharedParamsReadByIndex()	3112
7.1157.2.5SharedParamsValidateBuffer()	3113
7.1157.2.6SharedParamsVerifyChecksum()	3113
7.1157.2.7SharedParamsWriteByIndex()	3113
7.1157.2.8SharedParamsWriteChecksum()	3114
7.1158shared_params.c File Reference	3114
7.1158.1Detailed Description	3115
7.1158.2Function Documentation	3115
7.1158.2.1__attribute__()	3115
7.1158.2.2SharedParamsCalculateChecksum()	3116
7.1158.2.3SharedParamsInit()	3116
7.1158.2.4SharedParamsReadByIndex()	3116
7.1158.2.5SharedParamsValidateBuffer()	3117
7.1158.2.6SharedParamsVerifyChecksum()	3117
7.1158.2.7SharedParamsWriteByIndex()	3117
7.1158.2.8SharedParamsWriteChecksum()	3118
7.1159shared_params.c File Reference	3118
7.1159.1Detailed Description	3119
7.1159.2Function Documentation	3119
7.1159.2.1__attribute__()	3120
7.1159.2.2SharedParamsCalculateChecksum()	3120

7.1159.2.3SharedParamsInit()	3121
7.1159.2.4SharedParamsReadByIndex()	3121
7.1159.2.5SharedParamsValidateBuffer()	3121
7.1159.2.6SharedParamsVerifyChecksum()	3122
7.1159.2.7SharedParamsWriteByIndex()	3122
7.1159.2.8SharedParamsWriteChecksum()	3122
7.1160Shared_params.c File Reference	3123
7.1160.1Detailed Description	3124
7.1160.2Function Documentation	3124
7.1160.2.1__attribute__()	3124
7.1160.2.2SharedParamsCalculateChecksum()	3125
7.1160.2.3SharedParamsInit()	3125
7.1160.2.4SharedParamsReadByIndex()	3125
7.1160.2.5SharedParamsValidateBuffer()	3126
7.1160.2.6SharedParamsVerifyChecksum()	3126
7.1160.2.7SharedParamsWriteByIndex()	3126
7.1160.2.8SharedParamsWriteChecksum()	3127
7.1161Shared_params.c File Reference	3127
7.1161.1Detailed Description	3128
7.1161.2Function Documentation	3128
7.1161.2.1SharedParamsCalculateChecksum()	3128
7.1161.2.2SharedParamsInit()	3129
7.1161.2.3SharedParamsReadByIndex()	3129
7.1161.2.4SharedParamsValidateBuffer()	3129
7.1161.2.5SharedParamsVerifyChecksum()	3130
7.1161.2.6SharedParamsWriteByIndex()	3130
7.1161.2.7SharedParamsWriteChecksum()	3130
7.1161.3Variable Documentation	3131
7.1161.3.1shared	3131
7.1162Shared_params.c File Reference	3131

7.1162.1Detailed Description	3133
7.1162.2Function Documentation	3133
7.1162.2.1SharedParamsCalculateChecksum()	3133
7.1162.2.2SharedParamsInit()	3133
7.1162.2.3SharedParamsReadByIndex()	3133
7.1162.2.4SharedParamsValidateBuffer()	3134
7.1162.2.5SharedParamsVerifyChecksum()	3134
7.1162.2.6SharedParamsWriteByIndex()	3134
7.1162.2.7SharedParamsWriteChecksum()	3135
7.1162.3Variable Documentation	3135
7.1162.3.1shared	3135
7.1163shared_params.c File Reference	3136
7.1163.1Detailed Description	3137
7.1163.2Function Documentation	3137
7.1163.2.1__attribute__()	3137
7.1163.2.2SharedParamsCalculateChecksum()	3138
7.1163.2.3SharedParamsInit()	3138
7.1163.2.4SharedParamsReadByIndex()	3138
7.1163.2.5SharedParamsValidateBuffer()	3139
7.1163.2.6SharedParamsVerifyChecksum()	3139
7.1163.2.7SharedParamsWriteByIndex()	3139
7.1163.2.8SharedParamsWriteChecksum()	3140
7.1164shared_params.c File Reference	3140
7.1164.1Detailed Description	3141
7.1164.2Function Documentation	3141
7.1164.2.1__attribute__()	3141
7.1164.2.2SharedParamsCalculateChecksum()	3142
7.1164.2.3SharedParamsInit()	3142
7.1164.2.4SharedParamsReadByIndex()	3142
7.1164.2.5SharedParamsValidateBuffer()	3143

7.1164.2.6SharedParamsVerifyChecksum()	3143
7.1164.2.7SharedParamsWriteByIndex()	3143
7.1164.2.8SharedParamsWriteChecksum()	3144
7.1165Shared_params.c File Reference	3144
7.1165.1Detailed Description	3145
7.1165.2Function Documentation	3145
7.1165.2.1__attribute__()	3146
7.1165.2.2SharedParamsCalculateChecksum()	3146
7.1165.2.3SharedParamsInit()	3147
7.1165.2.4SharedParamsReadByIndex()	3147
7.1165.2.5SharedParamsValidateBuffer()	3147
7.1165.2.6SharedParamsVerifyChecksum()	3148
7.1165.2.7SharedParamsWriteByIndex()	3148
7.1165.2.8SharedParamsWriteChecksum()	3148
7.1166Shared_params.c File Reference	3149
7.1166.1Detailed Description	3150
7.1166.2Function Documentation	3150
7.1166.2.1__attribute__()	3150
7.1166.2.2SharedParamsCalculateChecksum()	3151
7.1166.2.3SharedParamsInit()	3151
7.1166.2.4SharedParamsReadByIndex()	3151
7.1166.2.5SharedParamsValidateBuffer()	3152
7.1166.2.6SharedParamsVerifyChecksum()	3152
7.1166.2.7SharedParamsWriteByIndex()	3152
7.1166.2.8SharedParamsWriteChecksum()	3153
7.1167Shared_params.c File Reference	3153
7.1167.1Detailed Description	3154
7.1167.2Function Documentation	3154
7.1167.2.1SharedParamsCalculateChecksum()	3154
7.1167.2.2SharedParamsInit()	3155

7.1167.2.3SharedParamsReadByIndex()	3155
7.1167.2.4SharedParamsValidateBuffer()	3155
7.1167.2.5SharedParamsVerifyChecksum()	3156
7.1167.2.6SharedParamsWriteByIndex()	3156
7.1167.2.7SharedParamsWriteChecksum()	3156
7.1167.3Variable Documentation	3157
7.1167.3.1shared	3157
7.1168shared_params.c File Reference	3157
7.1168.1Detailed Description	3159
7.1168.2Function Documentation	3159
7.1168.2.1SharedParamsCalculateChecksum()	3159
7.1168.2.2SharedParamsInit()	3159
7.1168.2.3SharedParamsReadByIndex()	3159
7.1168.2.4SharedParamsValidateBuffer()	3160
7.1168.2.5SharedParamsVerifyChecksum()	3160
7.1168.2.6SharedParamsWriteByIndex()	3160
7.1168.2.7SharedParamsWriteChecksum()	3161
7.1168.3Variable Documentation	3161
7.1168.3.1shared	3161
7.1169shared_params.c File Reference	3162
7.1169.1Detailed Description	3163
7.1169.2Function Documentation	3163
7.1169.2.1__attribute__()	3163
7.1169.2.2SharedParamsCalculateChecksum()	3164
7.1169.2.3SharedParamsInit()	3164
7.1169.2.4SharedParamsReadByIndex()	3164
7.1169.2.5SharedParamsValidateBuffer()	3165
7.1169.2.6SharedParamsVerifyChecksum()	3165
7.1169.2.7SharedParamsWriteByIndex()	3165
7.1169.2.8SharedParamsWriteChecksum()	3166

7.1170	shared_params.c File Reference	3166
7.1170.1	Detailed Description	3167
7.1170.2	Function Documentation	3167
7.1170.2.1	__attribute__((aligned(1)))	3167
7.1170.2.2	shared_params_calculate_checksum()	3168
7.1170.2.3	shared_params_init()	3168
7.1170.2.4	shared_params_read_by_index()	3168
7.1170.2.5	shared_params_validate_buffer()	3169
7.1170.2.6	shared_params_verify_checksum()	3169
7.1170.2.7	shared_params_write_by_index()	3169
7.1170.2.8	shared_params_write_checksum()	3170
7.1171	shared_params.h File Reference	3170
7.1171.1	Detailed Description	3171
7.1171.2	Function Documentation	3171
7.1171.2.1	shared_params_init()	3171
7.1171.2.2	shared_params_read_by_index()	3172
7.1171.2.3	shared_params_write_by_index()	3172
7.1172	shared_params.h File Reference	3172
7.1172.1	Detailed Description	3173
7.1172.2	Function Documentation	3174
7.1172.2.1	shared_params_init()	3174
7.1172.2.2	shared_params_read_by_index()	3174
7.1172.2.3	shared_params_write_by_index()	3174
7.1173	shared_params.h File Reference	3175
7.1173.1	Detailed Description	3176
7.1173.2	Function Documentation	3176
7.1173.2.1	shared_params_init()	3176
7.1173.2.2	shared_params_read_by_index()	3176
7.1173.2.3	shared_params_write_by_index()	3177
7.1174	shared_params.h File Reference	3177

7.1174.1Detailed Description	3178
7.1174.2Function Documentation	3178
7.1174.2.1SharedParamsInit()	3178
7.1174.2.2SharedParamsReadByIndex()	3179
7.1174.2.3SharedParamsWriteByIndex()	3179
7.1175shared_params.h File Reference	3179
7.1175.1Detailed Description	3180
7.1175.2Function Documentation	3181
7.1175.2.1SharedParamsInit()	3181
7.1175.2.2SharedParamsReadByIndex()	3181
7.1175.2.3SharedParamsWriteByIndex()	3181
7.1176shared_params.h File Reference	3182
7.1176.1Detailed Description	3183
7.1176.2Function Documentation	3183
7.1176.2.1SharedParamsInit()	3183
7.1176.2.2SharedParamsReadByIndex()	3183
7.1176.2.3SharedParamsWriteByIndex()	3184
7.1177shared_params.h File Reference	3184
7.1177.1Detailed Description	3185
7.1177.2Function Documentation	3185
7.1177.2.1SharedParamsInit()	3185
7.1177.2.2SharedParamsReadByIndex()	3186
7.1177.2.3SharedParamsWriteByIndex()	3186
7.1178shared_params.h File Reference	3186
7.1178.1Detailed Description	3188
7.1178.2Function Documentation	3188
7.1178.2.1SharedParamsInit()	3188
7.1178.2.2SharedParamsReadByIndex()	3188
7.1178.2.3SharedParamsWriteByIndex()	3188
7.1179shared_params.h File Reference	3189

7.1179.1Detailed Description	3190
7.1179.2Function Documentation	3190
7.1179.2.1SharedParamsInit()	3190
7.1179.2.2SharedParamsReadByIndex()	3190
7.1179.2.3SharedParamsWriteByIndex()	3191
7.1180Shared_params.h File Reference	3191
7.1180.1Detailed Description	3192
7.1180.2Function Documentation	3192
7.1180.2.1SharedParamsInit()	3192
7.1180.2.2SharedParamsReadByIndex()	3193
7.1180.2.3SharedParamsWriteByIndex()	3193
7.1181Shared_params.h File Reference	3193
7.1181.1Detailed Description	3195
7.1181.2Function Documentation	3195
7.1181.2.1SharedParamsInit()	3195
7.1181.2.2SharedParamsReadByIndex()	3195
7.1181.2.3SharedParamsWriteByIndex()	3195
7.1182Shared_params.h File Reference	3196
7.1182.1Detailed Description	3197
7.1182.2Function Documentation	3197
7.1182.2.1SharedParamsInit()	3197
7.1182.2.2SharedParamsReadByIndex()	3197
7.1182.2.3SharedParamsWriteByIndex()	3198
7.1183Shared_params.h File Reference	3198
7.1183.1Detailed Description	3199
7.1183.2Function Documentation	3199
7.1183.2.1SharedParamsInit()	3199
7.1183.2.2SharedParamsReadByIndex()	3200
7.1183.2.3SharedParamsWriteByIndex()	3200
7.1184Shared_params.h File Reference	3200

7.1184.1Detailed Description	3201
7.1184.2Function Documentation	3202
7.1184.2.1SharedParamsInit()	3202
7.1184.2.2SharedParamsReadByIndex()	3202
7.1184.2.3SharedParamsWriteByIndex()	3202
7.1185shared_params.h File Reference	3203
7.1185.1Detailed Description	3204
7.1185.2Function Documentation	3204
7.1185.2.1SharedParamsInit()	3204
7.1185.2.2SharedParamsReadByIndex()	3204
7.1185.2.3SharedParamsWriteByIndex()	3205
7.1186shared_params.h File Reference	3205
7.1186.1Detailed Description	3206
7.1186.2Function Documentation	3206
7.1186.2.1SharedParamsInit()	3206
7.1186.2.2SharedParamsReadByIndex()	3207
7.1186.2.3SharedParamsWriteByIndex()	3207
7.1187shared_params.h File Reference	3207
7.1187.1Detailed Description	3209
7.1187.2Function Documentation	3209
7.1187.2.1SharedParamsInit()	3209
7.1187.2.2SharedParamsReadByIndex()	3209
7.1187.2.3SharedParamsWriteByIndex()	3209
7.1188shared_params.h File Reference	3210
7.1188.1Detailed Description	3211
7.1188.2Function Documentation	3211
7.1188.2.1SharedParamsInit()	3211
7.1188.2.2SharedParamsReadByIndex()	3211
7.1188.2.3SharedParamsWriteByIndex()	3212
7.1189shared_params.h File Reference	3212

7.1189.1 Detailed Description	3213
7.1189.2 Function Documentation	3213
7.1189.2.1 SharedParamsInit()	3213
7.1189.2.2 SharedParamsReadByIndex()	3214
7.1189.2.3 SharedParamsWriteByIndex()	3214
7.1190 shared_params.h File Reference	3214
7.1190.1 Detailed Description	3215
7.1190.2 Function Documentation	3216
7.1190.2.1 SharedParamsInit()	3216
7.1190.2.2 SharedParamsReadByIndex()	3216
7.1190.2.3 SharedParamsWriteByIndex()	3216
7.1191 shared_params.h File Reference	3217
7.1191.1 Detailed Description	3218
7.1191.2 Function Documentation	3218
7.1191.2.1 SharedParamsInit()	3218
7.1191.2.2 SharedParamsReadByIndex()	3218
7.1191.2.3 SharedParamsWriteByIndex()	3219
7.1192 me.c File Reference	3219
7.1192.1 Detailed Description	3220
7.1192.2 Function Documentation	3220
7.1192.2.1 TimerDeinit()	3220
7.1192.2.2 TimerGet()	3220
7.1192.2.3 TimerInit()	3220
7.1192.2.4 TimerISRHandler()	3221
7.1192.2.5 TimerSet()	3221
7.1193 me.c File Reference	3221
7.1193.1 Detailed Description	3222
7.1193.2 Function Documentation	3222
7.1193.2.1 TimerDeinit()	3222
7.1193.2.2 TimerGet()	3222

7.1193.2.3TimerInit()	3223
7.1193.2.4TimerISRHandler()	3223
7.1193.2.5TimerSet()	3223
7.1194me.c File Reference	3223
7.1194.1Detailed Description	3224
7.1194.2Function Documentation	3224
7.1194.2.1TimerDeinit()	3224
7.1194.2.2TimerGet()	3225
7.1194.2.3TimerInit()	3225
7.1194.2.4TimerISRHandler()	3225
7.1194.2.5TimerSet()	3225
7.1195me.c File Reference	3226
7.1195.1Detailed Description	3226
7.1195.2Function Documentation	3226
7.1195.2.1TimerDeinit()	3227
7.1195.2.2TimerGet()	3227
7.1195.2.3TimerInit()	3227
7.1195.2.4TimerISRHandler()	3227
7.1195.2.5TimerSet()	3227
7.1196me.c File Reference	3228
7.1196.1Detailed Description	3228
7.1196.2Function Documentation	3229
7.1196.2.1TimerDeinit()	3229
7.1196.2.2TimerGet()	3229
7.1196.2.3TimerInit()	3229
7.1196.2.4TimerISRHandler()	3229
7.1196.2.5TimerSet()	3229
7.1197me.h File Reference	3230
7.1197.1Detailed Description	3230
7.1197.2Function Documentation	3231

7.1197.2.1TimerDeinit()	3231
7.1197.2.2TimerGet()	3231
7.1197.2.3TimerInit()	3231
7.1197.2.4TimerISRHandler()	3232
7.1197.2.5TimerSet()	3232
7.1198me.h File Reference	3232
7.1198.1Detailed Description	3233
7.1198.2Function Documentation	3233
7.1198.2.1TimerDeinit()	3233
7.1198.2.2TimerGet()	3233
7.1198.2.3TimerInit()	3234
7.1198.2.4TimerISRHandler()	3234
7.1198.2.5TimerSet()	3234
7.1199me.h File Reference	3235
7.1199.1Detailed Description	3235
7.1199.2Function Documentation	3235
7.1199.2.1TimerDeinit()	3235
7.1199.2.2TimerGet()	3236
7.1199.2.3TimerInit()	3236
7.1199.2.4TimerISRHandler()	3236
7.1199.2.5TimerSet()	3236
7.1200me.h File Reference	3237
7.1200.1Detailed Description	3237
7.1200.2Function Documentation	3238
7.1200.2.1TimerDeinit()	3238
7.1200.2.2TimerGet()	3238
7.1200.2.3TimerInit()	3238
7.1200.2.4TimerISRHandler()	3239
7.1200.2.5TimerSet()	3239
7.1201me.h File Reference	3239

7.1201.1 Detailed Description	3240
7.1201.2 Function Documentation	3240
7.1201.2.1 TimerDeinit()	3240
7.1201.2.2 TimerGet()	3240
7.1201.2.3 TimerInit()	3241
7.1201.2.4 TimerISRHandler()	3241
7.1201.2.5 TimerSet()	3241
7.1202 mer.c File Reference	3242
7.1202.1 Detailed Description	3242
7.1202.2 Function Documentation	3242
7.1202.2.1 TimerGet()	3243
7.1202.2.2 TimerInit()	3243
7.1202.2.3 TimerInterrupt()	3243
7.1203 mer.c File Reference	3244
7.1203.1 Detailed Description	3244
7.1203.2 Function Documentation	3244
7.1203.2.1 TimerGet()	3244
7.1203.2.2 TimerInit()	3245
7.1204 mer.c File Reference	3245
7.1204.1 Detailed Description	3245
7.1204.2 Function Documentation	3245
7.1204.2.1 SysTick_Handler()	3246
7.1204.2.2 TimerGet()	3246
7.1204.2.3 TimerInit()	3246
7.1205 mer.c File Reference	3246
7.1205.1 Detailed Description	3247
7.1205.2 Function Documentation	3247
7.1205.2.1 SysTick_Handler()	3247
7.1205.2.2 TimerGet()	3247
7.1205.2.3 TimerInit()	3248

7.1206mer.c File Reference	3248
7.1206.1Detailed Description	3248
7.1206.2Function Documentation	3248
7.1206.2.1SysTick_Handler()	3249
7.1206.2.2TimerGet()	3249
7.1206.2.3TimerInit()	3249
7.1207mer.c File Reference	3249
7.1207.1Detailed Description	3250
7.1207.2Function Documentation	3250
7.1207.2.1TimerGet()	3250
7.1207.2.2TimerInit()	3250
7.1208mer.c File Reference	3251
7.1208.1Detailed Description	3251
7.1208.2Function Documentation	3251
7.1208.2.1SysTick_Handler()	3251
7.1208.2.2TimerGet()	3252
7.1208.2.3TimerInit()	3252
7.1209mer.c File Reference	3252
7.1209.1Detailed Description	3253
7.1209.2Function Documentation	3253
7.1209.2.1SysTick_Handler()	3253
7.1209.2.2TimerGet()	3253
7.1209.2.3TimerInit()	3253
7.1210mer.c File Reference	3254
7.1210.1Detailed Description	3254
7.1210.2Function Documentation	3254
7.1210.2.1SysTick_Handler()	3254
7.1210.2.2TimerGet()	3255
7.1210.2.3TimerInit()	3255
7.1211mer.c File Reference	3255

7.1211.1Detailed Description	3256
7.1211.2Function Documentation	3256
7.1211.2.1TimerGet()	3256
7.1211.2.2TimerInit()	3256
7.1212mer.c File Reference	3256
7.1212.1Detailed Description	3257
7.1212.2Function Documentation	3257
7.1212.2.1SysTick_Handler()	3257
7.1212.2.2TimerGet()	3257
7.1212.2.3TimerInit()	3258
7.1213mer.c File Reference	3258
7.1213.1Detailed Description	3258
7.1213.2Function Documentation	3258
7.1213.2.1SysTick_Handler()	3259
7.1213.2.2TimerGet()	3259
7.1213.2.3TimerInit()	3259
7.1214mer.c File Reference	3259
7.1214.1Detailed Description	3260
7.1214.2Function Documentation	3260
7.1214.2.1SysTick_Handler()	3260
7.1214.2.2TimerGet()	3260
7.1214.2.3TimerInit()	3261
7.1215mer.c File Reference	3261
7.1215.1Detailed Description	3262
7.1215.2Function Documentation	3262
7.1215.2.1SysTick_Handler()	3262
7.1215.2.2TimerDeinit()	3262
7.1215.2.3TimerGet()	3262
7.1215.2.4TimerInit()	3263
7.1215.2.5TimerSet()	3263

7.1216mer.c File Reference	3263
7.1216.1Detailed Description	3264
7.1216.2Function Documentation	3264
7.1216.2.1SysTick_Handler()	3264
7.1216.2.2TimerDeinit()	3264
7.1216.2.3TimerGet()	3265
7.1216.2.4TimerInit()	3265
7.1216.2.5TimerSet()	3265
7.1217mer.c File Reference	3265
7.1217.1Detailed Description	3266
7.1217.2Function Documentation	3266
7.1217.2.1TimerGet()	3266
7.1217.2.2TimerInit()	3267
7.1218mer.c File Reference	3267
7.1218.1Detailed Description	3267
7.1218.2Function Documentation	3267
7.1218.2.1SysTick_Handler()	3268
7.1218.2.2TimerGet()	3268
7.1218.2.3TimerInit()	3268
7.1219mer.c File Reference	3268
7.1219.1Detailed Description	3269
7.1219.2Function Documentation	3269
7.1219.2.1SysTick_Handler()	3269
7.1219.2.2TimerGet()	3269
7.1219.2.3TimerInit()	3270
7.1220mer.c File Reference	3270
7.1220.1Detailed Description	3270
7.1220.2Function Documentation	3270
7.1220.2.1SysTick_Handler()	3271
7.1220.2.2TimerGet()	3271

7.1220.2.3TimerInit()	3271
7.1221mer.c File Reference	3271
7.1221.1Detailed Description	3272
7.1221.2Function Documentation	3272
7.1221.2.1TimerDeinit()	3272
7.1221.2.2TimerGet()	3272
7.1221.2.3TimerInit()	3273
7.1221.2.4TimerISRHandler()	3273
7.1221.2.5TimerSet()	3273
7.1222mer.c File Reference	3273
7.1222.1Detailed Description	3274
7.1222.2Function Documentation	3274
7.1222.2.1TimerDeinit()	3274
7.1222.2.2TimerGet()	3275
7.1222.2.3TimerInit()	3275
7.1222.2.4TimerISRHandler()	3275
7.1222.2.5TimerSet()	3275
7.1223mer.c File Reference	3276
7.1223.1Detailed Description	3276
7.1223.2Function Documentation	3276
7.1223.2.1TimerGet()	3276
7.1223.2.2TimerInit()	3277
7.1224mer.c File Reference	3277
7.1224.1Detailed Description	3277
7.1224.2Function Documentation	3277
7.1224.2.1SysTick_Handler()	3278
7.1224.2.2TimerGet()	3278
7.1224.2.3TimerInit()	3278
7.1225mer.c File Reference	3278
7.1225.1Detailed Description	3279

7.1225.2Function Documentation	3279
7.1225.2.1SysTick_Handler()	3279
7.1225.2.2TimerGet()	3279
7.1225.2.3TimerInit()	3280
7.1226mer.c File Reference	3280
7.1226.1Detailed Description	3280
7.1226.2Function Documentation	3280
7.1226.2.1SysTick_Handler()	3281
7.1226.2.2TimerGet()	3281
7.1226.2.3TimerInit()	3281
7.1227mer.c File Reference	3282
7.1227.1Detailed Description	3282
7.1227.2Function Documentation	3282
7.1227.2.1TimerGet()	3282
7.1227.2.2TimerInit()	3283
7.1228mer.c File Reference	3283
7.1228.1Detailed Description	3283
7.1228.2Function Documentation	3284
7.1228.2.1SysTick_Handler()	3284
7.1228.2.2TimerGet()	3284
7.1228.2.3TimerInit()	3284
7.1229mer.c File Reference	3285
7.1229.1Detailed Description	3285
7.1229.2Function Documentation	3285
7.1229.2.1SysTick_Handler()	3285
7.1229.2.2TimerGet()	3286
7.1229.2.3TimerInit()	3286
7.1230mer.c File Reference	3286
7.1230.1Detailed Description	3287
7.1230.2Function Documentation	3287

7.1230.2.1SysTick_Handler()	3287
7.1230.2.2TimerGet()	3287
7.1230.2.3TimerInit()	3288
7.1231mer.c File Reference	3288
7.1231.1Detailed Description	3288
7.1231.2Function Documentation	3288
7.1231.2.1TimerGet()	3289
7.1231.2.2TimerInit()	3289
7.1232mer.c File Reference	3289
7.1232.1Detailed Description	3290
7.1232.2Function Documentation	3290
7.1232.2.1SysTick_Handler()	3290
7.1232.2.2TimerGet()	3290
7.1232.2.3TimerInit()	3290
7.1233mer.c File Reference	3291
7.1233.1Detailed Description	3291
7.1233.2Function Documentation	3291
7.1233.2.1SysTick_Handler()	3291
7.1233.2.2TimerGet()	3292
7.1233.2.3TimerInit()	3292
7.1234mer.c File Reference	3292
7.1234.1Detailed Description	3293
7.1234.2Function Documentation	3293
7.1234.2.1SysTick_Handler()	3293
7.1234.2.2TimerGet()	3293
7.1234.2.3TimerInit()	3293
7.1235mer.c File Reference	3294
7.1235.1Detailed Description	3294
7.1235.2Function Documentation	3294
7.1235.2.1TimerGet()	3294

7.1235.2.2TimerInit()	3295
7.1236mer.c File Reference	3295
7.1236.1Detailed Description	3295
7.1236.2Function Documentation	3295
7.1236.2.1SysTick_Handler()	3296
7.1236.2.2TimerGet()	3296
7.1236.2.3TimerInit()	3296
7.1237mer.c File Reference	3296
7.1237.1Detailed Description	3297
7.1237.2Function Documentation	3297
7.1237.2.1SysTick_Handler()	3297
7.1237.2.2TimerGet()	3297
7.1237.2.3TimerInit()	3298
7.1238mer.c File Reference	3298
7.1238.1Detailed Description	3298
7.1238.2Function Documentation	3298
7.1238.2.1SysTick_Handler()	3299
7.1238.2.2TimerGet()	3299
7.1238.2.3TimerInit()	3299
7.1239mer.c File Reference	3299
7.1239.1Detailed Description	3300
7.1239.2Function Documentation	3300
7.1239.2.1TimerGet()	3300
7.1239.2.2TimerInit()	3300
7.1240mer.c File Reference	3301
7.1240.1Detailed Description	3301
7.1240.2Function Documentation	3301
7.1240.2.1SysTick_Handler()	3301
7.1240.2.2TimerGet()	3302
7.1240.2.3TimerInit()	3302

7.1241mer.c File Reference	3302
7.1241.1Detailed Description	3303
7.1241.2Function Documentation	3303
7.1241.2.1SysTick_Handler()	3303
7.1241.2.2TimerGet()	3303
7.1241.2.3TimerInit()	3303
7.1242mer.c File Reference	3304
7.1242.1Detailed Description	3304
7.1242.2Function Documentation	3304
7.1242.2.1SysTick_Handler()	3304
7.1242.2.2TimerGet()	3305
7.1242.2.3TimerInit()	3305
7.1243mer.c File Reference	3305
7.1243.1Detailed Description	3306
7.1243.2Function Documentation	3306
7.1243.2.1SysTick_Handler()	3306
7.1243.2.2TimerGet()	3306
7.1243.2.3TimerInit()	3306
7.1244mer.c File Reference	3307
7.1244.1Detailed Description	3307
7.1244.2Function Documentation	3307
7.1244.2.1SysTick_Handler()	3307
7.1244.2.2TimerGet()	3308
7.1244.2.3TimerInit()	3308
7.1245mer.c File Reference	3308
7.1245.1Detailed Description	3309
7.1245.2Function Documentation	3309
7.1245.2.1TimerGet()	3309
7.1245.2.2TimerInit()	3309
7.1246mer.c File Reference	3310

7.1246.1Detailed Description	3310
7.1246.2Function Documentation	3310
7.1246.2.1SysTick_Handler()	3310
7.1246.2.2TimerGet()	3311
7.1246.2.3TimerInit()	3311
7.1247timer.c File Reference	3311
7.1247.1Detailed Description	3312
7.1247.2Function Documentation	3312
7.1247.2.1SysTick_Handler()	3312
7.1247.2.2TimerGet()	3312
7.1247.2.3TimerInit()	3313
7.1248timer.c File Reference	3313
7.1248.1Detailed Description	3313
7.1248.2Function Documentation	3314
7.1248.2.1SysTick_Handler()	3314
7.1248.2.2TimerGet()	3314
7.1248.2.3TimerInit()	3314
7.1249timer.c File Reference	3315
7.1249.1Detailed Description	3315
7.1249.2Function Documentation	3315
7.1249.2.1TimerGet()	3315
7.1249.2.2TimerInit()	3316
7.1250timer.c File Reference	3316
7.1250.1Detailed Description	3316
7.1250.2Function Documentation	3316
7.1250.2.1SysTick_Handler()	3317
7.1250.2.2TimerGet()	3317
7.1250.2.3TimerInit()	3317
7.1251timer.c File Reference	3317
7.1251.1Detailed Description	3318

7.1251.2	Function Documentation	3318
7.1251.2.1	SysTick_Handler()	3318
7.1251.2.2	TimerGet()	3318
7.1251.2.3	TimerInit()	3319
7.1252	mer.c File Reference	3319
7.1252.1	Detailed Description	3319
7.1252.2	Function Documentation	3319
7.1252.2.1	SysTick_Handler()	3320
7.1252.2.2	TimerGet()	3320
7.1252.2.3	TimerInit()	3320
7.1253	mer.c File Reference	3320
7.1253.1	Detailed Description	3321
7.1253.2	Function Documentation	3321
7.1253.2.1	TimerGet()	3321
7.1253.2.2	TimerInit()	3321
7.1254	mer.c File Reference	3322
7.1254.1	Detailed Description	3322
7.1254.2	Function Documentation	3322
7.1254.2.1	SysTick_Handler()	3322
7.1254.2.2	TimerGet()	3323
7.1254.2.3	TimerInit()	3323
7.1255	mer.c File Reference	3323
7.1255.1	Detailed Description	3324
7.1255.2	Function Documentation	3324
7.1255.2.1	SysTick_Handler()	3324
7.1255.2.2	TimerGet()	3324
7.1255.2.3	TimerInit()	3324
7.1256	mer.c File Reference	3325
7.1256.1	Detailed Description	3325
7.1256.2	Function Documentation	3325

7.1256.2.1SysTick_Handler()	3325
7.1256.2.2TimerGet()	3326
7.1256.2.3TimerInit()	3326
7.1257mer.c File Reference	3326
7.1257.1Detailed Description	3327
7.1257.2Function Documentation	3327
7.1257.2.1TimerGet()	3327
7.1257.2.2TimerInit()	3327
7.1258mer.c File Reference	3327
7.1258.1Detailed Description	3328
7.1258.2Function Documentation	3328
7.1258.2.1SysTick_Handler()	3328
7.1258.2.2TimerGet()	3328
7.1258.2.3TimerInit()	3329
7.1259mer.c File Reference	3329
7.1259.1Detailed Description	3329
7.1259.2Function Documentation	3329
7.1259.2.1SysTick_Handler()	3330
7.1259.2.2TimerGet()	3330
7.1259.2.3TimerInit()	3330
7.1260mer.c File Reference	3330
7.1260.1Detailed Description	3331
7.1260.2Function Documentation	3331
7.1260.2.1SysTick_Handler()	3331
7.1260.2.2TimerGet()	3331
7.1260.2.3TimerInit()	3332
7.1261mer.c File Reference	3332
7.1261.1Detailed Description	3332
7.1261.2Function Documentation	3332
7.1261.2.1TimerGet()	3333

7.1261.2.2TimerInit()	3333
7.1262mer.c File Reference	3333
7.1262.1Detailed Description	3334
7.1262.2Function Documentation	3334
7.1262.2.1SysTick_Handler()	3334
7.1262.2.2TimerGet()	3334
7.1262.2.3TimerInit()	3334
7.1263mer.c File Reference	3335
7.1263.1Detailed Description	3335
7.1263.2Function Documentation	3335
7.1263.2.1SysTick_Handler()	3335
7.1263.2.2TimerGet()	3336
7.1263.2.3TimerInit()	3336
7.1264mer.c File Reference	3336
7.1264.1Detailed Description	3337
7.1264.2Function Documentation	3337
7.1264.2.1SysTick_Handler()	3337
7.1264.2.2TimerGet()	3337
7.1264.2.3TimerInit()	3337
7.1265mer.c File Reference	3338
7.1265.1Detailed Description	3338
7.1265.2Function Documentation	3338
7.1265.2.1SysTick_Handler()	3339
7.1265.2.2TimerDeinit()	3339
7.1265.2.3TimerGet()	3339
7.1265.2.4TimerInit()	3339
7.1265.2.5TimerSet()	3339
7.1266mer.c File Reference	3340
7.1266.1Detailed Description	3341
7.1266.2Function Documentation	3341

7.1266.2.1SysTick_Handler()	3341
7.1266.2.2TimerDeinit()	3341
7.1266.2.3TimerGet()	3341
7.1266.2.4TimerInit()	3342
7.1266.2.5TimerSet()	3342
7.1267timer.c File Reference	3342
7.1267.1Detailed Description	3343
7.1267.2Function Documentation	3343
7.1267.2.1TimerGet()	3343
7.1267.2.2TimerInit()	3343
7.1268timer.c File Reference	3344
7.1268.1Detailed Description	3344
7.1268.2Function Documentation	3344
7.1268.2.1SysTick_Handler()	3344
7.1268.2.2TimerGet()	3345
7.1268.2.3TimerInit()	3345
7.1269timer.c File Reference	3345
7.1269.1Detailed Description	3346
7.1269.2Function Documentation	3346
7.1269.2.1SysTick_Handler()	3346
7.1269.2.2TimerGet()	3346
7.1269.2.3TimerInit()	3346
7.1270timer.c File Reference	3347
7.1270.1Detailed Description	3347
7.1270.2Function Documentation	3347
7.1270.2.1SysTick_Handler()	3347
7.1270.2.2TimerGet()	3348
7.1270.2.3TimerInit()	3348
7.1271timer.c File Reference	3348
7.1271.1Detailed Description	3349

7.1271.2	Function Documentation	3349
7.1271.2.1	TimerGet()	3349
7.1271.2.2	TimerInit()	3349
7.1272	mer.c File Reference	3349
7.1272.1	Detailed Description	3350
7.1272.2	Function Documentation	3350
7.1272.2.1	SysTick_Handler()	3350
7.1272.2.2	TimerGet()	3350
7.1272.2.3	TimerInit()	3351
7.1273	mer.c File Reference	3351
7.1273.1	Detailed Description	3351
7.1273.2	Function Documentation	3351
7.1273.2.1	SysTick_Handler()	3352
7.1273.2.2	TimerGet()	3352
7.1273.2.3	TimerInit()	3352
7.1274	mer.c File Reference	3352
7.1274.1	Detailed Description	3353
7.1274.2	Function Documentation	3353
7.1274.2.1	SysTick_Handler()	3353
7.1274.2.2	TimerGet()	3353
7.1274.2.3	TimerInit()	3354
7.1275	mer.c File Reference	3354
7.1275.1	Detailed Description	3354
7.1275.2	Function Documentation	3354
7.1275.2.1	TimerGet()	3355
7.1275.2.2	TimerInit()	3355
7.1276	mer.c File Reference	3355
7.1276.1	Detailed Description	3356
7.1276.2	Function Documentation	3356
7.1276.2.1	SysTick_Handler()	3356

7.1276.2.2TimerGet()	3356
7.1276.2.3TimerInit()	3356
7.1277mer.c File Reference	3357
7.1277.1Detailed Description	3357
7.1277.2Function Documentation	3357
7.1277.2.1SysTick_Handler()	3357
7.1277.2.2TimerGet()	3358
7.1277.2.3TimerInit()	3358
7.1278mer.c File Reference	3358
7.1278.1Detailed Description	3359
7.1278.2Function Documentation	3359
7.1278.2.1SysTick_Handler()	3359
7.1278.2.2TimerGet()	3359
7.1278.2.3TimerInit()	3359
7.1279mer.c File Reference	3360
7.1279.1Detailed Description	3360
7.1279.2Function Documentation	3361
7.1279.2.1TimerDeinit()	3361
7.1279.2.2TimerGet()	3361
7.1279.2.3TimerInit()	3361
7.1279.2.4TimerISRHandler()	3361
7.1279.2.5TimerSet()	3361
7.1280mer.c File Reference	3362
7.1280.1Detailed Description	3363
7.1280.2Function Documentation	3363
7.1280.2.1TimerDeinit()	3363
7.1280.2.2TimerGet()	3363
7.1280.2.3TimerInit()	3363
7.1280.2.4TimerISRHandler()	3364
7.1280.2.5TimerSet()	3364

7.1281mer.c File Reference	3364
7.1281.1Detailed Description	3365
7.1281.2Function Documentation	3365
7.1281.2.1TimerGet()	3365
7.1281.2.2TimerInit()	3365
7.1281.2.3TimerReset()	3366
7.1281.2.4TimerUpdate()	3366
7.1282mer.c File Reference	3366
7.1282.1Detailed Description	3367
7.1282.2Function Documentation	3367
7.1282.2.1TimerGet()	3367
7.1282.2.2TimerInit()	3367
7.1282.2.3TimerReset()	3368
7.1282.2.4TimerUpdate()	3368
7.1283mer.c File Reference	3368
7.1283.1Detailed Description	3369
7.1283.2Function Documentation	3369
7.1283.2.1HAL_GetTick()	3369
7.1283.2.2SysTick_Handler()	3370
7.1283.2.3TimerGet()	3370
7.1283.2.4TimerInit()	3370
7.1283.2.5TimerReset()	3371
7.1283.2.6TimerUpdate()	3371
7.1284mer.c File Reference	3371
7.1284.1Detailed Description	3372
7.1284.2Function Documentation	3372
7.1284.2.1HAL_GetTick()	3372
7.1284.2.2SysTick_Handler()	3373
7.1284.2.3TimerGet()	3373
7.1284.2.4TimerInit()	3373

7.1284.2.5TimerReset()	3374
7.1284.2.6TimerUpdate()	3374
7.1285mer.c File Reference	3374
7.1285.1Detailed Description	3375
7.1285.2Function Documentation	3375
7.1285.2.1TimerGet()	3376
7.1285.2.2TimerInit()	3376
7.1285.2.3TimerReset()	3376
7.1285.2.4TimerUpdate()	3376
7.1286mer.c File Reference	3377
7.1286.1Detailed Description	3377
7.1286.2Function Documentation	3378
7.1286.2.1HAL_GetTick()	3378
7.1286.2.2SysTick_Handler()	3378
7.1286.2.3TimerGet()	3378
7.1286.2.4TimerInit()	3379
7.1286.2.5TimerReset()	3379
7.1286.2.6TimerUpdate()	3379
7.1287mer.c File Reference	3380
7.1287.1Detailed Description	3381
7.1287.2Function Documentation	3381
7.1287.2.1TimerGet()	3381
7.1287.2.2TimerInit()	3381
7.1287.2.3TimerReset()	3382
7.1287.2.4TimerUpdate()	3382
7.1288mer.c File Reference	3382
7.1288.1Detailed Description	3383
7.1288.2Function Documentation	3383
7.1288.2.1TimerGet()	3384
7.1288.2.2TimerInit()	3384

7.1288.2.3TimerReset()	3384
7.1288.2.4TimerUpdate()	3384
7.1289mer.c File Reference	3385
7.1289.1Detailed Description	3385
7.1289.2Function Documentation	3386
7.1289.2.1HAL_GetTick()	3386
7.1289.2.2SysTick_Handler()	3386
7.1289.2.3TimerGet()	3386
7.1289.2.4TimerInit()	3387
7.1289.2.5TimerReset()	3387
7.1289.2.6TimerUpdate()	3387
7.1290mer.c File Reference	3388
7.1290.1Detailed Description	3388
7.1290.2Function Documentation	3389
7.1290.2.1HAL_GetTick()	3389
7.1290.2.2SysTick_Handler()	3389
7.1290.2.3TimerGet()	3389
7.1290.2.4TimerInit()	3390
7.1290.2.5TimerReset()	3390
7.1290.2.6TimerUpdate()	3390
7.1291mer.c File Reference	3391
7.1291.1Detailed Description	3391
7.1291.2Function Documentation	3391
7.1291.2.1TimerGet()	3392
7.1291.2.2TimerInit()	3392
7.1291.2.3TimerReset()	3392
7.1291.2.4TimerUpdate()	3392
7.1292mer.c File Reference	3393
7.1292.1Detailed Description	3393
7.1292.2Function Documentation	3394

7.1292.2.1HAL_GetTick()	3394
7.1292.2.2SysTick_Handler()	3394
7.1292.2.3TimerGet()	3394
7.1292.2.4TimerInit()	3395
7.1292.2.5TimerReset()	3395
7.1292.2.6TimerUpdate()	3395
7.1292mer.c File Reference	3396
7.1293.1Detailed Description	3396
7.1293.2Function Documentation	3397
7.1293.2.1HAL_GetTick()	3397
7.1293.2.2SysTick_Handler()	3397
7.1293.2.3TimerGet()	3397
7.1293.2.4TimerInit()	3398
7.1293.2.5TimerReset()	3398
7.1293.2.6TimerUpdate()	3398
7.1294mer.c File Reference	3399
7.1294.1Detailed Description	3399
7.1294.2Function Documentation	3400
7.1294.2.1HAL_GetTick()	3400
7.1294.2.2SysTick_Handler()	3400
7.1294.2.3TimerGet()	3400
7.1294.2.4TimerInit()	3401
7.1294.2.5TimerReset()	3401
7.1294.2.6TimerUpdate()	3401
7.1295mer.c File Reference	3402
7.1295.1Detailed Description	3403
7.1295.2Function Documentation	3403
7.1295.2.1TimerGet()	3403
7.1295.2.2TimerInit()	3403
7.1295.2.3TimerReset()	3404

7.1295.2.4TimerUpdate()	3404
7.1296mer.c File Reference	3404
7.1296.1Detailed Description	3405
7.1296.2Function Documentation	3405
7.1296.2.1TimerGet()	3406
7.1296.2.2TimerInit()	3406
7.1296.2.3TimerReset()	3406
7.1296.2.4TimerUpdate()	3406
7.1297mer.c File Reference	3407
7.1297.1Detailed Description	3407
7.1297.2Function Documentation	3408
7.1297.2.1HAL_GetTick()	3408
7.1297.2.2SysTick_Handler()	3408
7.1297.2.3TimerGet()	3408
7.1297.2.4TimerInit()	3409
7.1297.2.5TimerReset()	3409
7.1297.2.6TimerUpdate()	3409
7.1298mer.c File Reference	3410
7.1298.1Detailed Description	3410
7.1298.2Function Documentation	3411
7.1298.2.1HAL_GetTick()	3411
7.1298.2.2SysTick_Handler()	3411
7.1298.2.3TimerGet()	3411
7.1298.2.4TimerInit()	3412
7.1298.2.5TimerReset()	3412
7.1298.2.6TimerUpdate()	3412
7.1299mer.c File Reference	3413
7.1299.1Detailed Description	3414
7.1299.2Function Documentation	3414
7.1299.2.1TimerGet()	3414

7.1299.2.2TimerInit()	3414
7.1299.2.3TimerReset()	3415
7.1299.2.4TimerUpdate()	3415
7.1300mer.h File Reference	3415
7.1300.1Detailed Description	3416
7.1300.2Function Documentation	3416
7.1300.2.1TimerGet()	3416
7.1300.2.2TimerInit()	3416
7.1301mer.h File Reference	3417
7.1301.1Detailed Description	3417
7.1301.2Function Documentation	3417
7.1301.2.1TimerGet()	3417
7.1301.2.2TimerInit()	3418
7.1302mer.h File Reference	3418
7.1302.1Detailed Description	3418
7.1302.2Function Documentation	3419
7.1302.2.1TimerGet()	3419
7.1302.2.2TimerInit()	3419
7.1303mer.h File Reference	3419
7.1303.1Detailed Description	3420
7.1303.2Function Documentation	3420
7.1303.2.1TimerGet()	3420
7.1303.2.2TimerInit()	3420
7.1304mer.h File Reference	3421
7.1304.1Detailed Description	3421
7.1304.2Function Documentation	3421
7.1304.2.1TimerGet()	3421
7.1304.2.2TimerInit()	3422
7.1305mer.h File Reference	3422
7.1305.1Detailed Description	3422

7.1305.2Function Documentation	3423
7.1305.2.1TimerGet()	3423
7.1305.2.2TimerInit()	3423
7.1306mer.h File Reference	3423
7.1306.1Detailed Description	3424
7.1306.2Function Documentation	3424
7.1306.2.1TimerGet()	3424
7.1306.2.2TimerInit()	3424
7.1307mer.h File Reference	3425
7.1307.1Detailed Description	3425
7.1307.2Function Documentation	3425
7.1307.2.1TimerGet()	3425
7.1307.2.2TimerInit()	3426
7.1308mer.h File Reference	3426
7.1308.1Detailed Description	3426
7.1308.2Function Documentation	3427
7.1308.2.1TimerGet()	3427
7.1308.2.2TimerInit()	3427
7.1309mer.h File Reference	3427
7.1309.1Detailed Description	3428
7.1309.2Function Documentation	3428
7.1309.2.1TimerGet()	3428
7.1309.2.2TimerInit()	3428
7.1310mer.h File Reference	3429
7.1310.1Detailed Description	3429
7.1310.2Function Documentation	3429
7.1310.2.1TimerGet()	3429
7.1310.2.2TimerInit()	3430
7.1311mer.h File Reference	3430
7.1311.1Detailed Description	3430

7.1311.2Function Documentation	3431
7.1311.2.1TimerGet()	3431
7.1311.2.2TimerInit()	3431
7.1312mer.h File Reference	3431
7.1312.1Detailed Description	3432
7.1312.2Function Documentation	3432
7.1312.2.1TimerGet()	3432
7.1312.2.2TimerInit()	3432
7.1313mer.h File Reference	3433
7.1313.1Detailed Description	3433
7.1313.2Function Documentation	3433
7.1313.2.1TimerDeinit()	3434
7.1313.2.2TimerGet()	3434
7.1313.2.3TimerInit()	3434
7.1313.2.4TimerSet()	3434
7.1314mer.h File Reference	3435
7.1314.1Detailed Description	3435
7.1314.2Function Documentation	3436
7.1314.2.1TimerDeinit()	3436
7.1314.2.2TimerGet()	3436
7.1314.2.3TimerInit()	3436
7.1314.2.4TimerSet()	3436
7.1315mer.h File Reference	3437
7.1315.1Detailed Description	3437
7.1315.2Function Documentation	3437
7.1315.2.1TimerGet()	3438
7.1315.2.2TimerInit()	3438
7.1316mer.h File Reference	3438
7.1316.1Detailed Description	3439
7.1316.2Function Documentation	3439

7.1316.2.1TimerGet()	3439
7.1316.2.2TimerInit()	3439
7.1317mer.h File Reference	3440
7.1317.1Detailed Description	3440
7.1317.2Function Documentation	3440
7.1317.2.1TimerGet()	3440
7.1317.2.2TimerInit()	3441
7.1318mer.h File Reference	3441
7.1318.1Detailed Description	3441
7.1318.2Function Documentation	3442
7.1318.2.1TimerGet()	3442
7.1318.2.2TimerInit()	3442
7.1319mer.h File Reference	3442
7.1319.1Detailed Description	3443
7.1319.2Function Documentation	3443
7.1319.2.1TimerDeinit()	3443
7.1319.2.2TimerGet()	3443
7.1319.2.3TimerInit()	3444
7.1319.2.4TimerISRHandler()	3444
7.1319.2.5TimerSet()	3444
7.1320mer.h File Reference	3445
7.1320.1Detailed Description	3445
7.1320.2Function Documentation	3445
7.1320.2.1TimerDeinit()	3446
7.1320.2.2TimerGet()	3446
7.1320.2.3TimerInit()	3446
7.1320.2.4TimerISRHandler()	3446
7.1320.2.5TimerSet()	3446
7.1321mer.h File Reference	3447
7.1321.1Detailed Description	3447

7.1321.2	Function Documentation	3447
7.1321.2.1	TimerGet()	3448
7.1321.2.2	TimerInit()	3448
7.1322	timer.h File Reference	3448
7.1322.1	Detailed Description	3449
7.1322.2	Function Documentation	3449
7.1322.2.1	TimerGet()	3449
7.1322.2.2	TimerInit()	3449
7.1323	timer.h File Reference	3450
7.1323.1	Detailed Description	3450
7.1323.2	Function Documentation	3450
7.1323.2.1	TimerGet()	3450
7.1323.2.2	TimerInit()	3451
7.1324	timer.h File Reference	3451
7.1324.1	Detailed Description	3451
7.1324.2	Function Documentation	3452
7.1324.2.1	TimerGet()	3452
7.1324.2.2	TimerInit()	3452
7.1325	timer.h File Reference	3452
7.1325.1	Detailed Description	3453
7.1325.2	Function Documentation	3453
7.1325.2.1	TimerGet()	3453
7.1325.2.2	TimerInit()	3453
7.1326	timer.h File Reference	3454
7.1326.1	Detailed Description	3454
7.1326.2	Function Documentation	3454
7.1326.2.1	TimerGet()	3454
7.1326.2.2	TimerInit()	3455
7.1327	timer.h File Reference	3455
7.1327.1	Detailed Description	3455

7.1327.2	Function Documentation	3456
7.1327.2.1	TimerGet()	3456
7.1327.2.2	TimerInit()	3456
7.1328	timer.h File Reference	3456
7.1328.1	Detailed Description	3457
7.1328.2	Function Documentation	3457
7.1328.2.1	TimerGet()	3457
7.1328.2.2	TimerInit()	3457
7.1329	timer.h File Reference	3458
7.1329.1	Detailed Description	3458
7.1329.2	Function Documentation	3458
7.1329.2.1	TimerGet()	3458
7.1329.2.2	TimerInit()	3459
7.1330	timer.h File Reference	3459
7.1330.1	Detailed Description	3459
7.1330.2	Function Documentation	3460
7.1330.2.1	TimerGet()	3460
7.1330.2.2	TimerInit()	3460
7.1331	timer.h File Reference	3460
7.1331.1	Detailed Description	3461
7.1331.2	Function Documentation	3461
7.1331.2.1	TimerGet()	3461
7.1331.2.2	TimerInit()	3461
7.1332	timer.h File Reference	3462
7.1332.1	Detailed Description	3462
7.1332.2	Function Documentation	3462
7.1332.2.1	TimerGet()	3462
7.1332.2.2	TimerInit()	3463
7.1333	timer.h File Reference	3463
7.1333.1	Detailed Description	3463

7.1333.2Function Documentation	3464
7.1333.2.1TimerGet()	3464
7.1333.2.2TimerInit()	3464
7.1334mer.h File Reference	3464
7.1334.1Detailed Description	3465
7.1334.2Function Documentation	3465
7.1334.2.1TimerGet()	3465
7.1334.2.2TimerInit()	3465
7.1335mer.h File Reference	3466
7.1335.1Detailed Description	3466
7.1335.2Function Documentation	3466
7.1335.2.1TimerGet()	3466
7.1335.2.2TimerInit()	3467
7.1336mer.h File Reference	3467
7.1336.1Detailed Description	3467
7.1336.2Function Documentation	3468
7.1336.2.1TimerGet()	3468
7.1336.2.2TimerInit()	3468
7.1337mer.h File Reference	3468
7.1337.1Detailed Description	3469
7.1337.2Function Documentation	3469
7.1337.2.1TimerGet()	3469
7.1337.2.2TimerInit()	3469
7.1338mer.h File Reference	3470
7.1338.1Detailed Description	3470
7.1338.2Function Documentation	3470
7.1338.2.1TimerGet()	3470
7.1338.2.2TimerInit()	3471
7.1339mer.h File Reference	3471
7.1339.1Detailed Description	3471

7.1339.2Function Documentation	3472
7.1339.2.1TimerGet()	3472
7.1339.2.2TimerInit()	3472
7.1340mer.h File Reference	3472
7.1340.1Detailed Description	3473
7.1340.2Function Documentation	3473
7.1340.2.1TimerGet()	3473
7.1340.2.2TimerInit()	3473
7.1341mer.h File Reference	3474
7.1341.1Detailed Description	3474
7.1341.2Function Documentation	3474
7.1341.2.1TimerGet()	3474
7.1341.2.2TimerInit()	3475
7.1342mer.h File Reference	3475
7.1342.1Detailed Description	3475
7.1342.2Function Documentation	3476
7.1342.2.1TimerGet()	3476
7.1342.2.2TimerInit()	3476
7.1343mer.h File Reference	3476
7.1343.1Detailed Description	3477
7.1343.2Function Documentation	3477
7.1343.2.1TimerGet()	3477
7.1343.2.2TimerInit()	3477
7.1344mer.h File Reference	3478
7.1344.1Detailed Description	3478
7.1344.2Function Documentation	3478
7.1344.2.1TimerGet()	3478
7.1344.2.2TimerInit()	3479
7.1345mer.h File Reference	3479
7.1345.1Detailed Description	3479

7.1345.2Function Documentation	3480
7.1345.2.1TimerGet()	3480
7.1345.2.2TimerInit()	3480
7.1346mer.h File Reference	3480
7.1346.1Detailed Description	3481
7.1346.2Function Documentation	3481
7.1346.2.1TimerGet()	3481
7.1346.2.2TimerInit()	3481
7.1347mer.h File Reference	3482
7.1347.1Detailed Description	3482
7.1347.2Function Documentation	3482
7.1347.2.1TimerGet()	3482
7.1347.2.2TimerInit()	3483
7.1348mer.h File Reference	3483
7.1348.1Detailed Description	3483
7.1348.2Function Documentation	3484
7.1348.2.1TimerGet()	3484
7.1348.2.2TimerInit()	3484
7.1349mer.h File Reference	3484
7.1349.1Detailed Description	3485
7.1349.2Function Documentation	3485
7.1349.2.1TimerGet()	3485
7.1349.2.2TimerInit()	3485
7.1350mer.h File Reference	3486
7.1350.1Detailed Description	3486
7.1350.2Function Documentation	3486
7.1350.2.1TimerGet()	3486
7.1350.2.2TimerInit()	3487
7.1351mer.h File Reference	3487
7.1351.1Detailed Description	3487

7.1351.2	Function Documentation	3488
7.1351.2.1	TimerGet()	3488
7.1351.2.2	TimerInit()	3488
7.1352	timer.h File Reference	3488
7.1352.1	Detailed Description	3489
7.1352.2	Function Documentation	3489
7.1352.2.1	TimerGet()	3489
7.1352.2.2	TimerInit()	3489
7.1353	timer.h File Reference	3490
7.1353.1	Detailed Description	3490
7.1353.2	Function Documentation	3490
7.1353.2.1	TimerGet()	3490
7.1353.2.2	TimerInit()	3491
7.1354	timer.h File Reference	3491
7.1354.1	Detailed Description	3491
7.1354.2	Function Documentation	3492
7.1354.2.1	TimerGet()	3492
7.1354.2.2	TimerInit()	3492
7.1355	timer.h File Reference	3492
7.1355.1	Detailed Description	3493
7.1355.2	Function Documentation	3493
7.1355.2.1	TimerGet()	3493
7.1355.2.2	TimerInit()	3493
7.1356	timer.h File Reference	3494
7.1356.1	Detailed Description	3494
7.1356.2	Function Documentation	3494
7.1356.2.1	TimerGet()	3494
7.1356.2.2	TimerInit()	3495
7.1357	timer.h File Reference	3495
7.1357.1	Detailed Description	3495

7.1357.2Function Documentation	3496
7.1357.2.1TimerGet()	3496
7.1357.2.2TimerInit()	3496
7.1358mer.h File Reference	3496
7.1358.1Detailed Description	3497
7.1358.2Function Documentation	3497
7.1358.2.1TimerGet()	3497
7.1358.2.2TimerInit()	3497
7.1359mer.h File Reference	3498
7.1359.1Detailed Description	3498
7.1359.2Function Documentation	3498
7.1359.2.1TimerGet()	3498
7.1359.2.2TimerInit()	3499
7.1360mer.h File Reference	3499
7.1360.1Detailed Description	3499
7.1360.2Function Documentation	3500
7.1360.2.1TimerGet()	3500
7.1360.2.2TimerInit()	3500
7.1361mer.h File Reference	3500
7.1361.1Detailed Description	3501
7.1361.2Function Documentation	3501
7.1361.2.1TimerGet()	3501
7.1361.2.2TimerInit()	3501
7.1362mer.h File Reference	3502
7.1362.1Detailed Description	3502
7.1362.2Function Documentation	3502
7.1362.2.1TimerGet()	3502
7.1362.2.2TimerInit()	3503
7.1363mer.h File Reference	3503
7.1363.1Detailed Description	3504

7.1363.2Function Documentation	3504
7.1363.2.1TimerDeinit()	3504
7.1363.2.2TimerGet()	3504
7.1363.2.3TimerInit()	3504
7.1363.2.4TimerSet()	3504
7.1364mer.h File Reference	3505
7.1364.1Detailed Description	3505
7.1364.2Function Documentation	3506
7.1364.2.1TimerDeinit()	3506
7.1364.2.2TimerGet()	3506
7.1364.2.3TimerInit()	3506
7.1364.2.4TimerSet()	3506
7.1365mer.h File Reference	3507
7.1365.1Detailed Description	3507
7.1365.2Function Documentation	3507
7.1365.2.1TimerGet()	3508
7.1365.2.2TimerInit()	3508
7.1366mer.h File Reference	3508
7.1366.1Detailed Description	3509
7.1366.2Function Documentation	3509
7.1366.2.1TimerGet()	3509
7.1366.2.2TimerInit()	3509
7.1367mer.h File Reference	3510
7.1367.1Detailed Description	3510
7.1367.2Function Documentation	3510
7.1367.2.1TimerGet()	3510
7.1367.2.2TimerInit()	3511
7.1368mer.h File Reference	3511
7.1368.1Detailed Description	3511
7.1368.2Function Documentation	3512

7.1368.2.1TimerGet()	3512
7.1368.2.2TimerInit()	3512
7.1369mer.h File Reference	3512
7.1369.1Detailed Description	3513
7.1369.2Function Documentation	3513
7.1369.2.1TimerGet()	3513
7.1369.2.2TimerInit()	3513
7.1370mer.h File Reference	3514
7.1370.1Detailed Description	3514
7.1370.2Function Documentation	3514
7.1370.2.1TimerGet()	3514
7.1370.2.2TimerInit()	3515
7.1371mer.h File Reference	3515
7.1371.1Detailed Description	3515
7.1371.2Function Documentation	3516
7.1371.2.1TimerGet()	3516
7.1371.2.2TimerInit()	3516
7.1372mer.h File Reference	3516
7.1372.1Detailed Description	3517
7.1372.2Function Documentation	3517
7.1372.2.1TimerGet()	3517
7.1372.2.2TimerInit()	3517
7.1373mer.h File Reference	3518
7.1373.1Detailed Description	3518
7.1373.2Function Documentation	3518
7.1373.2.1TimerGet()	3518
7.1373.2.2TimerInit()	3519
7.1374mer.h File Reference	3519
7.1374.1Detailed Description	3519
7.1374.2Function Documentation	3520

7.1374.2.1TimerGet()	3520
7.1374.2.2TimerInit()	3520
7.1375mer.h File Reference	3520
7.1375.1Detailed Description	3521
7.1375.2Function Documentation	3521
7.1375.2.1TimerGet()	3521
7.1375.2.2TimerInit()	3521
7.1376mer.h File Reference	3522
7.1376.1Detailed Description	3522
7.1376.2Function Documentation	3522
7.1376.2.1TimerGet()	3522
7.1376.2.2TimerInit()	3523
7.1377mer.h File Reference	3523
7.1377.1Detailed Description	3524
7.1377.2Function Documentation	3524
7.1377.2.1TimerDeinit()	3524
7.1377.2.2TimerGet()	3524
7.1377.2.3TimerInit()	3524
7.1377.2.4TimerISRHandler()	3525
7.1377.2.5TimerSet()	3525
7.1378mer.h File Reference	3525
7.1378.1Detailed Description	3526
7.1378.2Function Documentation	3526
7.1378.2.1TimerDeinit()	3526
7.1378.2.2TimerGet()	3526
7.1378.2.3TimerInit()	3527
7.1378.2.4TimerISRHandler()	3527
7.1378.2.5TimerSet()	3527
7.1379mer.h File Reference	3528
7.1379.1Detailed Description	3528

7.1379.2Function Documentation	3528
7.1379.2.1TimerGet()	3528
7.1379.2.2TimerInit()	3529
7.1379.2.3TimerReset()	3529
7.1379.2.4TimerUpdate()	3529
7.1380types.h File Reference	3530
7.1380.1Detailed Description	3530
7.1380.2Typedef Documentation	3530
7.1380.2.1blt_addr	3530
7.1380.2.2blt_bool	3530
7.1380.2.3blt_char	3531
7.1380.2.4blt_int16s	3531
7.1380.2.5blt_int16u	3531
7.1380.2.6blt_int32s	3531
7.1380.2.7blt_int32u	3531
7.1380.2.8blt_int8s	3531
7.1380.2.9blt_int8u	3531
7.1381types.h File Reference	3531
7.1381.1Detailed Description	3532
7.1381.2Typedef Documentation	3532
7.1381.2.1blt_addr	3532
7.1381.2.2blt_bool	3532
7.1381.2.3blt_char	3532
7.1381.2.4blt_int16s	3533
7.1381.2.5blt_int16u	3533
7.1381.2.6blt_int32s	3533
7.1381.2.7blt_int32u	3533
7.1381.2.8blt_int8s	3533
7.1381.2.9blt_int8u	3533
7.1382types.h File Reference	3533

7.1382.1Detailed Description	3534
7.1382.2Typedef Documentation	3534
7.1382.2.1b1t_addr	3534
7.1382.2.2b1t_bool	3534
7.1382.2.3b1t_char	3534
7.1382.2.4b1t_int16s	3535
7.1382.2.5b1t_int16u	3535
7.1382.2.6b1t_int32s	3535
7.1382.2.7b1t_int32u	3535
7.1382.2.8b1t_int8s	3535
7.1382.2.9b1t_int8u	3535
7.1382.3types.h File Reference	3535
7.1383.1Detailed Description	3536
7.1383.2Typedef Documentation	3536
7.1383.2.1b1t_addr	3536
7.1383.2.2b1t_bool	3536
7.1383.2.3b1t_char	3536
7.1383.2.4b1t_int16s	3537
7.1383.2.5b1t_int16u	3537
7.1383.2.6b1t_int32s	3537
7.1383.2.7b1t_int32u	3537
7.1383.2.8b1t_int64s	3537
7.1383.2.9b1t_int64u	3537
7.1383.2.10b1t_int8s	3537
7.1383.2.11b1t_int8u	3537
7.1383.3types.h File Reference	3538
7.1384.1Detailed Description	3538
7.1384.2Typedef Documentation	3538
7.1384.2.1b1t_addr	3538
7.1384.2.2b1t_bool	3538

7.1384.2.3b1t_char	3539
7.1384.2.4b1t_int16s	3539
7.1384.2.5b1t_int16u	3539
7.1384.2.6b1t_int32s	3539
7.1384.2.7b1t_int32u	3539
7.1384.2.8b1t_int8s	3539
7.1384.2.9b1t_int8u	3539
7.1385types.h File Reference	3539
7.1385.1Detailed Description	3540
7.1385.2Typedef Documentation	3540
7.1385.2.1b1t_addr	3540
7.1385.2.2b1t_bool	3540
7.1385.2.3b1t_char	3540
7.1385.2.4b1t_int16s	3541
7.1385.2.5b1t_int16u	3541
7.1385.2.6b1t_int32s	3541
7.1385.2.7b1t_int32u	3541
7.1385.2.8b1t_int64s	3541
7.1385.2.9b1t_int64u	3541
7.1385.2.10b1t_int8s	3541
7.1385.2.11b1t_int8u	3541
7.1386types.h File Reference	3542
7.1386.1Detailed Description	3542
7.1386.2Typedef Documentation	3542
7.1386.2.1b1t_addr	3542
7.1386.2.2b1t_bool	3542
7.1386.2.3b1t_char	3543
7.1386.2.4b1t_int16s	3543
7.1386.2.5b1t_int16u	3543
7.1386.2.6b1t_int32s	3543

7.1386.2.7blt_int32u	3543
7.1386.2.8blt_int8s	3543
7.1386.2.9blt_int8u	3543
7.1387types.h File Reference	3543
7.1387.1Detailed Description	3544
7.1387.2Typedef Documentation	3544
7.1387.2.1blt_addr	3544
7.1387.2.2blt_bool	3544
7.1387.2.3blt_char	3544
7.1387.2.4blt_int16s	3545
7.1387.2.5blt_int16u	3545
7.1387.2.6blt_int32s	3545
7.1387.2.7blt_int32u	3545
7.1387.2.8blt_int8s	3545
7.1387.2.9blt_int8u	3545
7.1388types.h File Reference	3545
7.1388.1Detailed Description	3546
7.1388.2Typedef Documentation	3546
7.1388.2.1blt_addr	3546
7.1388.2.2blt_bool	3546
7.1388.2.3blt_char	3546
7.1388.2.4blt_int16s	3547
7.1388.2.5blt_int16u	3547
7.1388.2.6blt_int32s	3547
7.1388.2.7blt_int32u	3547
7.1388.2.8blt_int8s	3547
7.1388.2.9blt_int8u	3547
7.1389types.h File Reference	3547
7.1389.1Detailed Description	3548
7.1389.2Typedef Documentation	3548

7.1389.2.1blt_addr	3548
7.1389.2.2blt_bool	3548
7.1389.2.3blt_char	3548
7.1389.2.4blt_int16s	3549
7.1389.2.5blt_int16u	3549
7.1389.2.6blt_int32s	3549
7.1389.2.7blt_int32u	3549
7.1389.2.8blt_int8s	3549
7.1389.2.9blt_int8u	3549
7.1390types.h File Reference	3549
7.1390.1Detailed Description	3550
7.1390.2Typedef Documentation	3550
7.1390.2.1blt_addr	3550
7.1390.2.2blt_bool	3550
7.1390.2.3blt_char	3550
7.1390.2.4blt_int16s	3551
7.1390.2.5blt_int16u	3551
7.1390.2.6blt_int32s	3551
7.1390.2.7blt_int32u	3551
7.1390.2.8blt_int8s	3551
7.1390.2.9blt_int8u	3551
7.1391types.h File Reference	3551
7.1391.1Detailed Description	3552
7.1391.2Typedef Documentation	3552
7.1391.2.1blt_addr	3552
7.1391.2.2blt_bool	3552
7.1391.2.3blt_char	3552
7.1391.2.4blt_int16s	3553
7.1391.2.5blt_int16u	3553
7.1391.2.6blt_int32s	3553

7.1391.2.7blt_int32u	3553
7.1391.2.8blt_int8s	3553
7.1391.2.9blt_int8u	3553
7.1392.1types.h File Reference	3553
7.1392.1Detailed Description	3554
7.1392.2Typedef Documentation	3554
7.1392.2.1blt_addr	3554
7.1392.2.2blt_bool	3554
7.1392.2.3blt_char	3554
7.1392.2.4blt_int16s	3555
7.1392.2.5blt_int16u	3555
7.1392.2.6blt_int32s	3555
7.1392.2.7blt_int32u	3555
7.1392.2.8blt_int8s	3555
7.1392.2.9blt_int8u	3555
7.1393.1types.h File Reference	3555
7.1393.1Detailed Description	3556
7.1393.2Typedef Documentation	3556
7.1393.2.1blt_addr	3556
7.1393.2.2blt_bool	3556
7.1393.2.3blt_char	3556
7.1393.2.4blt_int16s	3557
7.1393.2.5blt_int16u	3557
7.1393.2.6blt_int32s	3557
7.1393.2.7blt_int32u	3557
7.1393.2.8blt_int8s	3557
7.1393.2.9blt_int8u	3557
7.1394.1types.h File Reference	3557
7.1394.1Detailed Description	3558
7.1394.2Typedef Documentation	3558

7.1394.2.1blt_addr	3558
7.1394.2.2blt_bool	3558
7.1394.2.3blt_char	3558
7.1394.2.4blt_int16s	3559
7.1394.2.5blt_int16u	3559
7.1394.2.6blt_int32s	3559
7.1394.2.7blt_int32u	3559
7.1394.2.8blt_int8s	3559
7.1394.2.9blt_int8u	3559
7.1395types.h File Reference	3559
7.1395.1Detailed Description	3560
7.1395.2Typedef Documentation	3560
7.1395.2.1blt_addr	3560
7.1395.2.2blt_bool	3560
7.1395.2.3blt_char	3560
7.1395.2.4blt_int16s	3561
7.1395.2.5blt_int16u	3561
7.1395.2.6blt_int32s	3561
7.1395.2.7blt_int32u	3561
7.1395.2.8blt_int8s	3561
7.1395.2.9blt_int8u	3561
7.1396types.h File Reference	3561
7.1396.1Detailed Description	3562
7.1396.2Typedef Documentation	3562
7.1396.2.1blt_addr	3562
7.1396.2.2blt_bool	3562
7.1396.2.3blt_char	3562
7.1396.2.4blt_int16s	3563
7.1396.2.5blt_int16u	3563
7.1396.2.6blt_int32s	3563

7.1396.2.7blt_int32u	3563
7.1396.2.8blt_int8s	3563
7.1396.2.9blt_int8u	3563
7.1397types.h File Reference	3563
7.1397.1Detailed Description	3564
7.1397.2Typedef Documentation	3564
7.1397.2.1blt_addr	3564
7.1397.2.2blt_bool	3564
7.1397.2.3blt_char	3564
7.1397.2.4blt_int16s	3565
7.1397.2.5blt_int16u	3565
7.1397.2.6blt_int32s	3565
7.1397.2.7blt_int32u	3565
7.1397.2.8blt_int8s	3565
7.1397.2.9blt_int8u	3565
7.1398types.h File Reference	3565
7.1398.1Detailed Description	3566
7.1398.2Typedef Documentation	3566
7.1398.2.1blt_addr	3566
7.1398.2.2blt_bool	3566
7.1398.2.3blt_char	3566
7.1398.2.4blt_int16s	3567
7.1398.2.5blt_int16u	3567
7.1398.2.6blt_int32s	3567
7.1398.2.7blt_int32u	3567
7.1398.2.8blt_int8s	3567
7.1398.2.9blt_int8u	3567
7.1399sb.c File Reference	3568
7.1399.1Detailed Description	3569
7.1399.2Function Documentation	3569

7.1399.2.1	UsbFifoMgrCreate()	3569
7.1399.2.2	UsbFifoMgrInit()	3570
7.1399.2.3	UsbFifoMgrRead()	3570
7.1399.2.4	UsbFifoMgrScan()	3571
7.1399.2.5	UsbFifoMgrWrite()	3571
7.1399.2.6	UsbFree()	3571
7.1399.2.7	UsbInit()	3572
7.1399.2.8	UsbReceiveByte()	3572
7.1399.2.9	UsbReceivePacket()	3572
7.1399.2.10	UsbReceivePipeBulkOUT()	3573
7.1399.2.11	UsbTransmitByte()	3573
7.1399.2.12	UsbTransmitPacket()	3573
7.1399.2.13	UsbTransmitPipeBulkIN()	3574
7.1400	usb.c File Reference	3574
7.1400.1	Detailed Description	3576
7.1400.2	Function Documentation	3576
7.1400.2.1	UsbFifoMgrCreate()	3576
7.1400.2.2	UsbFifoMgrInit()	3576
7.1400.2.3	UsbFifoMgrRead()	3576
7.1400.2.4	UsbFifoMgrScan()	3577
7.1400.2.5	UsbFifoMgrWrite()	3577
7.1400.2.6	UsbFree()	3578
7.1400.2.7	UsbInit()	3578
7.1400.2.8	UsbReceiveByte()	3578
7.1400.2.9	UsbReceivePacket()	3579
7.1400.2.10	UsbReceivePipeBulkOUT()	3579
7.1400.2.11	UsbTransmitByte()	3579
7.1400.2.12	UsbTransmitPacket()	3580
7.1400.2.13	UsbTransmitPipeBulkIN()	3580
7.1401	usb.c File Reference	3580

7.1401.1Detailed Description	3582
7.1401.2Function Documentation	3582
7.1401.2.1UsbFifoMgrCreate()	3582
7.1401.2.2UsbFifoMgrInit()	3582
7.1401.2.3UsbFifoMgrRead()	3583
7.1401.2.4UsbFifoMgrScan()	3583
7.1401.2.5UsbFifoMgrWrite()	3584
7.1401.2.6UsbFree()	3584
7.1401.2.7UsbInit()	3584
7.1401.2.8UsbReceiveByte()	3584
7.1401.2.9UsbReceivePacket()	3585
7.1401.2.10UsbReceivePipeBulkOUT()	3585
7.1401.2.11UsbTransmitByte()	3585
7.1401.2.12UsbTransmitPacket()	3586
7.1401.2.13UsbTransmitPipeBulkIN()	3586
7.1402usb.c File Reference	3587
7.1402.1Detailed Description	3588
7.1402.2Function Documentation	3588
7.1402.2.1UsbFifoMgrCreate()	3588
7.1402.2.2UsbFifoMgrInit()	3589
7.1402.2.3UsbFifoMgrRead()	3589
7.1402.2.4UsbFifoMgrScan()	3590
7.1402.2.5UsbFifoMgrWrite()	3590
7.1402.2.6UsbFree()	3590
7.1402.2.7UsbInit()	3591
7.1402.2.8UsbReceiveByte()	3591
7.1402.2.9UsbReceivePacket()	3591
7.1402.2.10UsbReceivePipeBulkOUT()	3592
7.1402.2.11UsbTransmitByte()	3592
7.1402.2.12UsbTransmitPacket()	3592

7.1402.2.13	usbTransmitPipeBulkIN()	3593
7.1403	usb.c File Reference	3593
7.1403.1	Detailed Description	3594
7.1403.2	Function Documentation	3595
7.1403.2.1	UsbFifoMgrCreate()	3595
7.1403.2.2	UsbFifoMgrInit()	3595
7.1403.2.3	UsbFifoMgrRead()	3595
7.1403.2.4	UsbFifoMgrScan()	3596
7.1403.2.5	UsbFifoMgrWrite()	3596
7.1403.2.6	UsbFree()	3597
7.1403.2.7	UsbInit()	3597
7.1403.2.8	UsbReceiveByte()	3597
7.1403.2.9	UsbReceivePacket()	3598
7.1403.2.10	UsbReceivePipeBulkOUT()	3598
7.1403.2.11	UsbTransmitByte()	3598
7.1403.2.12	UsbTransmitPacket()	3599
7.1403.2.13	UsbTransmitPipeBulkIN()	3599
7.1404	usb.c File Reference	3599
7.1404.1	Detailed Description	3601
7.1404.2	Function Documentation	3601
7.1404.2.1	UsbBulkRxHandler()	3601
7.1404.2.2	UsbBulkTxHandler()	3601
7.1404.2.3	UsbFifoMgrCreate()	3602
7.1404.2.4	UsbFifoMgrInit()	3602
7.1404.2.5	UsbFifoMgrRead()	3603
7.1404.2.6	UsbFifoMgrScan()	3603
7.1404.2.7	UsbFifoMgrWrite()	3604
7.1404.2.8	UsbFree()	3604
7.1404.2.9	UsbInit()	3604
7.1404.2.10	UsbReceiveByte()	3604

7.1404.2.11	usbReceivePacket()	3605
7.1404.2.12	usbReceivePipeBulkOUT()	3605
7.1404.2.13	usbTransmitByte()	3606
7.1404.2.14	usbTransmitPacket()	3606
7.1404.2.15	usbTransmitPipeBulkIN()	3606
7.1405	usb.c File Reference	3607
7.1405.1	Detailed Description	3608
7.1405.2	Function Documentation	3609
7.1405.2.1	UsbFifoMgrCreate()	3609
7.1405.2.2	UsbFifoMgrInit()	3609
7.1405.2.3	UsbFifoMgrRead()	3609
7.1405.2.4	UsbFifoMgrScan()	3610
7.1405.2.5	UsbFifoMgrWrite()	3610
7.1405.2.6	UsbFree()	3611
7.1405.2.7	UsbInit()	3611
7.1405.2.8	UsbReceiveByte()	3611
7.1405.2.9	UsbReceivePacket()	3612
7.1405.2.10	UsbReceivePipeBulkOUT()	3612
7.1405.2.11	UsbTransmitByte()	3612
7.1405.2.12	UsbTransmitPacket()	3613
7.1405.2.13	UsbTransmitPipeBulkIN()	3613
7.1406	usb.c File Reference	3613
7.1406.1	Detailed Description	3615
7.1406.2	Function Documentation	3615
7.1406.2.1	UsbFifoMgrCreate()	3615
7.1406.2.2	UsbFifoMgrInit()	3615
7.1406.2.3	UsbFifoMgrRead()	3616
7.1406.2.4	UsbFifoMgrScan()	3616
7.1406.2.5	UsbFifoMgrWrite()	3617
7.1406.2.6	UsbFree()	3617

7.1406.2.7	UsbInit()	3617
7.1406.2.8	UsbReceiveByte()	3617
7.1406.2.9	UsbReceivePacket()	3618
7.1406.2.10	UsbReceivePipeBulkOUT()	3618
7.1406.2.11	UsbTransmitByte()	3618
7.1406.2.12	UsbTransmitPacket()	3619
7.1406.2.13	UsbTransmitPipeBulkIN()	3619
7.1407	usb.h File Reference	3620
7.1407.1	Detailed Description	3620
7.1407.2	Function Documentation	3620
7.1407.2.1	UsbConnectHook()	3620
7.1407.2.2	UsbEnterLowPowerModeHook()	3621
7.1407.2.3	UsbFree()	3621
7.1407.2.4	UsbInit()	3621
7.1407.2.5	UsbLeaveLowPowerModeHook()	3622
7.1407.2.6	UsbReceivePacket()	3622
7.1407.2.7	UsbTransmitPacket()	3622
7.1408	ectors.c File Reference	3623
7.1408.1	Detailed Description	3623
7.1408.2	Function Documentation	3623
7.1408.2.1	reset_handler()	3624
7.1408.2.2	UnusedISR()	3624
7.1409	ectors.c File Reference	3624
7.1409.1	Detailed Description	3625
7.1409.2	Function Documentation	3625
7.1409.2.1	reset_handler()	3625
7.1409.2.2	UnusedISR()	3626
7.1410	ectors.c File Reference	3626
7.1410.1	Detailed Description	3627
7.1410.2	Function Documentation	3627

7.1410.2.1reset_handler()	3627
7.1410.2.2UnusedISR()	3627
7.1411ectors.c File Reference	3627
7.1411.1Detailed Description	3628
7.1411.2Function Documentation	3628
7.1411.2.1UnusedISR()	3628
7.1412ectors.c File Reference	3628
7.1412.1Detailed Description	3629
7.1412.2Function Documentation	3629
7.1412.2.1reset_handler()	3629
7.1412.2.2UnusedISR()	3630
7.1413ectors.c File Reference	3630
7.1413.1Detailed Description	3630
7.1413.2Function Documentation	3631
7.1413.2.1reset_handler()	3631
7.1413.2.2UnusedISR()	3631
7.1414ectors.c File Reference	3631
7.1414.1Detailed Description	3632
7.1414.2Function Documentation	3632
7.1414.2.1reset_handler()	3632
7.1414.2.2UnusedISR()	3633
7.1415ectors.c File Reference	3633
7.1415.1Detailed Description	3633
7.1415.2Function Documentation	3633
7.1415.2.1UnusedISR()	3634
7.1416ectors.c File Reference	3634
7.1416.1Detailed Description	3634
7.1416.2Function Documentation	3635
7.1416.2.1reset_handler()	3635
7.1416.2.2UnusedISR()	3635

7.1417	ectors.c File Reference	3635
7.1417.1	Detailed Description	3636
7.1417.2	Function Documentation	3636
7.1417.2.1	reset_handler()	3636
7.1417.2.2	UnusedISR()	3637
7.1418	ectors.c File Reference	3637
7.1418.1	Detailed Description	3638
7.1418.2	Function Documentation	3638
7.1418.2.1	reset_handler()	3638
7.1418.2.2	UnusedISR()	3638
7.1419	ectors.c File Reference	3638
7.1419.1	Detailed Description	3639
7.1419.2	Function Documentation	3639
7.1419.2.1	UnusedISR()	3639
7.1420	ectors.c File Reference	3639
7.1420.1	Detailed Description	3640
7.1420.2	Function Documentation	3640
7.1420.2.1	reset_handler()	3640
7.1420.2.2	UnusedISR()	3641
7.1421	ectors.c File Reference	3641
7.1421.1	Detailed Description	3641
7.1421.2	Function Documentation	3642
7.1421.2.1	ResetISR()	3642
7.1421.2.2	UnusedISR()	3642
7.1422	ectors.c File Reference	3642
7.1422.1	Detailed Description	3646
7.1422.2	Macro Definition Documentation	3647
7.1422.2.1	VCT_USER_PROGRAM_VECTOR_TABLE_STARTADDR	3647
7.1422.3	Function Documentation	3647
7.1422.3.1	Vector0_handler()	3647

7.1422.3.2	Vector10_handler()	3648
7.1422.3.3	Vector11_handler()	3648
7.1422.3.4	Vector12_handler()	3648
7.1422.3.5	Vector13_handler()	3648
7.1422.3.6	Vector14_handler()	3649
7.1422.3.7	Vector15_handler()	3649
7.1422.3.8	Vector16_handler()	3649
7.1422.3.9	Vector17_handler()	3649
7.1422.3.10	Vector18_handler()	3650
7.1422.3.11	Vector19_handler()	3650
7.1422.3.12	Vector1_handler()	3650
7.1422.3.13	Vector20_handler()	3650
7.1422.3.14	Vector21_handler()	3651
7.1422.3.15	Vector22_handler()	3651
7.1422.3.16	Vector23_handler()	3651
7.1422.3.17	Vector24_handler()	3651
7.1422.3.18	Vector25_handler()	3652
7.1422.3.19	Vector26_handler()	3652
7.1422.3.20	Vector27_handler()	3652
7.1422.3.21	Vector28_handler()	3652
7.1422.3.22	Vector29_handler()	3653
7.1422.3.23	Vector2_handler()	3653
7.1422.3.24	Vector30_handler()	3653
7.1422.3.25	Vector31_handler()	3653
7.1422.3.26	Vector32_handler()	3654
7.1422.3.27	Vector33_handler()	3654
7.1422.3.28	Vector34_handler()	3654
7.1422.3.29	Vector35_handler()	3654
7.1422.3.30	Vector36_handler()	3655
7.1422.3.31	Vector37_handler()	3655

7.1422.3.32	Vector38_handler()	3655
7.1422.3.33	Vector39_handler()	3655
7.1422.3.34	Vector40_handler()	3656
7.1422.3.35	Vector41_handler()	3656
7.1422.3.36	Vector42_handler()	3656
7.1422.3.37	Vector43_handler()	3657
7.1422.3.38	Vector44_handler()	3657
7.1422.3.39	Vector45_handler()	3657
7.1422.3.40	Vector46_handler()	3657
7.1422.3.41	Vector47_handler()	3658
7.1422.3.42	Vector48_handler()	3658
7.1422.3.43	Vector49_handler()	3658
7.1422.3.44	Vector50_handler()	3659
7.1422.3.45	Vector51_handler()	3659
7.1422.3.46	Vector52_handler()	3659
7.1422.3.47	Vector53_handler()	3659
7.1422.3.48	Vector54_handler()	3660
7.1422.3.49	Vector55_handler()	3660
7.1422.3.50	Vector56_handler()	3660
7.1422.3.51	Vector57_handler()	3660
7.1422.3.52	Vector58_handler()	3661
7.1422.3.53	Vector59_handler()	3661
7.1422.3.54	Vector60_handler()	3661
7.1422.3.55	Vector61_handler()	3662
7.1422.3.56	Vector62_handler()	3662
7.1422.3.57	Vector63_handler()	3662
7.1422.3.58	Vector64_handler()	3662

7.1422.3.62	Vector8_handler()	3663
7.1422.3.63	Vector9_handler()	3663
7.1423	vectors.c File Reference	3663
7.1423.1	Detailed Description	3664
7.1423.2	Variable Documentation	3664
7.1423.2.1	_vectab	3664
7.1424	vectors.c File Reference	3664
7.1424.1	Detailed Description	3669
7.1424.2	Macro Definition Documentation	3669
7.1424.2.1	VCT_USER_PROGRAM_VECTOR_TABLE_STARTADDR	3669
7.1424.3	Function Documentation	3669
7.1424.3.1	main()	3670
7.1424.3.2	reset_handler()	3670
7.1424.3.3	Vector0_handler()	3670
7.1424.3.4	Vector10_handler()	3671
7.1424.3.5	Vector11_handler()	3671
7.1424.3.6	Vector12_handler()	3671
7.1424.3.7	Vector13_handler()	3671
7.1424.3.8	Vector14_handler()	3672
7.1424.3.9	Vector15_handler()	3672
7.1424.3.10	Vector16_handler()	3672
7.1424.3.11	Vector17_handler()	3672
7.1424.3.12	Vector18_handler()	3673
7.1424.3.13	Vector19_handler()	3673
7.1424.3.14	Vector1_handler()	3673
7.1424.3.15	Vector20_handler()	3673
7.1424.3.16	Vector21_handler()	3674
7.1424.3.17	Vector22_handler()	3674
7.1424.3.18	Vector23_handler()	3674
7.1424.3.19	Vector24_handler()	3674

7.1424.3.20	Vector25_handler()	3675
7.1424.3.21	Vector26_handler()	3675
7.1424.3.22	Vector27_handler()	3675
7.1424.3.23	Vector28_handler()	3675
7.1424.3.24	Vector29_handler()	3676
7.1424.3.25	Vector30_handler()	3676
7.1424.3.26	Vector31_handler()	3676
7.1424.3.27	Vector32_handler()	3677
7.1424.3.28	Vector33_handler()	3677
7.1424.3.29	Vector34_handler()	3677
7.1424.3.30	Vector35_handler()	3677
7.1424.3.31	Vector36_handler()	3678
7.1424.3.32	Vector37_handler()	3678
7.1424.3.33	Vector38_handler()	3678
7.1424.3.34	Vector39_handler()	3678
7.1424.3.35	Vector40_handler()	3679
7.1424.3.36	Vector41_handler()	3679
7.1424.3.37	Vector42_handler()	3679
7.1424.3.38	Vector43_handler()	3680
7.1424.3.39	Vector44_handler()	3680
7.1424.3.40	Vector45_handler()	3680
7.1424.3.41	Vector46_handler()	3680
7.1424.3.42	Vector47_handler()	3681
7.1424.3.43	Vector48_handler()	3681
7.1424.3.44	Vector49_handler()	3681
7.1424.3.45	Vector50_handler()	3682
7.1424.3.46	Vector51_handler()	3682

7.1424.3.50	vector52_handler()	3682
7.1424.3.51	vector53_handler()	3682
7.1424.3.52	vector54_handler()	3683
7.1424.3.53	vector55_handler()	3683
7.1424.3.54	vector56_handler()	3683
7.1424.3.55	vector57_handler()	3683
7.1424.3.56	vector58_handler()	3684
7.1424.3.57	vector59_handler()	3684
7.1424.3.58	vector60_handler()	3684
7.1424.3.59	vector61_handler()	3685
7.1424.3.60	vector62_handler()	3685
7.1424.3.61	vector63_handler()	3685
7.1424.3.62	vector64_handler()	3686
7.1424.3.63	vector65_handler()	3686
7.1425	vectors.c File Reference	3686
7.1425.1	Detailed Description	3687
7.1425.2	Variable Documentation	3687
7.1425.2.1	vectab	3687
7.1426	xcp.c File Reference	3687
7.1426.1	Detailed Description	3689
7.1426.2	Function Documentation	3689
7.1426.2.1	XcpCmdBuildChecksum()	3689
7.1426.2.2	XcpCmdConnect()	3690
7.1426.2.3	XcpCmdDisconnect()	3690
7.1426.2.4	XcpCmdGetId()	3690
7.1426.2.5	XcpCmdGetSeed()	3691
7.1426.2.6	XcpCmdGetStatus()	3691
7.1426.2.7	XcpCmdProgram()	3691

7.1426.2.8XcpCmdProgramClear()	3692
7.1426.2.9XcpCmdProgramMax()	3692
7.1426.2.10XcpCmdProgramPrepare()	3693
7.1426.2.11XcpCmdProgramReset()	3693
7.1426.2.12XcpCmdProgramStart()	3693
7.1426.2.13XcpCmdSetMta()	3694
7.1426.2.14XcpCmdShortUpload()	3694
7.1426.2.15XcpCmdSynch()	3694
7.1426.2.16XcpCmdUnlock()	3695
7.1426.2.17XcpCmdUpload()	3695
7.1426.2.18XcpComputeChecksum()	3696
7.1426.2.19XcpGetSeed()	3696
7.1426.2.20XcpGetSeedHook()	3696
7.1426.2.21XcpInit()	3697
7.1426.2.22XcpIsConnected()	3697
7.1426.2.23XcpPacketReceived()	3697
7.1426.2.24XcpPacketTransmitted()	3698
7.1426.2.25XcpProtectResources()	3698
7.1426.2.26XcpSetCtoError()	3698
7.1426.2.27XcpTransmitPacket()	3699
7.1426.2.28XcpVerifyKey()	3699
7.1426.2.29XcpVerifyKeyHook()	3700
7.1427Xcp.h File Reference	3700
7.1427.1Detailed Description	3703
7.1427.2Macro Definition Documentation	3703
7.1427.2.1XCP_PACKET_RECEIVED_HOOK_EN	3704
7.1427.3Function Documentation	3704
7.1427.3.1XcpInit()	3704
7.1427.3.2XcpIsConnected()	3704
7.1427.3.3XcpPacketReceived()	3704
7.1427.3.4XcpPacketTransmitted()	3705

Chapter 1

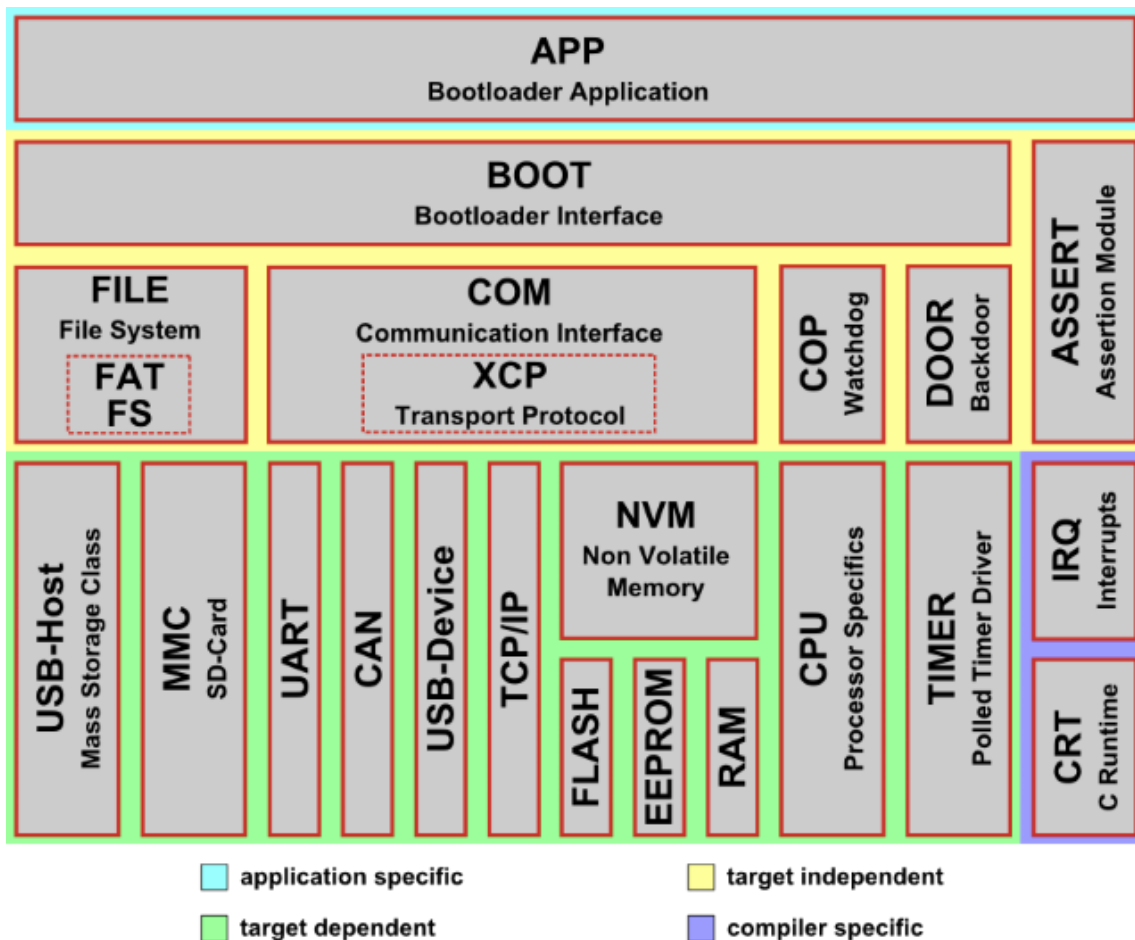
OpenBLT Firmware Documentation

1.1 Introduction

This documentation covers the OpenBLT (Open source BootLoader Tool) firmware. With OpenBLT you can make software updates through an on-chip communication interface (RS232, CAN, TCP/IP, USB etc.), without the need of specialized debugger hardware.

1.2 Software Architecture

The software program's architecture is divided into 4 major categories, namely the application code (App), target independent code (Core), target dependent code (Target), and compiler specific code (Comp).



To configure and fine-tune the bootloader for integration in your product, all you have to do is take the demo bootloader project for the microcontroller and compiler you are using, and (optionally) modify just the application code (App) to fit your needs. This typically involves changing the configuration header file (blt_conf.h) and the implementation of the hook functions (hooks.c).

For more in-depth information behind the design of the OpenBLT bootloader, you can visit: <https://www.feaser.com/openblt/doku.php?id=manual:design>.

1.3 Copyright and Licensing

C O P Y R I G H T

Copyright (c) by Feaser 2011-2021 <http://www.feaser.com> All rights reserved

L I C E N S E

This file is part of OpenBLT. OpenBLT is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

OpenBLT is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You have received a copy of the GNU General Public License along with OpenBLT. It should be located in ".\Doc\license.html". If not, contact Feaser to obtain a copy.

Chapter 2

Module Index

2.1 Modules

Here is a list of all modules:

Bootloader Demos	313
Template for demo programs	62
Bootloader	61
User Program	63
Demo for S32K118EVB/GCC	65
Bootloader	64
User Program	66
Demo for S32K118EVB/IAR	68
Bootloader	67
User Program	69
Demo for STM32F0-Discovery/STM32CubeIDE	71
Bootloader	70
User Program	72
Demo for STM32F0-Discovery/GCC	74
Bootloader	73
User Program	75
Demo for STM32F0-Discovery/IAR	77
Bootloader	76
User Program	78
Demo for STM32F0-Discovery/Keil	80
Bootloader	79
User Program	81
Demo for Nucleo-F091RC/STM32CubeIDE	83
Bootloader	82
User Program	84
Demo for Nucleo-F091RC/GCC	86
Bootloader	85
User Program	87
Demo for Nucleo-F091RC/IAR	89
Bootloader	88
User Program	90
Demo for Nucleo-F091RC/Keil	92
Bootloader	91
User Program	93

Demo for Nucleo-G071RB/STM32CubeIDE	95
Bootloader	94
User Program	96
Demo for Nucleo-G071RB/GCC	98
Bootloader	97
User Program	99
Demo for Nucleo-G071RB/IAR	101
Bootloader	100
User Program	102
Demo for Nucleo-G071RB/Keil	104
Bootloader	103
User Program	105
Demo for XMC1400 Boot Kit/GCC	107
Bootloader	106
User Program	108
Demo for XMC1400 Boot Kit/IAR	110
Bootloader	109
User Program	111
Demo for Nucleo-L552ZE/STM32CubeIDE	113
Bootloader	112
User Program	114
Demo for Nucleo-L552ZE/GCC	116
Bootloader	115
User Program	117
Demo for Nucleo-L552ZE/IAR	119
Bootloader	118
User Program	120
Demo for Nucleo-L552ZE/Keil	122
Bootloader	121
User Program	123
Demo for Olimex EM-32G880F128-STK/GCC	125
Bootloader	124
User Program	126
Demo for Olimex EM-32G880F128-STK/IAR	128
Bootloader	127
User Program	129
Demo for Texas Instruments EK-LM3S6965/GCC	131
Bootloader	130
User Program	132
Demo for Texas Instruments EK-LM3S6965/IAR	134
Bootloader	133
User Program	135
Demo for Texas Instruments EK-LM3S8962/GCC	137
Bootloader	136
User Program	138
Demo for Texas Instruments EK-LM3S8962/IAR	140
Bootloader	139
User Program	141
Demo for Nucleo-F103RB/STM32CubeIDE	143
Bootloader	142
User Program	144
Demo for Nucleo-F103RB/GCC	146
Bootloader	145
User Program	147
Demo for Nucleo-F103RB	149
Bootloader	148

User Program	150
Demo for Nucleo-F103RB/Keil	152
Bootloader	151
User Program	153
Demo for Olimex STM32-H103/STM32CubeIDE	155
Bootloader	154
User Program	156
Demo for Olimex STM32-H103/GCC	158
Bootloader	157
User Program	159
Demo for Olimex STM32-H103/IAR	161
Bootloader	160
User Program	162
Demo for Olimex STM32-H103/Keil	164
Bootloader	163
User Program	165
Demo for Olimex STM32-P103/STM32CubeIDE	167
Bootloader	166
User Program	168
Demo for Olimex STM32-P103/GCC	170
Bootloader	169
User Program	171
Demo for Olimex STM32-P103/IAR	173
Bootloader	172
User Program	174
Demo for Olimex STM32-P103/Keil	176
Bootloader	175
User Program	177
Demo for Olimexino-STM32/STM32CubeIDE	179
Bootloader	178
User Program	180
Demo for Olimexino-STM32/GCC	182
Bootloader	181
User Program	183
Demo for Olimexino-STM32/IAR	185
Bootloader	184
User Program	186
Demo for Olimexino-STM32/Keil	188
Bootloader	187
User Program	189
Demo for Olimex STM32-P207/STM32CubeIDE	191
Bootloader	190
User Program	192
Demo for Olimex STM32-P207/GCC	194
Bootloader	193
User Program	195
Demo for Olimex STM32-P207/IAR	197
Bootloader	196
User Program	198
Demo for Olimex STM32-P207/Keil	200
Bootloader	199
User Program	201
Demo for S32K144EVB/GCC	203
Bootloader	202
User Program	204
Demo for S32K144EVB/IAR	206

Bootloader	205
User Program	207
Demo for STM32F3-Discovery/STM32CubeIDE	209
Bootloader	208
User Program	210
Demo for STM32F3-Discovery/GCC	212
Bootloader	211
User Program	213
Demo for STM32F3-Discovery/IAR	215
Bootloader	214
User Program	216
Demo for STM32F3-Discovery/Keil	218
Bootloader	217
User Program	219
Demo for Nucleo-F303K8/STM32CubeIDE	221
Bootloader	220
User Program	222
Demo for Nucleo-F303K8/GCC	224
Bootloader	223
User Program	225
Demo for Nucleo-F303K8/IAR	227
Bootloader	226
User Program	228
Demo for Nucleo-F303K8/Keil	230
Bootloader	229
User Program	231
Demo for Nucleo-F429ZI/STM32CubeIDE	233
Bootloader	232
User Program	234
Demo for Nucleo-F429ZI/GCC	236
Bootloader	235
User Program	237
Demo for Nucleo-F429ZI/IAR	239
Bootloader	238
User Program	240
Demo for Nucleo-F429ZI/Keil	242
Bootloader	241
User Program	243
Demo for Olimex STM32-P405/STM32CubeIDE	245
Bootloader	244
User Program	246
Demo for Olimex STM32-P405/GCC	248
Bootloader	247
User Program	249
Demo for Olimex STM32-P405/IAR	251
Bootloader	250
User Program	252
Demo for Olimex STM32-P405/Keil	254
Bootloader	253
User Program	255
Demo for Nucleo-L476RG/STM32CubeIDE	257
Bootloader	256
User Program	258
Demo for Nucleo-L476RG/GCC	260
Bootloader	259
User Program	261

Demo for Nucleo-L476RG/IAR	263
Bootloader	262
User Program	264
Demo for Nucleo-L476RG/Keil	266
Bootloader	265
User Program	267
Demo for Texas Instruments DK-TM4C123G/IAR	269
Bootloader	268
User Program	270
Demo for XMC4700 Relax Kit/GCC	272
Bootloader	271
User Program	273
Demo for XMC4700 Relax Kit/IAR	275
Bootloader	274
User Program	276
Demo for Nucleo-F746ZG/STM32CubeIDE	278
Bootloader	277
User Program	279
Demo for Nucleo-F746ZG/GCC	281
Bootloader	280
User Program	282
Demo for Nucleo-F746ZG/IAR	284
Bootloader	283
User Program	285
Demo for Nucleo-F746ZG/Keil	287
Bootloader	286
User Program	288
Demo for Nucleo-F767ZI/STM32CubeIDE	290
Bootloader	289
User Program	291
Demo for Nucleo-F767ZI/GCC	293
Bootloader	292
User Program	294
Demo for Nucleo-F767ZI/IAR	296
Bootloader	295
User Program	297
Demo for Nucleo-F767ZI/Keil	299
Bootloader	298
User Program	300
Demo for Nucleo-H743ZI/STM32CubeIDE	302
Bootloader	301
User Program	303
Demo for Nucleo-H743ZI/GCC	305
Bootloader	304
User Program	306
Demo for Nucleo-H743ZI/IAR	308
Bootloader	307
User Program	309
Demo for Nucleo-H743ZI/Keil	311
Bootloader	310
User Program	312
Demo for NXP DevKit-S12G128/CodeWarrior	318
Bootloader	317
User Program	319
Demo for Dragon12-plus/CodeWarrior	321
Bootloader	320

User Program	322
Bootloader Core	350
Bootloader Ports	353
Target Port Template	329
CAN driver of a port	323
CPU driver of a port	324
Flash driver of a port	325
Compiler specifics of a port	326
Non-volatile memory driver of a port	327
RS232 UART driver of a port	328
Timer driver of a port	330
Type definitions of a port	331
USB driver of a port	332
Target ARMCM0 S32K11	333
Target ARMCM0 STM32F0	334
Target ARMCM0 STM32G0	335
Target ARMCM0 XMC1	336
Target ARMCM33 STM32L5	337
Target ARMCM3 EFM32	338
Target ARMCM3 LM3S	339
Target ARMCM3 STM32F1	340
Target ARMCM3 STM32F2	341
Target ARMCM4 S32K14	342
Target ARMCM4 STM32F3	343
Target ARMCM4 STM32F4	344
Target ARMCM4 STM32L4	345
Target ARMCM4 TM4C	346
Target ARMCM4 XMC4	347
Target ARMCM7 STM32F7	348
Target ARMCM7 STM32H7	349
Target HCS12	352

Chapter 3

Data Structure Index

3.1 Data Structures

Here are the data structures with brief descriptions:

tCanBusTiming	Structure type for grouping CAN bus timing related information	355
tCanRegs	Structure type with the layout of the CAN related control registers	358
tCanRxMsgSlot	Structure type with the layout of a CAN reception message slot	363
tCanTxMsgSlot	Structure type with the layout of a CAN transmit message slot	364
tFatFsObjects	Structure type for grouping FATFS related objects used by this module	365
tFifoCtrl	Structure type for fifo control	366
tFifoPipe	Structure type for a fifo pipe	368
tFileEraseInfo	Structure type with information for the memory erase opeartion	369
tFlashBlockInfo	Structure type for grouping flash block information	370
tFlashPrescalerSysclockMapping	Mapping table for finding the corect flash clock divider prescaler	371
tFlashRegs	Structure type for the flash control registers	372
tFlashSector	Flash sector descriptor type	375
tIsrFunc	Structure type for vector table entries	377
tSharedParamsBuffer	Layout of the shared parameters RAM buffer	378
tSrecLineParseObject	Structure type for grouping the parsing results of an S-record line	378
tSysTickRegs	Systick registers	380
tTimerRegs	Structure type with the layout of the timer related control registers	380
tXcpInfo	Structure type for grouping XCP internal module information	383

[uip_tcp_appstate_t](#)

Define the [uip_tcp_appstate_t](#) datatype. This is the state of our tcp/ip application, and the memory required for this state is allocated together with each TCP connection. One application state for each TCP connection [385](#)

Chapter 4

File Index

4.1 File List

Here is a list of all documented files with brief descriptions:

ARMCM0_STM32F0_Discovery_STM32F051_CubeIDE/Boot/App/app.c	
Bootloader application source file	387
ARMCM0_STM32F0_Discovery_STM32F051_CubeIDE/Prog/App/app.c	
User program application source file	388
ARMCM0_STM32F0_Nucleo_F091RC_CubeIDE/Boot/App/app.c	
Bootloader application source file	390
ARMCM0_STM32F0_Nucleo_F091RC_CubeIDE/Prog/App/app.c	
User program application source file	391
ARMCM0_STM32G0_Nucleo_G071RB_CubeIDE/Boot/App/app.c	
Bootloader application source file	392
ARMCM0_STM32G0_Nucleo_G071RB_CubeIDE/Prog/App/app.c	
User program application source file	394
ARMCM33_STM32L5_Nucleo_L552ZE_CubeIDE/Boot/App/app.c	
Bootloader application source file	395
ARMCM33_STM32L5_Nucleo_L552ZE_CubeIDE/Prog/App/app.c	
User program application source file	396
ARMCM3_STM32F1_Nucleo_F103RB_CubeIDE/Boot/App/app.c	
Bootloader application source file	397
ARMCM3_STM32F1_Nucleo_F103RB_CubeIDE/Prog/App/app.c	
User program application source file	399
ARMCM3_STM32F1_Olimex_STM32H103_CubeIDE/Boot/App/app.c	
Bootloader application source file	400
ARMCM3_STM32F1_Olimex_STM32H103_CubeIDE/Prog/App/app.c	
User program application source file	401
ARMCM3_STM32F1_Olimex_STM32P103_CubeIDE/Boot/App/app.c	
Bootloader application source file	403
ARMCM3_STM32F1_Olimex_STM32P103_CubeIDE/Prog/App/app.c	
User program application source file	404
ARMCM3_STM32F1_Olimexino_STM32_CubeIDE/Boot/App/app.c	
Bootloader application source file	405
ARMCM3_STM32F1_Olimexino_STM32_CubeIDE/Prog/App/app.c	
User program application source file	407
ARMCM3_STM32F2_Olimex_STM32P207_CubeIDE/Boot/App/app.c	
Bootloader application source file	408
ARMCM3_STM32F2_Olimex_STM32P207_CubeIDE/Prog/App/app.c	
User program application source file	409

ARMCM4_STM32F3_Discovery_F303VC_CubeIDE/Boot/App/app.c	
Bootloader application source file	410
ARMCM4_STM32F3_Discovery_F303VC_CubeIDE/Prog/App/app.c	
User program application source file	412
ARMCM4_STM32F3_Nucleo_F303K8_CubeIDE/Boot/App/app.c	
Bootloader application source file	413
ARMCM4_STM32F3_Nucleo_F303K8_CubeIDE/Prog/App/app.c	
User program application source file	414
ARMCM4_STM32F4_Nucleo_F429ZI_CubeIDE/Boot/App/app.c	
Bootloader application source file	416
ARMCM4_STM32F4_Nucleo_F429ZI_CubeIDE/Prog/App/app.c	
User program application source file	417
ARMCM4_STM32F4_Olimex_STM32P405_CubeIDE/Boot/App/app.c	
Bootloader application source file	418
ARMCM4_STM32F4_Olimex_STM32P405_CubeIDE/Prog/App/app.c	
User program application source file	420
ARMCM4_STM32L4_Nucleo_L476RG_CubeIDE/Boot/App/app.c	
Bootloader application source file	421
ARMCM4_STM32L4_Nucleo_L476RG_CubeIDE/Prog/App/app.c	
User program application source file	422
ARMCM7_STM32F7_Nucleo_F746ZG_CubeIDE/Boot/App/app.c	
Bootloader application source file	423
ARMCM7_STM32F7_Nucleo_F746ZG_CubeIDE/Prog/App/app.c	
User program application source file	425
ARMCM7_STM32F7_Nucleo_F767ZI_CubeIDE/Boot/App/app.c	
Bootloader application source file	426
ARMCM7_STM32F7_Nucleo_F767ZI_CubeIDE/Prog/App/app.c	
User program application source file	427
ARMCM7_STM32H7_Nucleo_H743ZI_CubeIDE/Boot/App/app.c	
Bootloader application source file	429
ARMCM7_STM32H7_Nucleo_H743ZI_CubeIDE/Prog/App/app.c	
User program application source file	430
ARMCM0_STM32F0_Discovery_STM32F051_CubeIDE/Boot/App/app.h	
Bootloader application header file	431
ARMCM0_STM32F0_Discovery_STM32F051_CubeIDE/Prog/App/app.h	
User program application header file	432
ARMCM0_STM32F0_Nucleo_F091RC_CubeIDE/Boot/App/app.h	
Bootloader application header file	434
ARMCM0_STM32F0_Nucleo_F091RC_CubeIDE/Prog/App/app.h	
User program application header file	435
ARMCM0_STM32G0_Nucleo_G071RB_CubeIDE/Boot/App/app.h	
Bootloader application header file	436
ARMCM0_STM32G0_Nucleo_G071RB_CubeIDE/Prog/App/app.h	
User program application header file	437
ARMCM33_STM32L5_Nucleo_L552ZE_CubeIDE/Boot/App/app.h	
Bootloader application header file	438
ARMCM33_STM32L5_Nucleo_L552ZE_CubeIDE/Prog/App/app.h	
User program application header file	440
ARMCM3_STM32F1_Nucleo_F103RB_CubeIDE/Boot/App/app.h	
Bootloader application header file	441
ARMCM3_STM32F1_Nucleo_F103RB_CubeIDE/Prog/App/app.h	
User program application header file	442
ARMCM3_STM32F1_Olimex_STM32H103_CubeIDE/Boot/App/app.h	
Bootloader application header file	443
ARMCM3_STM32F1_Olimex_STM32H103_CubeIDE/Prog/App/app.h	
User program application header file	445
ARMCM3_STM32F1_Olimex_STM32P103_CubeIDE/Boot/App/app.h	
Bootloader application header file	446

ARMCM3_STM32F1_Olimex_STM32P103_CubeIDE/Prog/App/app.h	
User program application header file	447
ARMCM3_STM32F1_Olimexino_STM32_CubeIDE/Boot/App/app.h	
Bootloader application header file	448
ARMCM3_STM32F1_Olimexino_STM32_CubeIDE/Prog/App/app.h	
User program application header file	450
ARMCM3_STM32F2_Olimex_STM32P207_CubeIDE/Boot/App/app.h	
Bootloader application header file	451
ARMCM3_STM32F2_Olimex_STM32P207_CubeIDE/Prog/App/app.h	
User program application header file	452
ARMCM4_STM32F3_Discovery_F303VC_CubeIDE/Boot/App/app.h	
Bootloader application header file	453
ARMCM4_STM32F3_Discovery_F303VC_CubeIDE/Prog/App/app.h	
User program application header file	455
ARMCM4_STM32F3_Nucleo_F303K8_CubeIDE/Boot/App/app.h	
Bootloader application header file	456
ARMCM4_STM32F3_Nucleo_F303K8_CubeIDE/Prog/App/app.h	
User program application header file	457
ARMCM4_STM32F4_Nucleo_F429ZI_CubeIDE/Boot/App/app.h	
Bootloader application header file	458
ARMCM4_STM32F4_Nucleo_F429ZI_CubeIDE/Prog/App/app.h	
User program application header file	460
ARMCM4_STM32F4_Olimex_STM32P405_CubeIDE/Boot/App/app.h	
Bootloader application header file	461
ARMCM4_STM32F4_Olimex_STM32P405_CubeIDE/Prog/App/app.h	
User program application header file	462
ARMCM4_STM32L4_Nucleo_L476RG_CubeIDE/Boot/App/app.h	
Bootloader application header file	463
ARMCM4_STM32L4_Nucleo_L476RG_CubeIDE/Prog/App/app.h	
User program application header file	465
ARMCM7_STM32F7_Nucleo_F746ZG_CubeIDE/Boot/App/app.h	
Bootloader application header file	466
ARMCM7_STM32F7_Nucleo_F746ZG_CubeIDE/Prog/App/app.h	
User program application header file	467
ARMCM7_STM32F7_Nucleo_F767ZI_CubeIDE/Boot/App/app.h	
Bootloader application header file	468
ARMCM7_STM32F7_Nucleo_F767ZI_CubeIDE/Prog/App/app.h	
User program application header file	470
ARMCM7_STM32H7_Nucleo_H743ZI_CubeIDE/Boot/App/app.h	
Bootloader application header file	471
ARMCM7_STM32H7_Nucleo_H743ZI_CubeIDE/Prog/App/app.h	
User program application header file	472
asserts.c	
Bootloader assertion module source file	474
asserts.h	
Bootloader assertion module header file	475
backdoor.c	
Bootloader backdoor entry source file	476
backdoor.h	
Bootloader backdoor entry header file	478
_template/Boot/blt_conf.h	
Bootloader configuration header file	479
ARMCM0_STM32F0_Discovery_STM32F051_CubeIDE/Boot/App/blt_conf.h	
Bootloader configuration header file	480
ARMCM0_STM32F0_Discovery_STM32F051_GCC/Boot/blt_conf.h	
Bootloader configuration header file	482
ARMCM0_STM32F0_Discovery_STM32F051_IAR/Boot/blt_conf.h	
Bootloader configuration header file	483

ARMCM0_STM32F0_Discovery_STM32F051_Keil/Boot/blt_conf.h	
Bootloader configuration header file	484
ARMCM0_STM32F0_Nucleo_F091RC_CubeIDE/Boot/App/blt_conf.h	
Bootloader configuration header file	486
ARMCM0_STM32F0_Nucleo_F091RC_GCC/Boot/blt_conf.h	
Bootloader configuration header file	487
ARMCM0_STM32F0_Nucleo_F091RC_IAR/Boot/blt_conf.h	
Bootloader configuration header file	489
ARMCM0_STM32F0_Nucleo_F091RC_Keil/Boot/blt_conf.h	
Bootloader configuration header file	490
ARMCM0_STM32G0_Nucleo_G071RB_CubeIDE/Boot/App/blt_conf.h	
Bootloader configuration header file	492
ARMCM0_STM32G0_Nucleo_G071RB_GCC/Boot/blt_conf.h	
Bootloader configuration header file	493
ARMCM0_STM32G0_Nucleo_G071RB_IAR/Boot/blt_conf.h	
Bootloader configuration header file	494
ARMCM0_STM32G0_Nucleo_G071RB_Keil/Boot/blt_conf.h	
Bootloader configuration header file	495
ARMCM0_XMC1_XMC1400_Boot_Kit_GCC/Boot/blt_conf.h	
Bootloader configuration header file	496
ARMCM0_XMC1_XMC1400_Boot_Kit_IAR/Boot/blt_conf.h	
Bootloader configuration header file	498
ARMCM33_STM32L5_Nucleo_L552ZE_CubeIDE/Boot/App/blt_conf.h	
Bootloader configuration header file	500
ARMCM33_STM32L5_Nucleo_L552ZE_GCC/Boot/blt_conf.h	
Bootloader configuration header file	501
ARMCM33_STM32L5_Nucleo_L552ZE_IAR/Boot/blt_conf.h	
Bootloader configuration header file	503
ARMCM33_STM32L5_Nucleo_L552ZE_Keil/Boot/blt_conf.h	
Bootloader configuration header file	504
ARMCM3_EFM32_Olimex_EM32G880F128STK_GCC/Boot/blt_conf.h	
Bootloader configuration header file	506
ARMCM3_EFM32_Olimex_EM32G880F128STK_IAR/Boot/blt_conf.h	
Bootloader configuration header file	507
ARMCM3_LM3S_EK_LM3S6965_GCC/Boot/blt_conf.h	
Bootloader configuration header file	509
ARMCM3_LM3S_EK_LM3S6965_IAR/Boot/blt_conf.h	
Bootloader configuration header file	511
ARMCM3_LM3S_EK_LM3S8962_GCC/Boot/blt_conf.h	
Bootloader configuration header file	513
ARMCM3_LM3S_EK_LM3S8962_IAR/Boot/blt_conf.h	
Bootloader configuration header file	514
ARMCM3_STM32F1_Nucleo_F103RB_CubeIDE/Boot/App/blt_conf.h	
Bootloader configuration header file	516
ARMCM3_STM32F1_Nucleo_F103RB_GCC/Boot/blt_conf.h	
Bootloader configuration header file	517
ARMCM3_STM32F1_Nucleo_F103RB_IAR/Boot/blt_conf.h	
Bootloader configuration header file	518
ARMCM3_STM32F1_Nucleo_F103RB_Keil/Boot/blt_conf.h	
Bootloader configuration header file	520
ARMCM3_STM32F1_Olimex_STM32H103_CubeIDE/Boot/App/blt_conf.h	
Bootloader configuration header file	521
ARMCM3_STM32F1_Olimex_STM32H103_GCC/Boot/blt_conf.h	
Bootloader configuration header file	522
ARMCM3_STM32F1_Olimex_STM32H103_IAR/Boot/blt_conf.h	
Bootloader configuration header file	523
ARMCM3_STM32F1_Olimex_STM32H103_Keil/Boot/blt_conf.h	
Bootloader configuration header file	525

ARMCM3_STM32F1_Olimex_STM32P103_CubeIDE/Boot/App/blt_conf.h	
Bootloader configuration header file	526
ARMCM3_STM32F1_Olimex_STM32P103_GCC/Boot/blt_conf.h	
Bootloader configuration header file	528
ARMCM3_STM32F1_Olimex_STM32P103_IAR/Boot/blt_conf.h	
Bootloader configuration header file	530
ARMCM3_STM32F1_Olimex_STM32P103_Keil/Boot/blt_conf.h	
Bootloader configuration header file	531
ARMCM3_STM32F1_Olimexino_STM32_CubeIDE/Boot/App/blt_conf.h	
Bootloader configuration header file	533
ARMCM3_STM32F1_Olimexino_STM32_GCC/Boot/blt_conf.h	
Bootloader configuration header file	535
ARMCM3_STM32F1_Olimexino_STM32_IAR/Boot/blt_conf.h	
Bootloader configuration header file	537
ARMCM3_STM32F1_Olimexino_STM32_Keil/Boot/blt_conf.h	
Bootloader configuration header file	539
ARMCM3_STM32F2_Olimex_STM32P207_CubeIDE/Boot/App/blt_conf.h	
Bootloader configuration header file	541
ARMCM3_STM32F2_Olimex_STM32P207_GCC/Boot/blt_conf.h	
Bootloader configuration header file	543
ARMCM3_STM32F2_Olimex_STM32P207_IAR/Boot/blt_conf.h	
Bootloader configuration header file	545
ARMCM3_STM32F2_Olimex_STM32P207_Keil/Boot/blt_conf.h	
Bootloader configuration header file	546
ARMCM4_S32K14_S32K144EVB_GCC/Boot/blt_conf.h	
Bootloader configuration header file	548
ARMCM4_S32K14_S32K144EVB_IAR/Boot/blt_conf.h	
Bootloader configuration header file	550
ARMCM4_STM32F3_Discovery_F303VC_CubeIDE/Boot/App/blt_conf.h	
Bootloader configuration header file	551
ARMCM4_STM32F3_Discovery_F303VC_GCC/Boot/blt_conf.h	
Bootloader configuration header file	552
ARMCM4_STM32F3_Discovery_F303VC_IAR/Boot/blt_conf.h	
Bootloader configuration header file	553
ARMCM4_STM32F3_Discovery_F303VC_Keil/Boot/blt_conf.h	
Bootloader configuration header file	554
ARMCM4_STM32F3_Nucleo_F303K8_CubeIDE/Boot/App/blt_conf.h	
Bootloader configuration header file	555
ARMCM4_STM32F3_Nucleo_F303K8_GCC/Boot/blt_conf.h	
Bootloader configuration header file	557
ARMCM4_STM32F3_Nucleo_F303K8_IAR/Boot/blt_conf.h	
Bootloader configuration header file	558
ARMCM4_STM32F3_Nucleo_F303K8_Keil/Boot/blt_conf.h	
Bootloader configuration header file	560
ARMCM4_STM32F4_Nucleo_F429ZI_CubeIDE/Boot/App/blt_conf.h	
Bootloader configuration header file	561
ARMCM4_STM32F4_Nucleo_F429ZI_GCC/Boot/blt_conf.h	
Bootloader configuration header file	564
ARMCM4_STM32F4_Nucleo_F429ZI_IAR/Boot/blt_conf.h	
Bootloader configuration header file	566
ARMCM4_STM32F4_Nucleo_F429ZI_Keil/Boot/blt_conf.h	
Bootloader configuration header file	569
ARMCM4_STM32F4_Olimex_STM32P405_CubeIDE/Boot/App/blt_conf.h	
Bootloader configuration header file	571
ARMCM4_STM32F4_Olimex_STM32P405_GCC/Boot/blt_conf.h	
Bootloader configuration header file	573
ARMCM4_STM32F4_Olimex_STM32P405_IAR/Boot/blt_conf.h	
Bootloader configuration header file	575

ARMCM4_STM32F4_Olimex_STM32P405_Keil/Boot/blt_conf.h	
Bootloader configuration header file	577
ARMCM4_STM32L4_Nucleo_L476RG_CubelIDE/Boot/App/blt_conf.h	
Bootloader configuration header file	578
ARMCM4_STM32L4_Nucleo_L476RG_GCC/Boot/blt_conf.h	
Bootloader configuration header file	580
ARMCM4_STM32L4_Nucleo_L476RG_IAR/Boot/blt_conf.h	
Bootloader configuration header file	582
ARMCM4_STM32L4_Nucleo_L476RG_Keil/Boot/blt_conf.h	
Bootloader configuration header file	583
ARMCM4_TM4C_DK_TM4C123G_IAR/Boot/blt_conf.h	
Bootloader configuration header file	585
ARMCM4_XMC4_XMC4700_Relax_Kit_GCC/Boot/blt_conf.h	
Bootloader configuration header file	586
ARMCM4_XMC4_XMC4700_Relax_Kit_IAR/Boot/blt_conf.h	
Bootloader configuration header file	589
ARMCM7_STM32F7_Nucleo_F746ZG_CubelIDE/Boot/App/blt_conf.h	
Bootloader configuration header file	591
ARMCM7_STM32F7_Nucleo_F746ZG_GCC/Boot/blt_conf.h	
Bootloader configuration header file	593
ARMCM7_STM32F7_Nucleo_F746ZG_IAR/Boot/blt_conf.h	
Bootloader configuration header file	594
ARMCM7_STM32F7_Nucleo_F746ZG_Keil/Boot/blt_conf.h	
Bootloader configuration header file	596
ARMCM7_STM32F7_Nucleo_F767ZI_CubelIDE/Boot/App/blt_conf.h	
Bootloader configuration header file	598
ARMCM7_STM32F7_Nucleo_F767ZI_GCC/Boot/blt_conf.h	
Bootloader configuration header file	600
ARMCM7_STM32F7_Nucleo_F767ZI_IAR/Boot/blt_conf.h	
Bootloader configuration header file	602
ARMCM7_STM32F7_Nucleo_F767ZI_Keil/Boot/blt_conf.h	
Bootloader configuration header file	604
ARMCM7_STM32H7_Nucleo_H743ZI_CubelIDE/Boot/App/blt_conf.h	
Bootloader configuration header file	606
ARMCM7_STM32H7_Nucleo_H743ZI_GCC/Boot/blt_conf.h	
Bootloader configuration header file	607
ARMCM7_STM32H7_Nucleo_H743ZI_IAR/Boot/blt_conf.h	
Bootloader configuration header file	609
ARMCM7_STM32H7_Nucleo_H743ZI_Keil/Boot/blt_conf.h	
Bootloader configuration header file	611
HCS12_DevKit_S12G128_CodeWarrior/Boot/blt_conf.h	
Bootloader configuration header file	612
HCS12_Evbplus_Dragon12p_CodeWarrior/Boot/blt_conf.h	
Bootloader configuration header file	614
Demo/_template/Prog/boot.c	
Demo program bootloader interface source file	615
Demo/ARMCM0_STM32F0_Discovery_STM32F051_CubelIDE/Prog/App/boot.c	
Demo program bootloader interface source file	619
Demo/ARMCM0_STM32F0_Discovery_STM32F051_GCC/Prog/boot.c	
Demo program bootloader interface source file	622
Demo/ARMCM0_STM32F0_Discovery_STM32F051_IAR/Prog/boot.c	
Demo program bootloader interface source file	625
Demo/ARMCM0_STM32F0_Discovery_STM32F051_Keil/Prog/boot.c	
Demo program bootloader interface source file	628
Demo/ARMCM0_STM32F0_Nucleo_F091RC_CubelIDE/Prog/App/boot.c	
Demo program bootloader interface source file	631
Demo/ARMCM0_STM32F0_Nucleo_F091RC_GCC/Prog/boot.c	
Demo program bootloader interface source file	636

Demo/ARMCM0_STM32F0_Nucleo_F091RC_IAR/Prog/boot.c	
Demo program bootloader interface source file	641
Demo/ARMCM0_STM32F0_Nucleo_F091RC_Keil/Prog/boot.c	
Demo program bootloader interface source file	646
Demo/ARMCM0_STM32G0_Nucleo_G071RB_CubeIDE/Prog/App/boot.c	
Demo program bootloader interface source file	651
Demo/ARMCM0_STM32G0_Nucleo_G071RB_GCC/Prog/boot.c	
Demo program bootloader interface source file	654
Demo/ARMCM0_STM32G0_Nucleo_G071RB_IAR/Prog/boot.c	
Demo program bootloader interface source file	657
Demo/ARMCM0_STM32G0_Nucleo_G071RB_Keil/Prog/boot.c	
Demo program bootloader interface source file	660
Demo/ARMCM0_XMC1_XMC1400_Boot_Kit_GCC/Prog/boot.c	
Demo program bootloader interface source file	663
Demo/ARMCM0_XMC1_XMC1400_Boot_Kit_IAR/Prog/boot.c	
Demo program bootloader interface source file	666
Demo/ARMCM33_STM32L5_Nucleo_L552ZE_CubeIDE/Prog/App/boot.c	
Demo program bootloader interface source file	670
Demo/ARMCM33_STM32L5_Nucleo_L552ZE_GCC/Prog/boot.c	
Demo program bootloader interface source file	675
Demo/ARMCM33_STM32L5_Nucleo_L552ZE_IAR/Prog/boot.c	
Demo program bootloader interface source file	680
Demo/ARMCM33_STM32L5_Nucleo_L552ZE_Keil/Prog/boot.c	
Demo program bootloader interface source file	685
Demo/ARMCM3_EFM32_Olimex_EM32G880F128STK_GCC/Prog/boot.c	
Demo program bootloader interface source file	690
Demo/ARMCM3_EFM32_Olimex_EM32G880F128STK_IAR/Prog/boot.c	
Demo program bootloader interface source file	693
Demo/ARMCM3_LM3S_EK_LM3S6965_GCC/Prog/boot.c	
Demo program bootloader interface source file	696
Demo/ARMCM3_LM3S_EK_LM3S6965_IAR/Prog/boot.c	
Demo program bootloader interface source file	698
Demo/ARMCM3_LM3S_EK_LM3S8962_GCC/Prog/boot.c	
Demo program bootloader interface source file	701
Demo/ARMCM3_LM3S_EK_LM3S8962_IAR/Prog/boot.c	
Demo program bootloader interface source file	705
Demo/ARMCM3_STM32F1_Nucleo_F103RB_CubeIDE/Prog/App/boot.c	
Demo program bootloader interface source file	708
Demo/ARMCM3_STM32F1_Nucleo_F103RB_GCC/Prog/boot.c	
Demo program bootloader interface source file	712
Demo/ARMCM3_STM32F1_Nucleo_F103RB_IAR/Prog/boot.c	
Demo program bootloader interface source file	715
Demo/ARMCM3_STM32F1_Nucleo_F103RB_Keil/Prog/boot.c	
Demo program bootloader interface source file	718
Demo/ARMCM3_STM32F1_Olimex_STM32P103_CubeIDE/Prog/App/boot.c	
Demo program bootloader interface source file	721
Demo/ARMCM3_STM32F1_Olimex_STM32P103_GCC/Prog/boot.c	
Demo program bootloader interface source file	726
Demo/ARMCM3_STM32F1_Olimex_STM32P103_IAR/Prog/boot.c	
Demo program bootloader interface source file	731
Demo/ARMCM3_STM32F1_Olimex_STM32P103_Keil/Prog/boot.c	
Demo program bootloader interface source file	736
Demo/ARMCM3_STM32F1_Olimexino_STM32_CubeIDE/Prog/App/boot.c	
Demo program bootloader interface source file	741
Demo/ARMCM3_STM32F1_Olimexino_STM32_GCC/Prog/boot.c	
Demo program bootloader interface source file	744
Demo/ARMCM3_STM32F1_Olimexino_STM32_IAR/Prog/boot.c	
Demo program bootloader interface source file	748

Demo/ARMCM3_STM32F1_Olimexino_STM32_Keil/Prog/boot.c	
Demo program bootloader interface source file	752
Demo/ARMCM3_STM32F2_Olimex_STM32P207_CubelIDE/Prog/App/boot.c	
Demo program bootloader interface source file	756
Demo/ARMCM3_STM32F2_Olimex_STM32P207_GCC/Prog/boot.c	
Demo program bootloader interface source file	761
Demo/ARMCM3_STM32F2_Olimex_STM32P207_IAR/Prog/boot.c	
Demo program bootloader interface source file	766
Demo/ARMCM3_STM32F2_Olimex_STM32P207_Keil/Prog/boot.c	
Demo program bootloader interface source file	771
Demo/ARMCM4_S32K14_S32K144EVB_GCC/Prog/boot.c	
Demo program bootloader interface source file	776
Demo/ARMCM4_S32K14_S32K144EVB_IAR/Prog/boot.c	
Demo program bootloader interface source file	783
Demo/ARMCM4_STM32F3_Nucleo_F303K8_CubelIDE/Prog/App/boot.c	
Demo program bootloader interface source file	790
Demo/ARMCM4_STM32F3_Nucleo_F303K8_GCC/Prog/boot.c	
Demo program bootloader interface source file	795
Demo/ARMCM4_STM32F3_Nucleo_F303K8_IAR/Prog/boot.c	
Demo program bootloader interface source file	800
Demo/ARMCM4_STM32F3_Nucleo_F303K8_Keil/Prog/boot.c	
Demo program bootloader interface source file	805
Demo/ARMCM4_STM32F4_Nucleo_F429ZI_CubelIDE/Prog/App/boot.c	
Demo program bootloader interface source file	810
Demo/ARMCM4_STM32F4_Nucleo_F429ZI_GCC/Prog/boot.c	
Demo program bootloader interface source file	815
Demo/ARMCM4_STM32F4_Nucleo_F429ZI_IAR/Prog/boot.c	
Demo program bootloader interface source file	820
Demo/ARMCM4_STM32F4_Nucleo_F429ZI_Keil/Prog/boot.c	
Demo program bootloader interface source file	825
Demo/ARMCM4_STM32F4_Olimex_STM32P405_CubelIDE/Prog/App/boot.c	
Demo program bootloader interface source file	830
Demo/ARMCM4_STM32F4_Olimex_STM32P405_GCC/Prog/boot.c	
Demo program bootloader interface source file	835
Demo/ARMCM4_STM32F4_Olimex_STM32P405_IAR/Prog/boot.c	
Demo program bootloader interface source file	840
Demo/ARMCM4_STM32F4_Olimex_STM32P405_Keil/Prog/boot.c	
Demo program bootloader interface source file	845
Demo/ARMCM4_STM32L4_Nucleo_L476RG_CubelIDE/Prog/App/boot.c	
Demo program bootloader interface source file	850
Demo/ARMCM4_STM32L4_Nucleo_L476RG_GCC/Prog/boot.c	
Demo program bootloader interface source file	855
Demo/ARMCM4_STM32L4_Nucleo_L476RG_IAR/Prog/boot.c	
Demo program bootloader interface source file	860
Demo/ARMCM4_STM32L4_Nucleo_L476RG_Keil/Prog/boot.c	
Demo program bootloader interface source file	865
Demo/ARMCM4_TM4C_DK_TM4C123G_IAR/Prog/boot.c	
Demo program bootloader interface source file	869
Demo/ARMCM4_XMC4_XMC4700_Relax_Kit_GCC/Prog/boot.c	
Demo program bootloader interface source file	872
Demo/ARMCM4_XMC4_XMC4700_Relax_Kit_IAR/Prog/boot.c	
Demo program bootloader interface source file	875
Demo/ARMCM7_STM32F7_Nucleo_F746ZG_CubelIDE/Prog/App/boot.c	
Demo program bootloader interface source file	879
Demo/ARMCM7_STM32F7_Nucleo_F746ZG_GCC/Prog/boot.c	
Demo program bootloader interface source file	884
Demo/ARMCM7_STM32F7_Nucleo_F746ZG_IAR/Prog/boot.c	
Demo program bootloader interface source file	889

Demo/ARMCM7_STM32F7_Nucleo_F746ZG_Keil/Prog/boot.c	
Demo program bootloader interface source file	894
Demo/ARMCM7_STM32F7_Nucleo_F767ZI_CubelIDE/Prog/App/boot.c	
Demo program bootloader interface source file	899
Demo/ARMCM7_STM32F7_Nucleo_F767ZI_GCC/Prog/boot.c	
Demo program bootloader interface source file	902
Demo/ARMCM7_STM32F7_Nucleo_F767ZI_IAR/Prog/boot.c	
Demo program bootloader interface source file	905
Demo/ARMCM7_STM32F7_Nucleo_F767ZI_Keil/Prog/boot.c	
Demo program bootloader interface source file	908
Demo/ARMCM7_STM32H7_Nucleo_H743ZI_CubelIDE/Prog/App/boot.c	
Demo program bootloader interface source file	911
Demo/ARMCM7_STM32H7_Nucleo_H743ZI_GCC/Prog/boot.c	
Demo program bootloader interface source file	916
Demo/ARMCM7_STM32H7_Nucleo_H743ZI_IAR/Prog/boot.c	
Demo program bootloader interface source file	921
Demo/ARMCM7_STM32H7_Nucleo_H743ZI_Keil/Prog/boot.c	
Demo program bootloader interface source file	926
Demo/HCS12_Evbplus_Dragon12p_CodeWarrior/Prog/boot.c	
Demo program bootloader interface source file	931
Source/boot.c	
Bootloader core module source file	936
Demo/_template/Prog/boot.h	
Demo program bootloader interface header file	937
Demo/ARMCM0_STM32F0_Discovery_STM32F051_CubelIDE/Prog/App/boot.h	
Demo program bootloader interface header file	939
Demo/ARMCM0_STM32F0_Discovery_STM32F051_GCC/Prog/boot.h	
Demo program bootloader interface header file	941
Demo/ARMCM0_STM32F0_Discovery_STM32F051_IAR/Prog/boot.h	
Demo program bootloader interface header file	942
Demo/ARMCM0_STM32F0_Discovery_STM32F051_Keil/Prog/boot.h	
Demo program bootloader interface header file	944
Demo/ARMCM0_STM32F0_Nucleo_F091RC_CubelIDE/Prog/App/boot.h	
Demo program bootloader interface header file	946
Demo/ARMCM0_STM32F0_Nucleo_F091RC_GCC/Prog/boot.h	
Demo program bootloader interface header file	947
Demo/ARMCM0_STM32F0_Nucleo_F091RC_IAR/Prog/boot.h	
Demo program bootloader interface header file	949
Demo/ARMCM0_STM32F0_Nucleo_F091RC_Keil/Prog/boot.h	
Demo program bootloader interface header file	951
Demo/ARMCM0_STM32G0_Nucleo_G071RB_CubelIDE/Prog/App/boot.h	
Demo program bootloader interface header file	952
Demo/ARMCM0_STM32G0_Nucleo_G071RB_GCC/Prog/boot.h	
Demo program bootloader interface header file	954
Demo/ARMCM0_STM32G0_Nucleo_G071RB_IAR/Prog/boot.h	
Demo program bootloader interface header file	956
Demo/ARMCM0_STM32G0_Nucleo_G071RB_Keil/Prog/boot.h	
Demo program bootloader interface header file	957
Demo/ARMCM0_XMC1_XMC1400_Boot_Kit_GCC/Prog/boot.h	
Demo program bootloader interface header file	959
Demo/ARMCM0_XMC1_XMC1400_Boot_Kit_IAR/Prog/boot.h	
Demo program bootloader interface header file	961
Demo/ARMCM33_STM32L5_Nucleo_L552ZE_CubelIDE/Prog/App/boot.h	
Demo program bootloader interface header file	962
Demo/ARMCM33_STM32L5_Nucleo_L552ZE_GCC/Prog/boot.h	
Demo program bootloader interface header file	964
Demo/ARMCM33_STM32L5_Nucleo_L552ZE_IAR/Prog/boot.h	
Demo program bootloader interface header file	966

Demo/ARMCM33_STM32L5_Nucleo_L552ZE_Keil/Prog/boot.h	
Demo program bootloader interface header file	967
Demo/ARMCM3_EFM32_Olimex_EM32G880F128STK_GCC/Prog/boot.h	
Demo program bootloader interface header file	969
Demo/ARMCM3_EFM32_Olimex_EM32G880F128STK_IAR/Prog/boot.h	
Demo program bootloader interface header file	970
Demo/ARMCM3_LM3S_EK_LM3S6965_GCC/Prog/boot.h	
Demo program bootloader interface header file	972
Demo/ARMCM3_LM3S_EK_LM3S6965_IAR/Prog/boot.h	
Demo program bootloader interface header file	973
Demo/ARMCM3_LM3S_EK_LM3S8962_GCC/Prog/boot.h	
Demo program bootloader interface header file	975
Demo/ARMCM3_LM3S_EK_LM3S8962_IAR/Prog/boot.h	
Demo program bootloader interface header file	976
Demo/ARMCM3_STM32F1_Nucleo_F103RB_CubeIDE/Prog/App/boot.h	
Demo program bootloader interface header file	978
Demo/ARMCM3_STM32F1_Nucleo_F103RB_GCC/Prog/boot.h	
Demo program bootloader interface header file	980
Demo/ARMCM3_STM32F1_Nucleo_F103RB_IAR/Prog/boot.h	
Demo program bootloader interface header file	981
Demo/ARMCM3_STM32F1_Nucleo_F103RB_Keil/Prog/boot.h	
Demo program bootloader interface header file	983
Demo/ARMCM3_STM32F1_Olimex_STM32P103_CubeIDE/Prog/App/boot.h	
Demo program bootloader interface header file	985
Demo/ARMCM3_STM32F1_Olimex_STM32P103_GCC/Prog/boot.h	
Demo program bootloader interface header file	986
Demo/ARMCM3_STM32F1_Olimex_STM32P103_IAR/Prog/boot.h	
Demo program bootloader interface header file	988
Demo/ARMCM3_STM32F1_Olimex_STM32P103_Keil/Prog/boot.h	
Demo program bootloader interface header file	990
Demo/ARMCM3_STM32F1_Olimexino_STM32_CubeIDE/Prog/App/boot.h	
Demo program bootloader interface header file	991
Demo/ARMCM3_STM32F1_Olimexino_STM32_GCC/Prog/boot.h	
Demo program bootloader interface header file	993
Demo/ARMCM3_STM32F1_Olimexino_STM32_IAR/Prog/boot.h	
Demo program bootloader interface header file	995
Demo/ARMCM3_STM32F1_Olimexino_STM32_Keil/Prog/boot.h	
Demo program bootloader interface header file	996
Demo/ARMCM3_STM32F2_Olimex_STM32P207_CubeIDE/Prog/App/boot.h	
Demo program bootloader interface header file	998
Demo/ARMCM3_STM32F2_Olimex_STM32P207_GCC/Prog/boot.h	
Demo program bootloader interface header file	1000
Demo/ARMCM3_STM32F2_Olimex_STM32P207_IAR/Prog/boot.h	
Demo program bootloader interface header file	1001
Demo/ARMCM3_STM32F2_Olimex_STM32P207_Keil/Prog/boot.h	
Demo program bootloader interface header file	1003
Demo/ARMCM4_S32K14_S32K144EVB_GCC/Prog/boot.h	
Demo program bootloader interface header file	1005
Demo/ARMCM4_S32K14_S32K144EVB_IAR/Prog/boot.h	
Demo program bootloader interface header file	1006
Demo/ARMCM4_STM32F3_Nucleo_F303K8_CubeIDE/Prog/App/boot.h	
Demo program bootloader interface header file	1008
Demo/ARMCM4_STM32F3_Nucleo_F303K8_GCC/Prog/boot.h	
Demo program bootloader interface header file	1010
Demo/ARMCM4_STM32F3_Nucleo_F303K8_IAR/Prog/boot.h	
Demo program bootloader interface header file	1011
Demo/ARMCM4_STM32F3_Nucleo_F303K8_Keil/Prog/boot.h	
Demo program bootloader interface header file	1013

Demo/ARMCM4_STM32F4_Nucleo_F429ZI_CubeIDE/Prog/App/boot.h	
Demo program bootloader interface header file	1015
Demo/ARMCM4_STM32F4_Nucleo_F429ZI_GCC/Prog/boot.h	
Demo program bootloader interface header file	1016
Demo/ARMCM4_STM32F4_Nucleo_F429ZI_IAR/Prog/boot.h	
Demo program bootloader interface header file	1018
Demo/ARMCM4_STM32F4_Nucleo_F429ZI_Keil/Prog/boot.h	
Demo program bootloader interface header file	1020
Demo/ARMCM4_STM32F4_Olimex_STM32P405_CubeIDE/Prog/App/boot.h	
Demo program bootloader interface header file	1021
Demo/ARMCM4_STM32F4_Olimex_STM32P405_GCC/Prog/boot.h	
Demo program bootloader interface header file	1023
Demo/ARMCM4_STM32F4_Olimex_STM32P405_IAR/Prog/boot.h	
Demo program bootloader interface header file	1025
Demo/ARMCM4_STM32F4_Olimex_STM32P405_Keil/Prog/boot.h	
Demo program bootloader interface header file	1026
Demo/ARMCM4_STM32L4_Nucleo_L476RG_CubeIDE/Prog/App/boot.h	
Demo program bootloader interface header file	1028
Demo/ARMCM4_STM32L4_Nucleo_L476RG_GCC/Prog/boot.h	
Demo program bootloader interface header file	1030
Demo/ARMCM4_STM32L4_Nucleo_L476RG_IAR/Prog/boot.h	
Demo program bootloader interface header file	1031
Demo/ARMCM4_STM32L4_Nucleo_L476RG_Keil/Prog/boot.h	
Demo program bootloader interface header file	1033
Demo/ARMCM4_TM4C_DK_TM4C123G_IAR/Prog/boot.h	
Demo program bootloader interface header file	1035
Demo/ARMCM4_XMC4_XMC4700_Relax_Kit_GCC/Prog/boot.h	
Demo program bootloader interface header file	1036
Demo/ARMCM4_XMC4_XMC4700_Relax_Kit_IAR/Prog/boot.h	
Demo program bootloader interface header file	1038
Demo/ARMCM7_STM32F7_Nucleo_F746ZG_CubeIDE/Prog/App/boot.h	
Demo program bootloader interface header file	1040
Demo/ARMCM7_STM32F7_Nucleo_F746ZG_GCC/Prog/boot.h	
Demo program bootloader interface header file	1041
Demo/ARMCM7_STM32F7_Nucleo_F746ZG_IAR/Prog/boot.h	
Demo program bootloader interface header file	1043
Demo/ARMCM7_STM32F7_Nucleo_F746ZG_Keil/Prog/boot.h	
Demo program bootloader interface header file	1045
Demo/ARMCM7_STM32F7_Nucleo_F767ZI_CubeIDE/Prog/App/boot.h	
Demo program bootloader interface header file	1046
Demo/ARMCM7_STM32F7_Nucleo_F767ZI_GCC/Prog/boot.h	
Demo program bootloader interface header file	1048
Demo/ARMCM7_STM32F7_Nucleo_F767ZI_IAR/Prog/boot.h	
Demo program bootloader interface header file	1050
Demo/ARMCM7_STM32F7_Nucleo_F767ZI_Keil/Prog/boot.h	
Demo program bootloader interface header file	1051
Demo/ARMCM7_STM32H7_Nucleo_H743ZI_CubeIDE/Prog/App/boot.h	
Demo program bootloader interface header file	1053
Demo/ARMCM7_STM32H7_Nucleo_H743ZI_GCC/Prog/boot.h	
Demo program bootloader interface header file	1055
Demo/ARMCM7_STM32H7_Nucleo_H743ZI_IAR/Prog/boot.h	
Demo program bootloader interface header file	1056
Demo/ARMCM7_STM32H7_Nucleo_H743ZI_Keil/Prog/boot.h	
Demo program bootloader interface header file	1058
Demo/HCS12_Evbplus_Dragon12p_CodeWarrior/Prog/boot.h	
Demo program bootloader interface header file	1060
Source/boot.h	
Bootloader core module header file	1061

_template/can.c	
Bootloader CAN communication interface source file	1063
ARMCM0_S32K11/can.c	
Bootloader CAN communication interface source file	1066
ARMCM0_STM32F0/can.c	
Bootloader CAN communication interface source file	1071
ARMCM0_XMC1/can.c	
Bootloader CAN communication interface source file	1075
ARMCM33_STM32L5/can.c	
Bootloader CAN communication interface source file	1077
ARMCM3_LM3S/can.c	
Bootloader CAN communication interface source file	1081
ARMCM3_STM32F1/can.c	
Bootloader CAN communication interface source file	1083
ARMCM3_STM32F2/can.c	
Bootloader CAN communication interface source file	1086
ARMCM4_S32K14/can.c	
Bootloader CAN communication interface source file	1090
ARMCM4_STM32F3/can.c	
Bootloader CAN communication interface source file	1095
ARMCM4_STM32F4/can.c	
Bootloader CAN communication interface source file	1099
ARMCM4_STM32L4/can.c	
Bootloader CAN communication interface source file	1102
ARMCM4_XMC4/can.c	
Bootloader CAN communication interface source file	1105
ARMCM7_STM32F7/can.c	
Bootloader CAN communication interface source file	1108
ARMCM7_STM32H7/can.c	
Bootloader CAN communication interface source file	1111
HCS12/can.c	
Bootloader CAN communication interface source file	1115
can.h	
Bootloader CAN communication interface header file	1119
com.c	
Bootloader communication interface source file	1121
com.h	
Bootloader communication interface header file	1124
cop.c	
Bootloader watchdog module source file	1128
cop.h	
Bootloader watchdog module header file	1130
_template/cpu.c	
Bootloader cpu module source file	1131
ARMCM0_S32K11/cpu.c	
Bootloader cpu module source file	1134
ARMCM0_STM32F0/cpu.c	
Bootloader cpu module source file	1137
ARMCM0_STM32G0/cpu.c	
Bootloader cpu module source file	1140
ARMCM0_XMC1/cpu.c	
Bootloader cpu module source file	1143
ARMCM33_STM32L5/cpu.c	
Bootloader cpu module source file	1146
ARMCM3_EFM32/cpu.c	
Bootloader cpu module source file	1149
ARMCM3_LM3S/cpu.c	
Bootloader cpu module source file	1152

ARMCM3_STM32F1/cpu.c	
Bootloader cpu module source file	1155
ARMCM3_STM32F2/cpu.c	
Bootloader cpu module source file	1158
ARMCM4_S32K14/cpu.c	
Bootloader cpu module source file	1161
ARMCM4_STM32F3/cpu.c	
Bootloader cpu module source file	1164
ARMCM4_STM32F4/cpu.c	
Bootloader cpu module source file	1167
ARMCM4_STM32L4/cpu.c	
Bootloader cpu module source file	1170
ARMCM4_TM4C/cpu.c	
Bootloader cpu module source file	1173
ARMCM4_XMC4/cpu.c	
Bootloader cpu module source file	1176
ARMCM7_STM32F7/cpu.c	
Bootloader cpu module source file	1179
ARMCM7_STM32H7/cpu.c	
Bootloader cpu module source file	1182
HCS12/cpu.c	
Bootloader cpu module source file	1185
cpu.h	
Bootloader cpu module header file	1188
_template/GCC/cpu_comp.c	
Bootloader cpu module source file	1190
ARMCM0_S32K11/GCC/cpu_comp.c	
Bootloader cpu module source file	1192
ARMCM0_S32K11/IAR/cpu_comp.c	
Bootloader cpu module source file	1193
ARMCM0_STM32F0/GCC/cpu_comp.c	
Bootloader cpu module source file	1194
ARMCM0_STM32F0/IAR/cpu_comp.c	
Bootloader cpu module source file	1196
ARMCM0_STM32F0/Keil/cpu_comp.c	
Bootloader cpu module source file	1197
ARMCM0_STM32G0/GCC/cpu_comp.c	
Bootloader cpu module source file	1198
ARMCM0_STM32G0/IAR/cpu_comp.c	
Bootloader cpu module source file	1199
ARMCM0_STM32G0/Keil/cpu_comp.c	
Bootloader cpu module source file	1201
ARMCM0_XMC1/GCC/cpu_comp.c	
Bootloader cpu module source file	1202
ARMCM0_XMC1/IAR/cpu_comp.c	
Bootloader cpu module source file	1203
ARMCM33_STM32L5/GCC/cpu_comp.c	
Bootloader cpu module source file	1204
ARMCM33_STM32L5/IAR/cpu_comp.c	
Bootloader cpu module source file	1206
ARMCM33_STM32L5/Keil/cpu_comp.c	
Bootloader cpu module source file	1207
ARMCM3_EFM32/GCC/cpu_comp.c	
Bootloader cpu module source file	1208
ARMCM3_EFM32/IAR/cpu_comp.c	
Bootloader cpu module source file	1209
ARMCM3_LM3S/GCC/cpu_comp.c	
Bootloader cpu module source file	1211

ARMCM3_LM3S/IAR/cpu_comp.c	
Bootloader cpu module source file	1212
ARMCM3_STM32F1/GCC/cpu_comp.c	
Bootloader cpu module source file	1213
ARMCM3_STM32F1/IAR/cpu_comp.c	
Bootloader cpu module source file	1214
ARMCM3_STM32F1/Keil/cpu_comp.c	
Bootloader cpu module source file	1216
ARMCM3_STM32F2/GCC/cpu_comp.c	
Bootloader cpu module source file	1217
ARMCM3_STM32F2/IAR/cpu_comp.c	
Bootloader cpu module source file	1218
ARMCM3_STM32F2/Keil/cpu_comp.c	
Bootloader cpu module source file	1219
ARMCM4_S32K14/GCC/cpu_comp.c	
Bootloader cpu module source file	1221
ARMCM4_S32K14/IAR/cpu_comp.c	
Bootloader cpu module source file	1222
ARMCM4_STM32F3/GCC/cpu_comp.c	
Bootloader cpu module source file	1223
ARMCM4_STM32F3/IAR/cpu_comp.c	
Bootloader cpu module source file	1224
ARMCM4_STM32F3/Keil/cpu_comp.c	
Bootloader cpu module source file	1226
ARMCM4_STM32F4/GCC/cpu_comp.c	
Bootloader cpu module source file	1227
ARMCM4_STM32F4/IAR/cpu_comp.c	
Bootloader cpu module source file	1228
ARMCM4_STM32F4/Keil/cpu_comp.c	
Bootloader cpu module source file	1229
ARMCM4_STM32L4/GCC/cpu_comp.c	
Bootloader cpu module source file	1231
ARMCM4_STM32L4/IAR/cpu_comp.c	
Bootloader cpu module source file	1232
ARMCM4_STM32L4/Keil/cpu_comp.c	
Bootloader cpu module source file	1233
ARMCM4_TM4C/IAR/cpu_comp.c	
Bootloader cpu module source file	1234
ARMCM4_XMC4/GCC/cpu_comp.c	
Bootloader cpu module source file	1236
ARMCM4_XMC4/IAR/cpu_comp.c	
Bootloader cpu module source file	1237
ARMCM7_STM32F7/GCC/cpu_comp.c	
Bootloader cpu module source file	1238
ARMCM7_STM32F7/IAR/cpu_comp.c	
Bootloader cpu module source file	1239
ARMCM7_STM32F7/Keil/cpu_comp.c	
Bootloader cpu module source file	1241
ARMCM7_STM32H7/GCC/cpu_comp.c	
Bootloader cpu module source file	1242
ARMCM7_STM32H7/IAR/cpu_comp.c	
Bootloader cpu module source file	1243
ARMCM7_STM32H7/Keil/cpu_comp.c	
Bootloader cpu module source file	1244
HCS12/CodeWarrior/cpu_comp.c	
Bootloader cpu module source file	1246
ARMCM3_EFM32_Olimex_EM32G880F128STK_GCC/Boot/cstart.c	
Bootloader C startup source file	1247

ARMCM3_EFM32_Olimex_EM32G880F128STK_GCC/Prog/cstart.c	
Demo program C startup source file	1248
ARMCM3_LM3S_EK_LM3S6965_GCC/Boot/cstart.c	
Bootloader C startup source file	1249
ARMCM3_LM3S_EK_LM3S6965_GCC/Prog/cstart.c	
Demo program C startup source file	1251
ARMCM3_LM3S_EK_LM3S8962_GCC/Boot/cstart.c	
Bootloader C startup source file	1252
ARMCM3_LM3S_EK_LM3S8962_GCC/Prog/cstart.c	
Demo program C startup source file	1253
file.c	
Bootloader file system interface source file	1254
file.h	
Bootloader file system interface header file	1263
_template/flash.c	
Bootloader flash driver source file	1268
ARMCM0_S32K11/flash.c	
Bootloader flash driver source file	1276
ARMCM0_STM32F0/flash.c	
Bootloader flash driver source file	1285
ARMCM0_STM32G0/flash.c	
Bootloader flash driver source file	1293
ARMCM0_XMC1/flash.c	
Bootloader flash driver source file	1302
ARMCM33_STM32L5/flash.c	
Bootloader flash driver source file	1311
ARMCM3_EFM32/flash.c	
Bootloader flash driver source file	1320
ARMCM3_LM3S/flash.c	
Bootloader flash driver source file	1329
ARMCM3_STM32F1/flash.c	
Bootloader flash driver source file	1338
ARMCM3_STM32F2/flash.c	
Bootloader flash driver source file	1345
ARMCM4_S32K14/flash.c	
Bootloader flash driver source file	1354
ARMCM4_STM32F3/flash.c	
Bootloader flash driver source file	1362
ARMCM4_STM32F4/flash.c	
Bootloader flash driver source file	1369
ARMCM4_STM32L4/flash.c	
Bootloader flash driver source file	1377
ARMCM4_TM4C/flash.c	
Bootloader flash driver source file	1386
ARMCM4_XMC4/flash.c	
Bootloader flash driver source file	1395
ARMCM7_STM32F7/flash.c	
Bootloader flash driver source file	1405
ARMCM7_STM32H7/flash.c	
Bootloader flash driver source file	1414
HCS12/flash.c	
Bootloader flash driver source file	1422
_template/flash.h	
Bootloader flash driver header file	1432
ARMCM0_S32K11/flash.h	
Bootloader flash driver header file	1436
ARMCM0_STM32F0/flash.h	
Bootloader flash driver header file	1440

ARMCM0_STM32G0/flash.h	
Bootloader flash driver header file	1443
ARMCM0_XMC1/flash.h	
Bootloader flash driver header file	1447
ARMCM33_STM32L5/flash.h	
Bootloader flash driver header file	1451
ARMCM3_EFM32/flash.h	
Bootloader flash driver header file	1454
ARMCM3_LM3S/flash.h	
Bootloader flash driver header file	1458
ARMCM3_STM32F1/flash.h	
Bootloader flash driver header file	1462
ARMCM3_STM32F2/flash.h	
Bootloader flash driver header file	1465
ARMCM4_S32K14/flash.h	
Bootloader flash driver header file	1469
ARMCM4_STM32F3/flash.h	
Bootloader flash driver header file	1473
ARMCM4_STM32F4/flash.h	
Bootloader flash driver header file	1476
ARMCM4_STM32L4/flash.h	
Bootloader flash driver header file	1480
ARMCM4_TM4C/flash.h	
Bootloader flash driver header file	1484
ARMCM4_XMC4/flash.h	
Bootloader flash driver header file	1487
ARMCM7_STM32F7/flash.h	
Bootloader flash driver header file	1491
ARMCM7_STM32H7/flash.h	
Bootloader flash driver header file	1495
HCS12/flash.h	
Bootloader flash driver header file	1498
flash_ecc.c	
Bootloader flash driver source file for HCS12 derivatives with error correction code in flash mem- ory, such as the HCS12Pxx. This flash memory uses a different addressing scheme than other HCS12 derivatives	1502
ARMCM0_STM32F0_Discovery_STM32F051_CubeIDE/Boot/App/flash_layout.c	
Custom flash layout table source file	1513
ARMCM0_STM32F0_Discovery_STM32F051_GCC/Boot/flash_layout.c	
Custom flash layout table source file	1514
ARMCM0_STM32F0_Discovery_STM32F051_IAR/Boot/flash_layout.c	
Custom flash layout table source file	1515
ARMCM0_STM32F0_Discovery_STM32F051_Keil/Boot/flash_layout.c	
Custom flash layout table source file	1516
ARMCM3_LM3S_EK_LM3S8962_GCC/Boot/flash_layout.c	
Custom flash layout table source file	1517
ARMCM3_LM3S_EK_LM3S8962_IAR/Boot/flash_layout.c	
Custom flash layout table source file	1518
ARMCM3_STM32F1_Nucleo_F103RB_CubeIDE/Boot/App/flash_layout.c	
Custom flash layout table source file	1519
ARMCM3_STM32F1_Nucleo_F103RB_GCC/Boot/flash_layout.c	
Custom flash layout table source file	1520
ARMCM3_STM32F1_Nucleo_F103RB_IAR/Boot/flash_layout.c	
Custom flash layout table source file	1521
ARMCM3_STM32F1_Nucleo_F103RB_Keil/Boot/flash_layout.c	
Custom flash layout table source file	1522
ARMCM3_STM32F1_Olimex_STM32H103_CubeIDE/Boot/App/flash_layout.c	
Custom flash layout table source file	1523

ARMCM3_STM32F1_Olimex_STM32H103_GCC/Boot/flash_layout.c	
Custom flash layout table source file	1524
ARMCM3_STM32F1_Olimex_STM32H103_IAR/Boot/flash_layout.c	
Custom flash layout table source file	1525
ARMCM3_STM32F1_Olimex_STM32H103_Keil/Boot/flash_layout.c	
Custom flash layout table source file	1526
ARMCM3_STM32F1_Olimexino_STM32_CubeIDE/Boot/App/flash_layout.c	
Custom flash layout table source file	1527
ARMCM3_STM32F1_Olimexino_STM32_GCC/Boot/flash_layout.c	
Custom flash layout table source file	1528
ARMCM3_STM32F1_Olimexino_STM32_IAR/Boot/flash_layout.c	
Custom flash layout table source file	1529
ARMCM3_STM32F1_Olimexino_STM32_Keil/Boot/flash_layout.c	
Custom flash layout table source file	1530
ARMCM4_STM32F3_Discovery_F303VC_CubeIDE/Boot/App/flash_layout.c	
Custom flash layout table source file	1531
ARMCM4_STM32F3_Discovery_F303VC_GCC/Boot/flash_layout.c	
Custom flash layout table source file	1532
ARMCM4_STM32F3_Discovery_F303VC_IAR/Boot/flash_layout.c	
Custom flash layout table source file	1533
ARMCM4_STM32F3_Discovery_F303VC_Keil/Boot/flash_layout.c	
Custom flash layout table source file	1534
_template/Prog/header.h	
Generic header file	1535
ARMCM0_STM32F0_Discovery_STM32F051_CubeIDE/Prog/App/header.h	
Generic header file	1536
ARMCM0_STM32F0_Discovery_STM32F051_GCC/Prog/header.h	
Generic header file	1537
ARMCM0_STM32F0_Discovery_STM32F051_IAR/Prog/header.h	
Generic header file	1538
ARMCM0_STM32F0_Discovery_STM32F051_Keil/Prog/header.h	
Generic header file	1539
ARMCM0_STM32F0_Nucleo_F091RC_CubeIDE/Prog/App/header.h	
Generic header file	1540
ARMCM0_STM32F0_Nucleo_F091RC_GCC/Prog/header.h	
Generic header file	1541
ARMCM0_STM32F0_Nucleo_F091RC_IAR/Prog/header.h	
Generic header file	1542
ARMCM0_STM32F0_Nucleo_F091RC_Keil/Prog/header.h	
Generic header file	1543
ARMCM0_STM32G0_Nucleo_G071RB_CubeIDE/Prog/App/header.h	
Generic header file	1544
ARMCM0_STM32G0_Nucleo_G071RB_GCC/Prog/header.h	
Generic header file	1545
ARMCM0_STM32G0_Nucleo_G071RB_IAR/Prog/header.h	
Generic header file	1546
ARMCM0_STM32G0_Nucleo_G071RB_Keil/Prog/header.h	
Generic header file	1547
ARMCM0_XMC1_XMC1400_Boot_Kit_GCC/Prog/header.h	
Generic header file	1548
ARMCM0_XMC1_XMC1400_Boot_Kit_IAR/Prog/header.h	
Generic header file	1549
ARMCM33_STM32L5_Nucleo_L552ZE_CubeIDE/Prog/App/header.h	
Generic header file	1550
ARMCM33_STM32L5_Nucleo_L552ZE_GCC/Prog/header.h	
Generic header file	1551
ARMCM33_STM32L5_Nucleo_L552ZE_IAR/Prog/header.h	
Generic header file	1552

ARMCM33_STM32L5_Nucleo_L552ZE_Keil/Prog/header.h	
Generic header file	1553
ARMCM3_EFM32_Olimex_EM32G880F128STK_GCC/Prog/header.h	
Generic header file	1554
ARMCM3_EFM32_Olimex_EM32G880F128STK_IAR/Prog/header.h	
Generic header file	1555
ARMCM3_LM3S_EK_LM3S6965_GCC/Prog/header.h	
Generic header file	1556
ARMCM3_LM3S_EK_LM3S6965_IAR/Prog/header.h	
Generic header file	1557
ARMCM3_LM3S_EK_LM3S8962_GCC/Prog/header.h	
Generic header file	1558
ARMCM3_LM3S_EK_LM3S8962_IAR/Prog/header.h	
Generic header file	1559
ARMCM3_STM32F1_Nucleo_F103RB_CubeIDE/Prog/App/header.h	
Generic header file	1560
ARMCM3_STM32F1_Nucleo_F103RB_GCC/Prog/header.h	
Generic header file	1561
ARMCM3_STM32F1_Nucleo_F103RB_IAR/Prog/header.h	
Generic header file	1562
ARMCM3_STM32F1_Nucleo_F103RB_Keil/Prog/header.h	
Generic header file	1563
ARMCM3_STM32F1_Olimex_STM32H103_CubeIDE/Prog/App/header.h	
Generic header file	1564
ARMCM3_STM32F1_Olimex_STM32H103_GCC/Prog/header.h	
Generic header file	1565
ARMCM3_STM32F1_Olimex_STM32H103_IAR/Prog/header.h	
Generic header file	1566
ARMCM3_STM32F1_Olimex_STM32H103_Keil/Prog/header.h	
Generic header file	1567
ARMCM3_STM32F1_Olimex_STM32P103_CubeIDE/Prog/App/header.h	
Generic header file	1568
ARMCM3_STM32F1_Olimex_STM32P103_GCC/Prog/header.h	
Generic header file	1569
ARMCM3_STM32F1_Olimex_STM32P103_IAR/Prog/header.h	
Generic header file	1570
ARMCM3_STM32F1_Olimex_STM32P103_Keil/Prog/header.h	
Generic header file	1571
ARMCM3_STM32F1_Olimexino_STM32_CubeIDE/Prog/App/header.h	
Generic header file	1572
ARMCM3_STM32F1_Olimexino_STM32_GCC/Prog/header.h	
Generic header file	1573
ARMCM3_STM32F1_Olimexino_STM32_IAR/Prog/header.h	
Generic header file	1574
ARMCM3_STM32F1_Olimexino_STM32_Keil/Prog/header.h	
Generic header file	1575
ARMCM3_STM32F2_Olimex_STM32P207_CubeIDE/Prog/App/header.h	
Generic header file	1576
ARMCM3_STM32F2_Olimex_STM32P207_GCC/Prog/header.h	
Generic header file	1577
ARMCM3_STM32F2_Olimex_STM32P207_IAR/Prog/header.h	
Generic header file	1578
ARMCM3_STM32F2_Olimex_STM32P207_Keil/Prog/header.h	
Generic header file	1579
ARMCM4_S32K14_S32K144EVB_GCC/Prog/header.h	
Generic header file	1580
ARMCM4_S32K14_S32K144EVB_IAR/Prog/header.h	
Generic header file	1581

ARMCM4_STM32F3_Discovery_F303VC_CubeIDE/Prog/App/header.h	
Generic header file	1582
ARMCM4_STM32F3_Discovery_F303VC_GCC/Prog/header.h	
Generic header file	1583
ARMCM4_STM32F3_Discovery_F303VC_IAR/Prog/header.h	
Generic header file	1584
ARMCM4_STM32F3_Discovery_F303VC_Keil/Prog/header.h	
Generic header file	1585
ARMCM4_STM32F3_Nucleo_F303K8_CubeIDE/Prog/App/header.h	
Generic header file	1586
ARMCM4_STM32F3_Nucleo_F303K8_GCC/Prog/header.h	
Generic header file	1587
ARMCM4_STM32F3_Nucleo_F303K8_IAR/Prog/header.h	
Generic header file	1588
ARMCM4_STM32F3_Nucleo_F303K8_Keil/Prog/header.h	
Generic header file	1589
ARMCM4_STM32F4_Nucleo_F429ZI_CubeIDE/Prog/App/header.h	
Generic header file	1590
ARMCM4_STM32F4_Nucleo_F429ZI_GCC/Prog/header.h	
Generic header file	1591
ARMCM4_STM32F4_Nucleo_F429ZI_IAR/Prog/header.h	
Generic header file	1592
ARMCM4_STM32F4_Nucleo_F429ZI_Keil/Prog/header.h	
Generic header file	1593
ARMCM4_STM32F4_Olimex_STM32P405_CubeIDE/Prog/App/header.h	
Generic header file	1594
ARMCM4_STM32F4_Olimex_STM32P405_GCC/Prog/header.h	
Generic header file	1595
ARMCM4_STM32F4_Olimex_STM32P405_IAR/Prog/header.h	
Generic header file	1596
ARMCM4_STM32F4_Olimex_STM32P405_Keil/Prog/header.h	
Generic header file	1597
ARMCM4_STM32L4_Nucleo_L476RG_CubeIDE/Prog/App/header.h	
Generic header file	1598
ARMCM4_STM32L4_Nucleo_L476RG_GCC/Prog/header.h	
Generic header file	1599
ARMCM4_STM32L4_Nucleo_L476RG_IAR/Prog/header.h	
Generic header file	1600
ARMCM4_STM32L4_Nucleo_L476RG_Keil/Prog/header.h	
Generic header file	1601
ARMCM4_TM4C_DK_TM4C123G_IAR/Prog/header.h	
Generic header file	1602
ARMCM4_XMC4_XMC4700_Relax_Kit_GCC/Prog/header.h	
Generic header file	1603
ARMCM4_XMC4_XMC4700_Relax_Kit_IAR/Prog/header.h	
Generic header file	1604
ARMCM7_STM32F7_Nucleo_F746ZG_CubeIDE/Prog/App/header.h	
Generic header file	1605
ARMCM7_STM32F7_Nucleo_F746ZG_GCC/Prog/header.h	
Generic header file	1606
ARMCM7_STM32F7_Nucleo_F746ZG_IAR/Prog/header.h	
Generic header file	1607
ARMCM7_STM32F7_Nucleo_F746ZG_Keil/Prog/header.h	
Generic header file	1608
ARMCM7_STM32F7_Nucleo_F767ZI_CubeIDE/Prog/App/header.h	
Generic header file	1609
ARMCM7_STM32F7_Nucleo_F767ZI_GCC/Prog/header.h	
Generic header file	1610

ARMCM7_STM32F7_Nucleo_F767ZI_IAR/Prog/header.h	
Generic header file	1611
ARMCM7_STM32F7_Nucleo_F767ZI_Keil/Prog/header.h	
Generic header file	1612
ARMCM7_STM32H7_Nucleo_H743ZI_CubeIDE/Prog/App/header.h	
Generic header file	1613
ARMCM7_STM32H7_Nucleo_H743ZI_GCC/Prog/header.h	
Generic header file	1614
ARMCM7_STM32H7_Nucleo_H743ZI_IAR/Prog/header.h	
Generic header file	1615
ARMCM7_STM32H7_Nucleo_H743ZI_Keil/Prog/header.h	
Generic header file	1616
HCS12_DevKit_S12G128_CodeWarrior/Prog/header.h	
Generic header file	1617
HCS12_Evbplus_Dragon12p_CodeWarrior/Prog/header.h	
Generic header file	1619
_template/Boot/hooks.c	
Bootloader callback source file	1620
ARMCM0_STM32F0_Discovery_STM32F051_CubeIDE/Boot/App/hooks.c	
Bootloader callback source file	1625
ARMCM0_STM32F0_Discovery_STM32F051_GCC/Boot/hooks.c	
Bootloader callback source file	1630
ARMCM0_STM32F0_Discovery_STM32F051_IAR/Boot/hooks.c	
Bootloader callback source file	1635
ARMCM0_STM32F0_Discovery_STM32F051_Keil/Boot/hooks.c	
Bootloader callback source file	1640
ARMCM0_STM32F0_Nucleo_F091RC_CubeIDE/Boot/App/hooks.c	
Bootloader callback source file	1645
ARMCM0_STM32F0_Nucleo_F091RC_GCC/Boot/hooks.c	
Bootloader callback source file	1650
ARMCM0_STM32F0_Nucleo_F091RC_IAR/Boot/hooks.c	
Bootloader callback source file	1655
ARMCM0_STM32F0_Nucleo_F091RC_Keil/Boot/hooks.c	
Bootloader callback source file	1660
ARMCM0_STM32G0_Nucleo_G071RB_CubeIDE/Boot/App/hooks.c	
Bootloader callback source file	1665
ARMCM0_STM32G0_Nucleo_G071RB_GCC/Boot/hooks.c	
Bootloader callback source file	1670
ARMCM0_STM32G0_Nucleo_G071RB_IAR/Boot/hooks.c	
Bootloader callback source file	1675
ARMCM0_STM32G0_Nucleo_G071RB_Keil/Boot/hooks.c	
Bootloader callback source file	1680
ARMCM0_XMC1_XMC1400_Boot_Kit_GCC/Boot/hooks.c	
Bootloader callback source file	1685
ARMCM0_XMC1_XMC1400_Boot_Kit_IAR/Boot/hooks.c	
Bootloader callback source file	1690
ARMCM33_STM32L5_Nucleo_L552ZE_CubeIDE/Boot/App/hooks.c	
Bootloader callback source file	1695
ARMCM33_STM32L5_Nucleo_L552ZE_GCC/Boot/hooks.c	
Bootloader callback source file	1701
ARMCM33_STM32L5_Nucleo_L552ZE_IAR/Boot/hooks.c	
Bootloader callback source file	1706
ARMCM33_STM32L5_Nucleo_L552ZE_Keil/Boot/hooks.c	
Bootloader callback source file	1712
ARMCM3_EFM32_Olimex_EM32G880F128STK_GCC/Boot/hooks.c	
Bootloader callback source file	1718
ARMCM3_EFM32_Olimex_EM32G880F128STK_IAR/Boot/hooks.c	
Bootloader callback source file	1724

ARMCM3_LM3S_EK_LM3S6965_GCC/Boot/hooks.c	
Bootloader callback source file	1729
ARMCM3_LM3S_EK_LM3S6965_IAR/Boot/hooks.c	
Bootloader callback source file	1737
ARMCM3_LM3S_EK_LM3S8962_GCC/Boot/hooks.c	
Bootloader callback source file	1744
ARMCM3_LM3S_EK_LM3S8962_IAR/Boot/hooks.c	
Bootloader callback source file	1749
ARMCM3_STM32F1_Nucleo_F103RB_CubeIDE/Boot/App/hooks.c	
Bootloader callback source file	1755
ARMCM3_STM32F1_Nucleo_F103RB_GCC/Boot/hooks.c	
Bootloader callback source file	1760
ARMCM3_STM32F1_Nucleo_F103RB_IAR/Boot/hooks.c	
Bootloader callback source file	1765
ARMCM3_STM32F1_Nucleo_F103RB_Keil/Boot/hooks.c	
Bootloader callback source file	1770
ARMCM3_STM32F1_Olimex_STM32H103_CubeIDE/Boot/App/hooks.c	
Bootloader callback source file	1775
ARMCM3_STM32F1_Olimex_STM32H103_GCC/Boot/hooks.c	
Bootloader callback source file	1780
ARMCM3_STM32F1_Olimex_STM32H103_IAR/Boot/hooks.c	
Bootloader callback source file	1786
ARMCM3_STM32F1_Olimex_STM32H103_Keil/Boot/hooks.c	
Bootloader callback source file	1792
ARMCM3_STM32F1_Olimex_STM32P103_CubeIDE/Boot/App/hooks.c	
Bootloader callback source file	1798
ARMCM3_STM32F1_Olimex_STM32P103_GCC/Boot/hooks.c	
Bootloader callback source file	1806
ARMCM3_STM32F1_Olimex_STM32P103_IAR/Boot/hooks.c	
Bootloader callback source file	1814
ARMCM3_STM32F1_Olimex_STM32P103_Keil/Boot/hooks.c	
Bootloader callback source file	1822
ARMCM3_STM32F1_Olimexino_STM32_CubeIDE/Boot/App/hooks.c	
Bootloader callback source file	1829
ARMCM3_STM32F1_Olimexino_STM32_GCC/Boot/hooks.c	
Bootloader callback source file	1838
ARMCM3_STM32F1_Olimexino_STM32_IAR/Boot/hooks.c	
Bootloader callback source file	1847
ARMCM3_STM32F1_Olimexino_STM32_Keil/Boot/hooks.c	
Bootloader callback source file	1856
ARMCM3_STM32F2_Olimex_STM32P207_CubeIDE/Boot/App/hooks.c	
Bootloader callback source file	1865
ARMCM3_STM32F2_Olimex_STM32P207_GCC/Boot/hooks.c	
Bootloader callback source file	1872
ARMCM3_STM32F2_Olimex_STM32P207_IAR/Boot/hooks.c	
Bootloader callback source file	1880
ARMCM3_STM32F2_Olimex_STM32P207_Keil/Boot/hooks.c	
Bootloader callback source file	1888
ARMCM4_S32K14_S32K144EVB_GCC/Boot/hooks.c	
Bootloader callback source file	1895
ARMCM4_S32K14_S32K144EVB_IAR/Boot/hooks.c	
Bootloader callback source file	1900
ARMCM4_STM32F3_Discovery_F303VC_CubeIDE/Boot/App/hooks.c	
Bootloader callback source file	1906
ARMCM4_STM32F3_Discovery_F303VC_GCC/Boot/hooks.c	
Bootloader callback source file	1911
ARMCM4_STM32F3_Discovery_F303VC_IAR/Boot/hooks.c	
Bootloader callback source file	1917

ARMCM4_STM32F3_Discovery_F303VC_Keil/Boot/hooks.c	
Bootloader callback source file	1923
ARMCM4_STM32F3_Nucleo_F303K8_CubeIDE/Boot/App/hooks.c	
Bootloader callback source file	1929
ARMCM4_STM32F3_Nucleo_F303K8_GCC/Boot/hooks.c	
Bootloader callback source file	1935
ARMCM4_STM32F3_Nucleo_F303K8_IAR/Boot/hooks.c	
Bootloader callback source file	1940
ARMCM4_STM32F3_Nucleo_F303K8_Keil/Boot/hooks.c	
Bootloader callback source file	1945
ARMCM4_STM32F4_Nucleo_F429ZI_CubeIDE/Boot/App/hooks.c	
Bootloader callback source file	1950
ARMCM4_STM32F4_Nucleo_F429ZI_GCC/Boot/hooks.c	
Bootloader callback source file	1955
ARMCM4_STM32F4_Nucleo_F429ZI_IAR/Boot/hooks.c	
Bootloader callback source file	1961
ARMCM4_STM32F4_Nucleo_F429ZI_Keil/Boot/hooks.c	
Bootloader callback source file	1967
ARMCM4_STM32F4_Olimex_STM32P405_CubeIDE/Boot/App/hooks.c	
Bootloader callback source file	1973
ARMCM4_STM32F4_Olimex_STM32P405_GCC/Boot/hooks.c	
Bootloader callback source file	1982
ARMCM4_STM32F4_Olimex_STM32P405_IAR/Boot/hooks.c	
Bootloader callback source file	1991
ARMCM4_STM32F4_Olimex_STM32P405_Keil/Boot/hooks.c	
Bootloader callback source file	2000
ARMCM4_STM32L4_Nucleo_L476RG_CubeIDE/Boot/App/hooks.c	
Bootloader callback source file	2009
ARMCM4_STM32L4_Nucleo_L476RG_GCC/Boot/hooks.c	
Bootloader callback source file	2014
ARMCM4_STM32L4_Nucleo_L476RG_IAR/Boot/hooks.c	
Bootloader callback source file	2019
ARMCM4_STM32L4_Nucleo_L476RG_Keil/Boot/hooks.c	
Bootloader callback source file	2024
ARMCM4_TM4C_DK_TM4C123G_IAR/Boot/hooks.c	
Bootloader callback source file	2029
ARMCM4_XMC4_XMC4700_Relax_Kit_GCC/Boot/hooks.c	
Bootloader callback source file	2038
ARMCM4_XMC4_XMC4700_Relax_Kit_IAR/Boot/hooks.c	
Bootloader callback source file	2045
ARMCM7_STM32F7_Nucleo_F746ZG_CubeIDE/Boot/App/hooks.c	
Bootloader callback source file	2053
ARMCM7_STM32F7_Nucleo_F746ZG_GCC/Boot/hooks.c	
Bootloader callback source file	2059
ARMCM7_STM32F7_Nucleo_F746ZG_IAR/Boot/hooks.c	
Bootloader callback source file	2065
ARMCM7_STM32F7_Nucleo_F746ZG_Keil/Boot/hooks.c	
Bootloader callback source file	2071
ARMCM7_STM32F7_Nucleo_F767ZI_CubeIDE/Boot/App/hooks.c	
Bootloader callback source file	2077
ARMCM7_STM32F7_Nucleo_F767ZI_GCC/Boot/hooks.c	
Bootloader callback source file	2083
ARMCM7_STM32F7_Nucleo_F767ZI_IAR/Boot/hooks.c	
Bootloader callback source file	2088
ARMCM7_STM32F7_Nucleo_F767ZI_Keil/Boot/hooks.c	
Bootloader callback source file	2093
ARMCM7_STM32H7_Nucleo_H743ZI_CubeIDE/Boot/App/hooks.c	
Bootloader callback source file	2098

ARMCM7_STM32H7_Nucleo_H743ZI_GCC/Boot/hooks.c	
Bootloader callback source file	2103
ARMCM7_STM32H7_Nucleo_H743ZI_IAR/Boot/hooks.c	
Bootloader callback source file	2109
ARMCM7_STM32H7_Nucleo_H743ZI_Keil/Boot/hooks.c	
Bootloader callback source file	2115
HCS12_DevKit_S12G128_CodeWarrior/Boot/hooks.c	
Bootloader callback source file	2121
HCS12_Evbplus_Dragon12p_CodeWarrior/Boot/hooks.c	
Bootloader callback source file	2127
irq.c	
IRQ driver source file	2132
irq.h	
IRQ driver header file	2134
_template/Boot/led.c	
LED driver source file	2136
_template/Prog/led.c	
LED driver source file	2138
ARMCM0_STM32F0_Discovery_STM32F051_CubeIDE/Boot/App/led.c	
LED driver source file	2139
ARMCM0_STM32F0_Discovery_STM32F051_CubeIDE/Prog/App/led.c	
LED driver source file	2141
ARMCM0_STM32F0_Discovery_STM32F051_GCC/Boot/led.c	
LED driver source file	2142
ARMCM0_STM32F0_Discovery_STM32F051_GCC/Prog/led.c	
LED driver source file	2144
ARMCM0_STM32F0_Discovery_STM32F051_IAR/Boot/led.c	
LED driver source file	2145
ARMCM0_STM32F0_Discovery_STM32F051_IAR/Prog/led.c	
LED driver source file	2147
ARMCM0_STM32F0_Discovery_STM32F051_Keil/Boot/led.c	
LED driver source file	2148
ARMCM0_STM32F0_Discovery_STM32F051_Keil/Prog/led.c	
LED driver source file	2150
ARMCM0_STM32F0_Nucleo_F091RC_CubeIDE/Boot/App/led.c	
LED driver source file	2151
ARMCM0_STM32F0_Nucleo_F091RC_CubeIDE/Prog/App/led.c	
LED driver source file	2153
ARMCM0_STM32F0_Nucleo_F091RC_GCC/Boot/led.c	
LED driver source file	2154
ARMCM0_STM32F0_Nucleo_F091RC_GCC/Prog/led.c	
LED driver source file	2156
ARMCM0_STM32F0_Nucleo_F091RC_IAR/Boot/led.c	
LED driver source file	2157
ARMCM0_STM32F0_Nucleo_F091RC_IAR/Prog/led.c	
LED driver source file	2159
ARMCM0_STM32F0_Nucleo_F091RC_Keil/Boot/led.c	
LED driver source file	2160
ARMCM0_STM32F0_Nucleo_F091RC_Keil/Prog/led.c	
LED driver source file	2162
ARMCM0_STM32G0_Nucleo_G071RB_CubeIDE/Boot/App/led.c	
LED driver source file	2163
ARMCM0_STM32G0_Nucleo_G071RB_CubeIDE/Prog/App/led.c	
LED driver source file	2165
ARMCM0_STM32G0_Nucleo_G071RB_GCC/Boot/led.c	
LED driver source file	2166
ARMCM0_STM32G0_Nucleo_G071RB_GCC/Prog/led.c	
LED driver source file	2168

ARMCM0_STM32G0_Nucleo_G071RB_IAR/Boot/led.c	
LED driver source file	2169
ARMCM0_STM32G0_Nucleo_G071RB_IAR/Prog/led.c	
LED driver source file	2171
ARMCM0_STM32G0_Nucleo_G071RB_Keil/Boot/led.c	
LED driver source file	2172
ARMCM0_STM32G0_Nucleo_G071RB_Keil/Prog/led.c	
LED driver source file	2174
ARMCM0_XMC1_XMC1400_Boot_Kit_GCC/Boot/led.c	
LED driver source file	2175
ARMCM0_XMC1_XMC1400_Boot_Kit_GCC/Prog/led.c	
LED driver source file	2177
ARMCM0_XMC1_XMC1400_Boot_Kit_IAR/Boot/led.c	
LED driver source file	2178
ARMCM0_XMC1_XMC1400_Boot_Kit_IAR/Prog/led.c	
LED driver source file	2180
ARMCM33_STM32L5_Nucleo_L552ZE_CubeIDE/Boot/App/led.c	
LED driver source file	2181
ARMCM33_STM32L5_Nucleo_L552ZE_CubeIDE/Prog/App/led.c	
LED driver source file	2183
ARMCM33_STM32L5_Nucleo_L552ZE_GCC/Boot/led.c	
LED driver source file	2184
ARMCM33_STM32L5_Nucleo_L552ZE_GCC/Prog/led.c	
LED driver source file	2186
ARMCM33_STM32L5_Nucleo_L552ZE_IAR/Boot/led.c	
LED driver source file	2187
ARMCM33_STM32L5_Nucleo_L552ZE_IAR/Prog/led.c	
LED driver source file	2189
ARMCM33_STM32L5_Nucleo_L552ZE_Keil/Boot/led.c	
LED driver source file	2190
ARMCM33_STM32L5_Nucleo_L552ZE_Keil/Prog/led.c	
LED driver source file	2192
ARMCM3_EFM32_Olimex_EM32G880F128STK_GCC/Prog/led.c	
LED driver source file	2193
ARMCM3_EFM32_Olimex_EM32G880F128STK_IAR/Prog/led.c	
LED driver source file	2195
ARMCM3_LM3S_EK_LM3S6965_GCC/Prog/led.c	
LED driver source file	2196
ARMCM3_LM3S_EK_LM3S6965_IAR/Prog/led.c	
LED driver source file	2197
ARMCM3_LM3S_EK_LM3S8962_GCC/Prog/led.c	
LED driver source file	2198
ARMCM3_LM3S_EK_LM3S8962_IAR/Prog/led.c	
LED driver source file	2200
ARMCM3_STM32F1_Nucleo_F103RB_CubeIDE/Boot/App/led.c	
LED driver source file	2201
ARMCM3_STM32F1_Nucleo_F103RB_CubeIDE/Prog/App/led.c	
LED driver source file	2203
ARMCM3_STM32F1_Nucleo_F103RB_GCC/Boot/led.c	
LED driver source file	2204
ARMCM3_STM32F1_Nucleo_F103RB_GCC/Prog/led.c	
LED driver source file	2206
ARMCM3_STM32F1_Nucleo_F103RB_IAR/Boot/led.c	
LED driver source file	2207
ARMCM3_STM32F1_Nucleo_F103RB_IAR/Prog/led.c	
LED driver source file	2209
ARMCM3_STM32F1_Nucleo_F103RB_Keil/Boot/led.c	
LED driver source file	2210

ARMCM3_STM32F1_Nucleo_F103RB_Keil/Prog/led.c	
LED driver source file	2212
ARMCM3_STM32F1_Olimex_STM32H103_CubeIDE/Boot/App/led.c	
LED driver source file	2213
ARMCM3_STM32F1_Olimex_STM32H103_CubeIDE/Prog/App/led.c	
LED driver source file	2215
ARMCM3_STM32F1_Olimex_STM32H103_GCC/Boot/led.c	
LED driver source file	2216
ARMCM3_STM32F1_Olimex_STM32H103_GCC/Prog/led.c	
LED driver source file	2218
ARMCM3_STM32F1_Olimex_STM32H103_IAR/Boot/led.c	
LED driver source file	2219
ARMCM3_STM32F1_Olimex_STM32H103_IAR/Prog/led.c	
LED driver source file	2221
ARMCM3_STM32F1_Olimex_STM32H103_Keil/Boot/led.c	
LED driver source file	2222
ARMCM3_STM32F1_Olimex_STM32H103_Keil/Prog/led.c	
LED driver source file	2224
ARMCM3_STM32F1_Olimex_STM32P103_CubeIDE/Boot/App/led.c	
LED driver source file	2225
ARMCM3_STM32F1_Olimex_STM32P103_CubeIDE/Prog/App/led.c	
LED driver source file	2227
ARMCM3_STM32F1_Olimex_STM32P103_GCC/Boot/led.c	
LED driver source file	2228
ARMCM3_STM32F1_Olimex_STM32P103_GCC/Prog/led.c	
LED driver source file	2230
ARMCM3_STM32F1_Olimex_STM32P103_IAR/Boot/led.c	
LED driver source file	2231
ARMCM3_STM32F1_Olimex_STM32P103_IAR/Prog/led.c	
LED driver source file	2233
ARMCM3_STM32F1_Olimex_STM32P103_Keil/Boot/led.c	
LED driver source file	2234
ARMCM3_STM32F1_Olimex_STM32P103_Keil/Prog/led.c	
LED driver source file	2236
ARMCM3_STM32F1_Olimexino_STM32_CubeIDE/Boot/App/led.c	
LED driver source file	2237
ARMCM3_STM32F1_Olimexino_STM32_CubeIDE/Prog/App/led.c	
LED driver source file	2239
ARMCM3_STM32F1_Olimexino_STM32_GCC/Boot/led.c	
LED driver source file	2240
ARMCM3_STM32F1_Olimexino_STM32_GCC/Prog/led.c	
LED driver source file	2242
ARMCM3_STM32F1_Olimexino_STM32_IAR/Boot/led.c	
LED driver source file	2243
ARMCM3_STM32F1_Olimexino_STM32_IAR/Prog/led.c	
LED driver source file	2245
ARMCM3_STM32F1_Olimexino_STM32_Keil/Boot/led.c	
LED driver source file	2246
ARMCM3_STM32F1_Olimexino_STM32_Keil/Prog/led.c	
LED driver source file	2248
ARMCM3_STM32F2_Olimex_STM32P207_CubeIDE/Boot/App/led.c	
LED driver source file	2249
ARMCM3_STM32F2_Olimex_STM32P207_CubeIDE/Prog/App/led.c	
LED driver source file	2251
ARMCM3_STM32F2_Olimex_STM32P207_GCC/Boot/led.c	
LED driver source file	2252
ARMCM3_STM32F2_Olimex_STM32P207_GCC/Prog/led.c	
LED driver source file	2254

ARMCM3_STM32F2_Olimex_STM32P207_IAR/Boot/led.c	
LED driver source file	2255
ARMCM3_STM32F2_Olimex_STM32P207_IAR/Prog/led.c	
LED driver source file	2257
ARMCM3_STM32F2_Olimex_STM32P207_Keil/Boot/led.c	
LED driver source file	2258
ARMCM3_STM32F2_Olimex_STM32P207_Keil/Prog/led.c	
LED driver source file	2260
ARMCM4_S32K14_S32K144EVB_GCC/Boot/led.c	
LED driver source file	2261
ARMCM4_S32K14_S32K144EVB_GCC/Prog/led.c	
LED driver source file	2263
ARMCM4_S32K14_S32K144EVB_IAR/Boot/led.c	
LED driver source file	2264
ARMCM4_S32K14_S32K144EVB_IAR/Prog/led.c	
LED driver source file	2266
ARMCM4_STM32F3_Discovery_F303VC_CubeIDE/Boot/App/led.c	
LED driver source file	2267
ARMCM4_STM32F3_Discovery_F303VC_CubeIDE/Prog/App/led.c	
LED driver source file	2269
ARMCM4_STM32F3_Discovery_F303VC_GCC/Boot/led.c	
LED driver source file	2271
ARMCM4_STM32F3_Discovery_F303VC_GCC/Prog/led.c	
LED driver source file	2272
ARMCM4_STM32F3_Discovery_F303VC_IAR/Boot/led.c	
LED driver source file	2274
ARMCM4_STM32F3_Discovery_F303VC_IAR/Prog/led.c	
LED driver source file	2276
ARMCM4_STM32F3_Discovery_F303VC_Keil/Boot/led.c	
LED driver source file	2277
ARMCM4_STM32F3_Discovery_F303VC_Keil/Prog/led.c	
LED driver source file	2279
ARMCM4_STM32F3_Nucleo_F303K8_CubeIDE/Boot/App/led.c	
LED driver source file	2281
ARMCM4_STM32F3_Nucleo_F303K8_CubeIDE/Prog/App/led.c	
LED driver source file	2282
ARMCM4_STM32F3_Nucleo_F303K8_GCC/Boot/led.c	
LED driver source file	2284
ARMCM4_STM32F3_Nucleo_F303K8_GCC/Prog/led.c	
LED driver source file	2286
ARMCM4_STM32F3_Nucleo_F303K8_IAR/Boot/led.c	
LED driver source file	2287
ARMCM4_STM32F3_Nucleo_F303K8_IAR/Prog/led.c	
LED driver source file	2289
ARMCM4_STM32F3_Nucleo_F303K8_Keil/Boot/led.c	
LED driver source file	2290
ARMCM4_STM32F3_Nucleo_F303K8_Keil/Prog/led.c	
LED driver source file	2292
ARMCM4_STM32F4_Nucleo_F429ZI_CubeIDE/Boot/App/led.c	
LED driver source file	2293
ARMCM4_STM32F4_Nucleo_F429ZI_CubeIDE/Prog/App/led.c	
LED driver source file	2295
ARMCM4_STM32F4_Nucleo_F429ZI_GCC/Prog/led.c	
LED driver source file	2296
ARMCM4_STM32F4_Nucleo_F429ZI_IAR/Prog/led.c	
LED driver source file	2297
ARMCM4_STM32F4_Nucleo_F429ZI_Keil/Prog/led.c	
LED driver source file	2299

ARMCM4_STM32F4_Olimex_STM32P405_CubeIDE/Boot/App/led.c	
LED driver source file	2300
ARMCM4_STM32F4_Olimex_STM32P405_CubeIDE/Prog/App/led.c	
LED driver source file	2302
ARMCM4_STM32F4_Olimex_STM32P405_GCC/Boot/led.c	
LED driver source file	2303
ARMCM4_STM32F4_Olimex_STM32P405_GCC/Prog/led.c	
LED driver source file	2305
ARMCM4_STM32F4_Olimex_STM32P405_IAR/Boot/led.c	
LED driver source file	2306
ARMCM4_STM32F4_Olimex_STM32P405_IAR/Prog/led.c	
LED driver source file	2308
ARMCM4_STM32F4_Olimex_STM32P405_Keil/Boot/led.c	
LED driver source file	2309
ARMCM4_STM32F4_Olimex_STM32P405_Keil/Prog/led.c	
LED driver source file	2311
ARMCM4_STM32L4_Nucleo_L476RG_CubeIDE/Boot/App/led.c	
LED driver source file	2312
ARMCM4_STM32L4_Nucleo_L476RG_CubeIDE/Prog/App/led.c	
LED driver source file	2314
ARMCM4_STM32L4_Nucleo_L476RG_GCC/Boot/led.c	
LED driver source file	2315
ARMCM4_STM32L4_Nucleo_L476RG_GCC/Prog/led.c	
LED driver source file	2317
ARMCM4_STM32L4_Nucleo_L476RG_IAR/Boot/led.c	
LED driver source file	2318
ARMCM4_STM32L4_Nucleo_L476RG_IAR/Prog/led.c	
LED driver source file	2320
ARMCM4_STM32L4_Nucleo_L476RG_Keil/Boot/led.c	
LED driver source file	2321
ARMCM4_STM32L4_Nucleo_L476RG_Keil/Prog/led.c	
LED driver source file	2323
ARMCM4_TM4C_DK_TM4C123G_IAR/Prog/led.c	
LED driver source file	2324
ARMCM4_XMC4_XMC4700_Relax_Kit_GCC/Boot/led.c	
LED driver source file	2325
ARMCM4_XMC4_XMC4700_Relax_Kit_GCC/Prog/led.c	
LED driver source file	2327
ARMCM4_XMC4_XMC4700_Relax_Kit_IAR/Boot/led.c	
LED driver source file	2328
ARMCM4_XMC4_XMC4700_Relax_Kit_IAR/Prog/led.c	
LED driver source file	2330
ARMCM7_STM32F7_Nucleo_F746ZG_CubeIDE/Boot/App/led.c	
LED driver source file	2331
ARMCM7_STM32F7_Nucleo_F746ZG_CubeIDE/Prog/App/led.c	
LED driver source file	2333
ARMCM7_STM32F7_Nucleo_F746ZG_GCC/Boot/led.c	
LED driver source file	2334
ARMCM7_STM32F7_Nucleo_F746ZG_GCC/Prog/led.c	
LED driver source file	2336
ARMCM7_STM32F7_Nucleo_F746ZG_IAR/Boot/led.c	
LED driver source file	2337
ARMCM7_STM32F7_Nucleo_F746ZG_IAR/Prog/led.c	
LED driver source file	2339
ARMCM7_STM32F7_Nucleo_F746ZG_Keil/Boot/led.c	
LED driver source file	2340
ARMCM7_STM32F7_Nucleo_F746ZG_Keil/Prog/led.c	
LED driver source file	2342

ARMCM7_STM32F7_Nucleo_F767ZI_CubeIDE/Boot/App/led.c	
LED driver source file	2343
ARMCM7_STM32F7_Nucleo_F767ZI_CubeIDE/Prog/App/led.c	
LED driver source file	2345
ARMCM7_STM32F7_Nucleo_F767ZI_GCC/Boot/led.c	
LED driver source file	2346
ARMCM7_STM32F7_Nucleo_F767ZI_GCC/Prog/led.c	
LED driver source file	2348
ARMCM7_STM32F7_Nucleo_F767ZI_IAR/Boot/led.c	
LED driver source file	2349
ARMCM7_STM32F7_Nucleo_F767ZI_IAR/Prog/led.c	
LED driver source file	2351
ARMCM7_STM32F7_Nucleo_F767ZI_Keil/Boot/led.c	
LED driver source file	2352
ARMCM7_STM32F7_Nucleo_F767ZI_Keil/Prog/led.c	
LED driver source file	2354
ARMCM7_STM32H7_Nucleo_H743ZI_CubeIDE/Boot/App/led.c	
LED driver source file	2355
ARMCM7_STM32H7_Nucleo_H743ZI_CubeIDE/Prog/App/led.c	
LED driver source file	2357
ARMCM7_STM32H7_Nucleo_H743ZI_GCC/Boot/led.c	
LED driver source file	2358
ARMCM7_STM32H7_Nucleo_H743ZI_GCC/Prog/led.c	
LED driver source file	2360
ARMCM7_STM32H7_Nucleo_H743ZI_IAR/Boot/led.c	
LED driver source file	2361
ARMCM7_STM32H7_Nucleo_H743ZI_IAR/Prog/led.c	
LED driver source file	2363
ARMCM7_STM32H7_Nucleo_H743ZI_Keil/Boot/led.c	
LED driver source file	2364
ARMCM7_STM32H7_Nucleo_H743ZI_Keil/Prog/led.c	
LED driver source file	2366
HCS12_DevKit_S12G128_CodeWarrior/Boot/led.c	
LED driver source file	2367
HCS12_DevKit_S12G128_CodeWarrior/Prog/led.c	
LED driver source file	2369
HCS12_Evbplus_Dragon12p_CodeWarrior/Prog/led.c	
LED driver source file	2370
_template/Boot/led.h	
LED driver header file	2372
_template/Prog/led.h	
LED driver header file	2373
ARMCM0_STM32F0_Discovery_STM32F051_CubeIDE/Boot/App/led.h	
LED driver header file	2374
ARMCM0_STM32F0_Discovery_STM32F051_CubeIDE/Prog/App/led.h	
LED driver header file	2376
ARMCM0_STM32F0_Discovery_STM32F051_GCC/Boot/led.h	
LED driver header file	2377
ARMCM0_STM32F0_Discovery_STM32F051_GCC/Prog/led.h	
LED driver header file	2379
ARMCM0_STM32F0_Discovery_STM32F051_IAR/Boot/led.h	
LED driver header file	2380
ARMCM0_STM32F0_Discovery_STM32F051_IAR/Prog/led.h	
LED driver header file	2382
ARMCM0_STM32F0_Discovery_STM32F051_Keil/Boot/led.h	
LED driver header file	2383
ARMCM0_STM32F0_Discovery_STM32F051_Keil/Prog/led.h	
LED driver header file	2385

ARMCM0_STM32F0_Nucleo_F091RC_CubeIDE/Boot/App/led.h	
LED driver header file	2386
ARMCM0_STM32F0_Nucleo_F091RC_CubeIDE/Prog/App/led.h	
LED driver header file	2388
ARMCM0_STM32F0_Nucleo_F091RC_GCC/Boot/led.h	
LED driver header file	2389
ARMCM0_STM32F0_Nucleo_F091RC_GCC/Prog/led.h	
LED driver header file	2391
ARMCM0_STM32F0_Nucleo_F091RC_IAR/Boot/led.h	
LED driver header file	2392
ARMCM0_STM32F0_Nucleo_F091RC_IAR/Prog/led.h	
LED driver header file	2394
ARMCM0_STM32F0_Nucleo_F091RC_Keil/Boot/led.h	
LED driver header file	2395
ARMCM0_STM32F0_Nucleo_F091RC_Keil/Prog/led.h	
LED driver header file	2397
ARMCM0_STM32G0_Nucleo_G071RB_CubeIDE/Boot/App/led.h	
LED driver header file	2398
ARMCM0_STM32G0_Nucleo_G071RB_CubeIDE/Prog/App/led.h	
LED driver header file	2400
ARMCM0_STM32G0_Nucleo_G071RB_GCC/Boot/led.h	
LED driver header file	2401
ARMCM0_STM32G0_Nucleo_G071RB_GCC/Prog/led.h	
LED driver header file	2403
ARMCM0_STM32G0_Nucleo_G071RB_IAR/Boot/led.h	
LED driver header file	2404
ARMCM0_STM32G0_Nucleo_G071RB_IAR/Prog/led.h	
LED driver header file	2406
ARMCM0_STM32G0_Nucleo_G071RB_Keil/Boot/led.h	
LED driver header file	2407
ARMCM0_STM32G0_Nucleo_G071RB_Keil/Prog/led.h	
LED driver header file	2409
ARMCM0_XMC1_XMC1400_Boot_Kit_GCC/Boot/led.h	
LED driver header file	2410
ARMCM0_XMC1_XMC1400_Boot_Kit_GCC/Prog/led.h	
LED driver header file	2412
ARMCM0_XMC1_XMC1400_Boot_Kit_IAR/Boot/led.h	
LED driver header file	2413
ARMCM0_XMC1_XMC1400_Boot_Kit_IAR/Prog/led.h	
LED driver header file	2415
ARMCM33_STM32L5_Nucleo_L552ZE_CubeIDE/Boot/App/led.h	
LED driver header file	2416
ARMCM33_STM32L5_Nucleo_L552ZE_CubeIDE/Prog/App/led.h	
LED driver header file	2418
ARMCM33_STM32L5_Nucleo_L552ZE_GCC/Boot/led.h	
LED driver header file	2419
ARMCM33_STM32L5_Nucleo_L552ZE_GCC/Prog/led.h	
LED driver header file	2421
ARMCM33_STM32L5_Nucleo_L552ZE_IAR/Boot/led.h	
LED driver header file	2422
ARMCM33_STM32L5_Nucleo_L552ZE_IAR/Prog/led.h	
LED driver header file	2424
ARMCM33_STM32L5_Nucleo_L552ZE_Keil/Boot/led.h	
LED driver header file	2425
ARMCM33_STM32L5_Nucleo_L552ZE_Keil/Prog/led.h	
LED driver header file	2427
ARMCM3_EFM32_Olimex_EM32G880F128STK_GCC/Prog/led.h	
LED driver header file	2428

ARMCM3_EFM32_Olimex_EM32G880F128STK_IAR/Prog/led.h	
LED driver header file	2430
ARMCM3_LM3S_EK_LM3S6965_GCC/Prog/led.h	
LED driver header file	2432
ARMCM3_LM3S_EK_LM3S6965_IAR/Prog/led.h	
LED driver header file	2433
ARMCM3_LM3S_EK_LM3S8962_GCC/Prog/led.h	
LED driver header file	2434
ARMCM3_LM3S_EK_LM3S8962_IAR/Prog/led.h	
LED driver header file	2436
ARMCM3_STM32F1_Nucleo_F103RB_CubeIDE/Boot/App/led.h	
LED driver header file	2437
ARMCM3_STM32F1_Nucleo_F103RB_CubeIDE/Prog/App/led.h	
LED driver header file	2438
ARMCM3_STM32F1_Nucleo_F103RB_GCC/Boot/led.h	
LED driver header file	2440
ARMCM3_STM32F1_Nucleo_F103RB_GCC/Prog/led.h	
LED driver header file	2441
ARMCM3_STM32F1_Nucleo_F103RB_IAR/Boot/led.h	
LED driver header file	2443
ARMCM3_STM32F1_Nucleo_F103RB_IAR/Prog/led.h	
LED driver header file	2444
ARMCM3_STM32F1_Nucleo_F103RB_Keil/Boot/led.h	
LED driver header file	2446
ARMCM3_STM32F1_Nucleo_F103RB_Keil/Prog/led.h	
LED driver header file	2447
ARMCM3_STM32F1_Olimex_STM32H103_CubeIDE/Boot/App/led.h	
LED driver header file	2449
ARMCM3_STM32F1_Olimex_STM32H103_CubeIDE/Prog/App/led.h	
LED driver header file	2450
ARMCM3_STM32F1_Olimex_STM32H103_GCC/Boot/led.h	
LED driver header file	2452
ARMCM3_STM32F1_Olimex_STM32H103_GCC/Prog/led.h	
LED driver header file	2453
ARMCM3_STM32F1_Olimex_STM32H103_IAR/Boot/led.h	
LED driver header file	2455
ARMCM3_STM32F1_Olimex_STM32H103_IAR/Prog/led.h	
LED driver header file	2456
ARMCM3_STM32F1_Olimex_STM32H103_Keil/Boot/led.h	
LED driver header file	2458
ARMCM3_STM32F1_Olimex_STM32H103_Keil/Prog/led.h	
LED driver header file	2459
ARMCM3_STM32F1_Olimex_STM32P103_CubeIDE/Boot/App/led.h	
LED driver header file	2461
ARMCM3_STM32F1_Olimex_STM32P103_CubeIDE/Prog/App/led.h	
LED driver header file	2462
ARMCM3_STM32F1_Olimex_STM32P103_GCC/Boot/led.h	
LED driver header file	2464
ARMCM3_STM32F1_Olimex_STM32P103_GCC/Prog/led.h	
LED driver header file	2465
ARMCM3_STM32F1_Olimex_STM32P103_IAR/Boot/led.h	
LED driver header file	2467
ARMCM3_STM32F1_Olimex_STM32P103_IAR/Prog/led.h	
LED driver header file	2468
ARMCM3_STM32F1_Olimex_STM32P103_Keil/Boot/led.h	
LED driver header file	2470
ARMCM3_STM32F1_Olimex_STM32P103_Keil/Prog/led.h	
LED driver header file	2471

ARMCM3_STM32F1_Olimexino_STM32_CubeIDE/Boot/App/led.h	
LED driver header file	2473
ARMCM3_STM32F1_Olimexino_STM32_CubeIDE/Prog/App/led.h	
LED driver header file	2474
ARMCM3_STM32F1_Olimexino_STM32_GCC/Boot/led.h	
LED driver header file	2476
ARMCM3_STM32F1_Olimexino_STM32_GCC/Prog/led.h	
LED driver header file	2477
ARMCM3_STM32F1_Olimexino_STM32_IAR/Boot/led.h	
LED driver header file	2479
ARMCM3_STM32F1_Olimexino_STM32_IAR/Prog/led.h	
LED driver header file	2480
ARMCM3_STM32F1_Olimexino_STM32_Keil/Boot/led.h	
LED driver header file	2482
ARMCM3_STM32F1_Olimexino_STM32_Keil/Prog/led.h	
LED driver header file	2483
ARMCM3_STM32F2_Olimex_STM32P207_CubeIDE/Boot/App/led.h	
LED driver header file	2485
ARMCM3_STM32F2_Olimex_STM32P207_CubeIDE/Prog/App/led.h	
LED driver header file	2486
ARMCM3_STM32F2_Olimex_STM32P207_GCC/Boot/led.h	
LED driver header file	2488
ARMCM3_STM32F2_Olimex_STM32P207_GCC/Prog/led.h	
LED driver header file	2489
ARMCM3_STM32F2_Olimex_STM32P207_IAR/Boot/led.h	
LED driver header file	2491
ARMCM3_STM32F2_Olimex_STM32P207_IAR/Prog/led.h	
LED driver header file	2492
ARMCM3_STM32F2_Olimex_STM32P207_Keil/Boot/led.h	
LED driver header file	2494
ARMCM3_STM32F2_Olimex_STM32P207_Keil/Prog/led.h	
LED driver header file	2495
ARMCM4_S32K14_S32K144EVB_GCC/Boot/led.h	
LED driver header file	2497
ARMCM4_S32K14_S32K144EVB_GCC/Prog/led.h	
LED driver header file	2498
ARMCM4_S32K14_S32K144EVB_IAR/Boot/led.h	
LED driver header file	2500
ARMCM4_S32K14_S32K144EVB_IAR/Prog/led.h	
LED driver header file	2501
ARMCM4_STM32F3_Discovery_F303VC_CubeIDE/Boot/App/led.h	
LED driver header file	2503
ARMCM4_STM32F3_Discovery_F303VC_CubeIDE/Prog/App/led.h	
LED driver header file	2504
ARMCM4_STM32F3_Discovery_F303VC_GCC/Boot/led.h	
LED driver header file	2506
ARMCM4_STM32F3_Discovery_F303VC_GCC/Prog/led.h	
LED driver header file	2507
ARMCM4_STM32F3_Discovery_F303VC_IAR/Boot/led.h	
LED driver header file	2509
ARMCM4_STM32F3_Discovery_F303VC_IAR/Prog/led.h	
LED driver header file	2510
ARMCM4_STM32F3_Discovery_F303VC_Keil/Boot/led.h	
LED driver header file	2512
ARMCM4_STM32F3_Discovery_F303VC_Keil/Prog/led.h	
LED driver header file	2513
ARMCM4_STM32F3_Nucleo_F303K8_CubeIDE/Boot/App/led.h	
LED driver header file	2515

ARMCM4_STM32F3_Nucleo_F303K8_CubeIDE/Prog/App/led.h	
LED driver header file	2516
ARMCM4_STM32F3_Nucleo_F303K8_GCC/Boot/led.h	
LED driver header file	2518
ARMCM4_STM32F3_Nucleo_F303K8_GCC/Prog/led.h	
LED driver header file	2519
ARMCM4_STM32F3_Nucleo_F303K8_IAR/Boot/led.h	
LED driver header file	2521
ARMCM4_STM32F3_Nucleo_F303K8_IAR/Prog/led.h	
LED driver header file	2522
ARMCM4_STM32F3_Nucleo_F303K8_Keil/Boot/led.h	
LED driver header file	2524
ARMCM4_STM32F3_Nucleo_F303K8_Keil/Prog/led.h	
LED driver header file	2525
ARMCM4_STM32F4_Nucleo_F429ZI_CubeIDE/Boot/App/led.h	
LED driver header file	2527
ARMCM4_STM32F4_Nucleo_F429ZI_CubeIDE/Prog/App/led.h	
LED driver header file	2528
ARMCM4_STM32F4_Nucleo_F429ZI_GCC/Boot/led.h	
LED driver header file	2530
ARMCM4_STM32F4_Nucleo_F429ZI_IAR/Boot/led.h	
LED driver header file	2531
ARMCM4_STM32F4_Nucleo_F429ZI_Keil/Boot/led.h	
LED driver header file	2533
ARMCM4_STM32F4_Olimex_STM32P405_CubeIDE/Boot/App/led.h	
LED driver header file	2535
ARMCM4_STM32F4_Olimex_STM32P405_CubeIDE/Prog/App/led.h	
LED driver header file	2536
ARMCM4_STM32F4_Olimex_STM32P405_GCC/Boot/led.h	
LED driver header file	2537
ARMCM4_STM32F4_Olimex_STM32P405_GCC/Prog/led.h	
LED driver header file	2539
ARMCM4_STM32F4_Olimex_STM32P405_IAR/Boot/led.h	
LED driver header file	2540
ARMCM4_STM32F4_Olimex_STM32P405_IAR/Prog/led.h	
LED driver header file	2542
ARMCM4_STM32F4_Olimex_STM32P405_Keil/Boot/led.h	
LED driver header file	2543
ARMCM4_STM32F4_Olimex_STM32P405_Keil/Prog/led.h	
LED driver header file	2545
ARMCM4_STM32L4_Nucleo_L476RG_CubeIDE/Boot/App/led.h	
LED driver header file	2546
ARMCM4_STM32L4_Nucleo_L476RG_CubeIDE/Prog/App/led.h	
LED driver header file	2548
ARMCM4_STM32L4_Nucleo_L476RG_GCC/Boot/led.h	
LED driver header file	2549
ARMCM4_STM32L4_Nucleo_L476RG_GCC/Prog/led.h	
LED driver header file	2551
ARMCM4_STM32L4_Nucleo_L476RG_IAR/Boot/led.h	
LED driver header file	2552
ARMCM4_STM32L4_Nucleo_L476RG_IAR/Prog/led.h	
LED driver header file	2554
ARMCM4_STM32L4_Nucleo_L476RG_Keil/Boot/led.h	
LED driver header file	2555
ARMCM4_STM32L4_Nucleo_L476RG_Keil/Prog/led.h	
LED driver header file	2557
ARMCM4_TM4C_DK_TM4C123G_IAR/Prog/led.h	
LED driver header file	2558

ARMCM4_XMC4_XMC4700_Relax_Kit_GCC/Boot/led.h	
LED driver header file	2560
ARMCM4_XMC4_XMC4700_Relax_Kit_GCC/Prog/led.h	
LED driver header file	2561
ARMCM4_XMC4_XMC4700_Relax_Kit_IAR/Boot/led.h	
LED driver header file	2562
ARMCM4_XMC4_XMC4700_Relax_Kit_IAR/Prog/led.h	
LED driver header file	2564
ARMCM7_STM32F7_Nucleo_F746ZG_CubeIDE/Boot/App/led.h	
LED driver header file	2565
ARMCM7_STM32F7_Nucleo_F746ZG_CubeIDE/Prog/App/led.h	
LED driver header file	2567
ARMCM7_STM32F7_Nucleo_F746ZG_GCC/Boot/led.h	
LED driver header file	2568
ARMCM7_STM32F7_Nucleo_F746ZG_GCC/Prog/led.h	
LED driver header file	2570
ARMCM7_STM32F7_Nucleo_F746ZG_IAR/Boot/led.h	
LED driver header file	2571
ARMCM7_STM32F7_Nucleo_F746ZG_IAR/Prog/led.h	
LED driver header file	2573
ARMCM7_STM32F7_Nucleo_F746ZG_Keil/Boot/led.h	
LED driver header file	2574
ARMCM7_STM32F7_Nucleo_F746ZG_Keil/Prog/led.h	
LED driver header file	2576
ARMCM7_STM32F7_Nucleo_F767ZI_CubeIDE/Boot/App/led.h	
LED driver header file	2577
ARMCM7_STM32F7_Nucleo_F767ZI_CubeIDE/Prog/App/led.h	
LED driver header file	2579
ARMCM7_STM32F7_Nucleo_F767ZI_GCC/Boot/led.h	
LED driver header file	2580
ARMCM7_STM32F7_Nucleo_F767ZI_GCC/Prog/led.h	
LED driver header file	2582
ARMCM7_STM32F7_Nucleo_F767ZI_IAR/Boot/led.h	
LED driver header file	2583
ARMCM7_STM32F7_Nucleo_F767ZI_IAR/Prog/led.h	
LED driver header file	2585
ARMCM7_STM32F7_Nucleo_F767ZI_Keil/Boot/led.h	
LED driver header file	2586
ARMCM7_STM32F7_Nucleo_F767ZI_Keil/Prog/led.h	
LED driver header file	2588
ARMCM7_STM32H7_Nucleo_H743ZI_CubeIDE/Boot/App/led.h	
LED driver header file	2589
ARMCM7_STM32H7_Nucleo_H743ZI_CubeIDE/Prog/App/led.h	
LED driver header file	2591
ARMCM7_STM32H7_Nucleo_H743ZI_GCC/Boot/led.h	
LED driver header file	2592
ARMCM7_STM32H7_Nucleo_H743ZI_GCC/Prog/led.h	
LED driver header file	2594
ARMCM7_STM32H7_Nucleo_H743ZI_IAR/Boot/led.h	
LED driver header file	2595
ARMCM7_STM32H7_Nucleo_H743ZI_IAR/Prog/led.h	
LED driver header file	2597
ARMCM7_STM32H7_Nucleo_H743ZI_Keil/Boot/led.h	
LED driver header file	2598
ARMCM7_STM32H7_Nucleo_H743ZI_Keil/Prog/led.h	
LED driver header file	2600
HCS12_DevKit_S12G128_CodeWarrior/Boot/led.h	
LED driver header file	2601

HCS12_DevKit_S12G128_CodeWarrior/Prog/led.h	
LED driver header file	2603
HCS12_Evbplus_Dragon12p_CodeWarrior/Prog/led.h	
LED driver header file	2604
_template/Boot/main.c	
Bootloader application source file	2606
_template/Prog/main.c	
Demo program application source file	2607
ARMCM0_STM32F0_Discovery_STM32F051_GCC/Boot/main.c	
Bootloader application source file	2608
ARMCM0_STM32F0_Discovery_STM32F051_GCC/Prog/main.c	
Demo program application source file	2611
ARMCM0_STM32F0_Discovery_STM32F051_IAR/Boot/main.c	
Bootloader application source file	2613
ARMCM0_STM32F0_Discovery_STM32F051_IAR/Prog/main.c	
Demo program application source file	2615
ARMCM0_STM32F0_Discovery_STM32F051_Keil/Boot/main.c	
Bootloader application source file	2618
ARMCM0_STM32F0_Discovery_STM32F051_Keil/Prog/main.c	
Demo program application source file	2620
ARMCM0_STM32F0_Nucleo_F091RC_GCC/Boot/main.c	
Bootloader application source file	2622
ARMCM0_STM32F0_Nucleo_F091RC_GCC/Prog/main.c	
Demo program application source file	2625
ARMCM0_STM32F0_Nucleo_F091RC_IAR/Boot/main.c	
Bootloader application source file	2627
ARMCM0_STM32F0_Nucleo_F091RC_IAR/Prog/main.c	
Demo program application source file	2629
ARMCM0_STM32F0_Nucleo_F091RC_Keil/Boot/main.c	
Bootloader application source file	2632
ARMCM0_STM32F0_Nucleo_F091RC_Keil/Prog/main.c	
Demo program application source file	2634
ARMCM0_STM32G0_Nucleo_G071RB_GCC/Boot/main.c	
Bootloader application source file	2636
ARMCM0_STM32G0_Nucleo_G071RB_GCC/Prog/main.c	
Demo program application source file	2639
ARMCM0_STM32G0_Nucleo_G071RB_IAR/Boot/main.c	
Bootloader application source file	2641
ARMCM0_STM32G0_Nucleo_G071RB_IAR/Prog/main.c	
Demo program application source file	2644
ARMCM0_STM32G0_Nucleo_G071RB_Keil/Boot/main.c	
Bootloader application source file	2646
ARMCM0_STM32G0_Nucleo_G071RB_Keil/Prog/main.c	
Demo program application source file	2649
ARMCM0_XMC1_XMC1400_Boot_Kit_GCC/Boot/main.c	
Bootloader application source file	2651
ARMCM0_XMC1_XMC1400_Boot_Kit_GCC/Prog/main.c	
Demo program application source file	2653
ARMCM0_XMC1_XMC1400_Boot_Kit_IAR/Boot/main.c	
Bootloader application source file	2654
ARMCM0_XMC1_XMC1400_Boot_Kit_IAR/Prog/main.c	
Demo program application source file	2656
ARMCM33_STM32L5_Nucleo_L552ZE_GCC/Boot/main.c	
Bootloader application source file	2657
ARMCM33_STM32L5_Nucleo_L552ZE_GCC/Prog/main.c	
Demo program application source file	2659
ARMCM33_STM32L5_Nucleo_L552ZE_IAR/Boot/main.c	
Bootloader application source file	2662

ARMCM33_STM32L5_Nucleo_L552ZE_IAR/Prog/main.c	
Demo program application source file	2664
ARMCM33_STM32L5_Nucleo_L552ZE_Keil/Boot/main.c	
Bootloader application source file	2667
ARMCM33_STM32L5_Nucleo_L552ZE_Keil/Prog/main.c	
Demo program application source file	2669
ARMCM3_EFM32_Olimex_EM32G880F128STK_GCC/Boot/main.c	
Bootloader application source file	2672
ARMCM3_EFM32_Olimex_EM32G880F128STK_GCC/Prog/main.c	
Demo program application source file	2673
ARMCM3_EFM32_Olimex_EM32G880F128STK_IAR/Boot/main.c	
Bootloader application source file	2675
ARMCM3_EFM32_Olimex_EM32G880F128STK_IAR/Prog/main.c	
Demo program application source file	2676
ARMCM3_LM3S_EK_LM3S6965_GCC/Boot/main.c	
Bootloader application source file	2677
ARMCM3_LM3S_EK_LM3S6965_GCC/Prog/main.c	
Demo program application source file	2679
ARMCM3_LM3S_EK_LM3S6965_IAR/Boot/main.c	
Bootloader application source file	2680
ARMCM3_LM3S_EK_LM3S6965_IAR/Prog/main.c	
Demo program application source file	2681
ARMCM3_LM3S_EK_LM3S8962_GCC/Boot/main.c	
Bootloader application source file	2682
ARMCM3_LM3S_EK_LM3S8962_GCC/Prog/main.c	
Demo program application source file	2684
ARMCM3_LM3S_EK_LM3S8962_IAR/Boot/main.c	
Bootloader application source file	2685
ARMCM3_LM3S_EK_LM3S8962_IAR/Prog/main.c	
Demo program application source file	2686
ARMCM3_STM32F1_Nucleo_F103RB_GCC/Boot/main.c	
Bootloader application source file	2687
ARMCM3_STM32F1_Nucleo_F103RB_GCC/Prog/main.c	
Demo program application source file	2690
ARMCM3_STM32F1_Nucleo_F103RB_IAR/Boot/main.c	
Bootloader application source file	2692
ARMCM3_STM32F1_Nucleo_F103RB_IAR/Prog/main.c	
Demo program application source file	2695
ARMCM3_STM32F1_Nucleo_F103RB_Keil/Boot/main.c	
Bootloader application source file	2697
ARMCM3_STM32F1_Nucleo_F103RB_Keil/Prog/main.c	
Demo program application source file	2700
ARMCM3_STM32F1_Olimex_STM32H103_GCC/Boot/main.c	
Bootloader application source file	2702
ARMCM3_STM32F1_Olimex_STM32H103_GCC/Prog/main.c	
Demo program application source file	2705
ARMCM3_STM32F1_Olimex_STM32H103_IAR/Boot/main.c	
Bootloader application source file	2707
ARMCM3_STM32F1_Olimex_STM32H103_IAR/Prog/main.c	
Demo program application source file	2710
ARMCM3_STM32F1_Olimex_STM32H103_Keil/Boot/main.c	
Bootloader application source file	2712
ARMCM3_STM32F1_Olimex_STM32H103_Keil/Prog/main.c	
Demo program application source file	2715
ARMCM3_STM32F1_Olimex_STM32P103_GCC/Boot/main.c	
Bootloader application source file	2717
ARMCM3_STM32F1_Olimex_STM32P103_GCC/Prog/main.c	
Demo program application source file	2720

ARMCM3_STM32F1_Olimex_STM32P103_IAR/Boot/main.c	
Bootloader application source file	2722
ARMCM3_STM32F1_Olimex_STM32P103_IAR/Prog/main.c	
Demo program application source file	2725
ARMCM3_STM32F1_Olimex_STM32P103_Keil/Boot/main.c	
Bootloader application source file	2727
ARMCM3_STM32F1_Olimex_STM32P103_Keil/Prog/main.c	
Demo program application source file	2730
ARMCM3_STM32F1_Olimexino_STM32_GCC/Boot/main.c	
Bootloader application source file	2732
ARMCM3_STM32F1_Olimexino_STM32_GCC/Prog/main.c	
Demo program application source file	2735
ARMCM3_STM32F1_Olimexino_STM32_IAR/Boot/main.c	
Bootloader application source file	2737
ARMCM3_STM32F1_Olimexino_STM32_IAR/Prog/main.c	
Demo program application source file	2740
ARMCM3_STM32F1_Olimexino_STM32_Keil/Boot/main.c	
Bootloader application source file	2742
ARMCM3_STM32F1_Olimexino_STM32_Keil/Prog/main.c	
Demo program application source file	2745
ARMCM3_STM32F2_Olimex_STM32P207_GCC/Boot/main.c	
Bootloader application source file	2747
ARMCM3_STM32F2_Olimex_STM32P207_GCC/Prog/main.c	
Demo program application source file	2750
ARMCM3_STM32F2_Olimex_STM32P207_IAR/Boot/main.c	
Bootloader application source file	2752
ARMCM3_STM32F2_Olimex_STM32P207_IAR/Prog/main.c	
Demo program application source file	2755
ARMCM3_STM32F2_Olimex_STM32P207_Keil/Boot/main.c	
Bootloader application source file	2757
ARMCM3_STM32F2_Olimex_STM32P207_Keil/Prog/main.c	
Demo program application source file	2760
ARMCM4_S32K14_S32K144EVB_GCC/Boot/main.c	
Bootloader application source file	2762
ARMCM4_S32K14_S32K144EVB_GCC/Prog/main.c	
Demo program application source file	2764
ARMCM4_S32K14_S32K144EVB_IAR/Boot/main.c	
Bootloader application source file	2766
ARMCM4_S32K14_S32K144EVB_IAR/Prog/main.c	
Demo program application source file	2767
ARMCM4_STM32F3_Discovery_F303VC_GCC/Boot/main.c	
Bootloader application source file	2769
ARMCM4_STM32F3_Discovery_F303VC_GCC/Prog/main.c	
Demo program application source file	2772
ARMCM4_STM32F3_Discovery_F303VC_IAR/Boot/main.c	
Bootloader application source file	2775
ARMCM4_STM32F3_Discovery_F303VC_IAR/Prog/main.c	
Demo program application source file	2777
ARMCM4_STM32F3_Discovery_F303VC_Keil/Boot/main.c	
Bootloader application source file	2780
ARMCM4_STM32F3_Discovery_F303VC_Keil/Prog/main.c	
Demo program application source file	2782
ARMCM4_STM32F3_Nucleo_F303K8_GCC/Boot/main.c	
Bootloader application source file	2785
ARMCM4_STM32F3_Nucleo_F303K8_GCC/Prog/main.c	
Demo program application source file	2787
ARMCM4_STM32F3_Nucleo_F303K8_IAR/Boot/main.c	
Bootloader application source file	2790

ARMCM4_STM32F3_Nucleo_F303K8_IAR/Prog/main.c	
Demo program application source file	2792
ARMCM4_STM32F3_Nucleo_F303K8_Keil/Boot/main.c	
Bootloader application source file	2795
ARMCM4_STM32F3_Nucleo_F303K8_Keil/Prog/main.c	
Demo program application source file	2797
ARMCM4_STM32F4_Nucleo_F429ZI_GCC/Boot/main.c	
Bootloader application source file	2800
ARMCM4_STM32F4_Nucleo_F429ZI_GCC/Prog/main.c	
Demo program application source file	2802
ARMCM4_STM32F4_Nucleo_F429ZI_IAR/Boot/main.c	
Bootloader application source file	2805
ARMCM4_STM32F4_Nucleo_F429ZI_IAR/Prog/main.c	
Demo program application source file	2807
ARMCM4_STM32F4_Nucleo_F429ZI_Keil/Boot/main.c	
Bootloader application source file	2810
ARMCM4_STM32F4_Nucleo_F429ZI_Keil/Prog/main.c	
Demo program application source file	2812
ARMCM4_STM32F4_Olimex_STM32P405_GCC/Boot/main.c	
Bootloader application source file	2815
ARMCM4_STM32F4_Olimex_STM32P405_GCC/Prog/main.c	
Demo program application source file	2817
ARMCM4_STM32F4_Olimex_STM32P405_IAR/Boot/main.c	
Bootloader application source file	2820
ARMCM4_STM32F4_Olimex_STM32P405_IAR/Prog/main.c	
Demo program application source file	2822
ARMCM4_STM32F4_Olimex_STM32P405_Keil/Boot/main.c	
Bootloader application source file	2825
ARMCM4_STM32F4_Olimex_STM32P405_Keil/Prog/main.c	
Demo program application source file	2827
ARMCM4_STM32L4_Nucleo_L476RG_GCC/Boot/main.c	
Bootloader application source file	2830
ARMCM4_STM32L4_Nucleo_L476RG_GCC/Prog/main.c	
Demo program application source file	2832
ARMCM4_STM32L4_Nucleo_L476RG_IAR/Boot/main.c	
Bootloader application source file	2835
ARMCM4_STM32L4_Nucleo_L476RG_IAR/Prog/main.c	
Demo program application source file	2837
ARMCM4_STM32L4_Nucleo_L476RG_Keil/Boot/main.c	
Bootloader application source file	2840
ARMCM4_STM32L4_Nucleo_L476RG_Keil/Prog/main.c	
Demo program application source file	2842
ARMCM4_TM4C_DK_TM4C123G_IAR/Boot/main.c	
Bootloader application source file	2845
ARMCM4_TM4C_DK_TM4C123G_IAR/Prog/main.c	
Demo program application source file	2846
ARMCM4_XMC4_XMC4700_Relax_Kit_GCC/Boot/main.c	
Bootloader application source file	2847
ARMCM4_XMC4_XMC4700_Relax_Kit_GCC/Prog/main.c	
Demo program application source file	2849
ARMCM4_XMC4_XMC4700_Relax_Kit_IAR/Boot/main.c	
Bootloader application source file	2850
ARMCM4_XMC4_XMC4700_Relax_Kit_IAR/Prog/main.c	
Demo program application source file	2852
ARMCM7_STM32F7_Nucleo_F746ZG_GCC/Boot/main.c	
Bootloader application source file	2853
ARMCM7_STM32F7_Nucleo_F746ZG_GCC/Prog/main.c	
Demo program application source file	2856

ARMCM7_STM32F7_Nucleo_F746ZG_IAR/Boot/main.c	
Bootloader application source file	2858
ARMCM7_STM32F7_Nucleo_F746ZG_IAR/Prog/main.c	
Demo program application source file	2861
ARMCM7_STM32F7_Nucleo_F746ZG_Keil/Boot/main.c	
Bootloader application source file	2863
ARMCM7_STM32F7_Nucleo_F746ZG_Keil/Prog/main.c	
Demo program application source file	2866
ARMCM7_STM32F7_Nucleo_F767ZI_GCC/Boot/main.c	
Bootloader application source file	2868
ARMCM7_STM32F7_Nucleo_F767ZI_GCC/Prog/main.c	
Demo program application source file	2871
ARMCM7_STM32F7_Nucleo_F767ZI_IAR/Boot/main.c	
Bootloader application source file	2873
ARMCM7_STM32F7_Nucleo_F767ZI_IAR/Prog/main.c	
Demo program application source file	2876
ARMCM7_STM32F7_Nucleo_F767ZI_Keil/Boot/main.c	
Bootloader application source file	2878
ARMCM7_STM32F7_Nucleo_F767ZI_Keil/Prog/main.c	
Demo program application source file	2881
ARMCM7_STM32H7_Nucleo_H743ZI_GCC/Boot/main.c	
Bootloader application source file	2883
ARMCM7_STM32H7_Nucleo_H743ZI_GCC/Prog/main.c	
Demo program application source file	2886
ARMCM7_STM32H7_Nucleo_H743ZI_IAR/Boot/main.c	
Bootloader application source file	2888
ARMCM7_STM32H7_Nucleo_H743ZI_IAR/Prog/main.c	
Demo program application source file	2891
ARMCM7_STM32H7_Nucleo_H743ZI_Keil/Boot/main.c	
Bootloader application source file	2893
ARMCM7_STM32H7_Nucleo_H743ZI_Keil/Prog/main.c	
Demo program application source file	2896
HCS12_DevKit_S12G128_CodeWarrior/Boot/main.c	
Demo program application source file	2898
HCS12_DevKit_S12G128_CodeWarrior/Prog/main.c	
Demo program application source file	2900
HCS12_Evbplus_Dragon12p_CodeWarrior/Boot/main.c	
Bootloader application source file	2902
HCS12_Evbplus_Dragon12p_CodeWarrior/Prog/main.c	
Demo program application source file	2903
Demo/ARMCM3_LM3S_EK_LM3S6965_GCC/Prog/net.c	
Network application for the uIP TCP/IP stack	2905
Demo/ARMCM3_LM3S_EK_LM3S6965_IAR/Prog/net.c	
Network application for the uIP TCP/IP stack	2907
Demo/ARMCM4_STM32F4_Nucleo_F429ZI_CubeIDE/Prog/App/net.c	
Network application for the uIP TCP/IP stack	2909
Demo/ARMCM4_STM32F4_Nucleo_F429ZI_GCC/Prog/net.c	
Network application for the uIP TCP/IP stack	2911
Demo/ARMCM4_STM32F4_Nucleo_F429ZI_IAR/Prog/net.c	
Network application for the uIP TCP/IP stack	2913
Demo/ARMCM4_STM32F4_Nucleo_F429ZI_Keil/Prog/net.c	
Network application for the uIP TCP/IP stack	2915
Demo/ARMCM4_XMC4_XMC4700_Relax_Kit_GCC/Prog/net.c	
Network application for the uIP TCP/IP stack	2917
Demo/ARMCM4_XMC4_XMC4700_Relax_Kit_IAR/Prog/net.c	
Network application for the uIP TCP/IP stack	2919
Demo/ARMCM7_STM32F7_Nucleo_F767ZI_CubeIDE/Prog/App/net.c	
Network application for the uIP TCP/IP stack	2921

Demo/ARMCM7_STM32F7_Nucleo_F767ZI_GCC/Prog/net.c	
Network application for the uIP TCP/IP stack	2923
Demo/ARMCM7_STM32F7_Nucleo_F767ZI_IAR/Prog/net.c	
Network application for the uIP TCP/IP stack	2925
Demo/ARMCM7_STM32F7_Nucleo_F767ZI_Keil/Prog/net.c	
Network application for the uIP TCP/IP stack	2927
Source/net.c	
Bootloader TCP/IP network communication interface source file	2929
Demo/ARMCM3_LM3S_EK_LM3S6965_GCC/Prog/net.h	
Network application for the uIP TCP/IP stack	2932
Demo/ARMCM3_LM3S_EK_LM3S6965_IAR/Prog/net.h	
Network application for the uIP TCP/IP stack	2934
Demo/ARMCM4_STM32F4_Nucleo_F429ZI_CubelIDE/Prog/App/net.h	
Network application for the uIP TCP/IP stack	2936
Demo/ARMCM4_STM32F4_Nucleo_F429ZI_GCC/Prog/net.h	
Network application for the uIP TCP/IP stack	2938
Demo/ARMCM4_STM32F4_Nucleo_F429ZI_IAR/Prog/net.h	
Network application for the uIP TCP/IP stack	2940
Demo/ARMCM4_STM32F4_Nucleo_F429ZI_Keil/Prog/net.h	
Network application for the uIP TCP/IP stack	2942
Demo/ARMCM4_XMC4_XMC4700_Relax_Kit_GCC/Prog/net.h	
Network application for the uIP TCP/IP stack	2944
Demo/ARMCM4_XMC4_XMC4700_Relax_Kit_IAR/Prog/net.h	
Network application for the uIP TCP/IP stack	2946
Demo/ARMCM7_STM32F7_Nucleo_F767ZI_CubelIDE/Prog/App/net.h	
Network application for the uIP TCP/IP stack	2948
Demo/ARMCM7_STM32F7_Nucleo_F767ZI_GCC/Prog/net.h	
Network application for the uIP TCP/IP stack	2950
Demo/ARMCM7_STM32F7_Nucleo_F767ZI_IAR/Prog/net.h	
Network application for the uIP TCP/IP stack	2952
Demo/ARMCM7_STM32F7_Nucleo_F767ZI_Keil/Prog/net.h	
Network application for the uIP TCP/IP stack	2954
Source/net.h	
Bootloader TCP/IP network communication interface header file	2956
_template/nvm.c	
Bootloader non-volatile memory driver source file	2958
ARMCM0_S32K11/nvm.c	
Bootloader non-volatile memory driver source file	2964
ARMCM0_STM32F0/nvm.c	
Bootloader non-volatile memory driver source file	2969
ARMCM0_STM32G0/nvm.c	
Bootloader non-volatile memory driver source file	2975
ARMCM0_XMC1/nvm.c	
Bootloader non-volatile memory driver source file	2980
ARMCM33_STM32L5/nvm.c	
Bootloader non-volatile memory driver source file	2985
ARMCM3_EFM32/nvm.c	
Bootloader non-volatile memory driver source file	2990
ARMCM3_LM3S/nvm.c	
Bootloader non-volatile memory driver source file	2995
ARMCM3_STM32F1/nvm.c	
Bootloader non-volatile memory driver source file	3000
ARMCM3_STM32F2/nvm.c	
Bootloader non-volatile memory driver source file	3005
ARMCM4_S32K14/nvm.c	
Bootloader non-volatile memory driver source file	3010
ARMCM4_STM32F3/nvm.c	
Bootloader non-volatile memory driver source file	3015

ARMCM4_STM32F4/nvm.c	
Bootloader non-volatile memory driver source file	3020
ARMCM4_STM32L4/nvm.c	
Bootloader non-volatile memory driver source file	3025
ARMCM4_TM4C/nvm.c	
Bootloader non-volatile memory driver source file	3030
ARMCM4_XMC4/nvm.c	
Bootloader non-volatile memory driver source file	3035
ARMCM7_STM32F7/nvm.c	
Bootloader non-volatile memory driver source file	3040
ARMCM7_STM32H7/nvm.c	
Bootloader non-volatile memory driver source file	3045
HCS12/nvm.c	
Bootloader non-volatile memory driver source file	3050
nvm.h	
Bootloader non-volatile memory driver header file	3055
plausibility.h	
Bootloader plausibility check header file, for checking the configuration at compile time	3058
_template/rs232.c	
Bootloader RS232 communication interface source file	3058
ARMCM0_STM32F0/rs232.c	
Bootloader RS232 communication interface source file	3059
ARMCM0_STM32G0/rs232.c	
Bootloader RS232 communication interface source file	3059
ARMCM0_XMC1/rs232.c	
Bootloader RS232 communication interface source file	3060
ARMCM33_STM32L5/rs232.c	
Bootloader RS232 communication interface source file	3060
ARMCM3_EFM32/rs232.c	
Bootloader RS232 communication interface source file	3061
ARMCM3_LM3S/rs232.c	
Bootloader RS232 communication interface source file	3061
ARMCM3_STM32F1/rs232.c	
Bootloader RS232 communication interface source file	3062
ARMCM3_STM32F2/rs232.c	
Bootloader RS232 communication interface source file	3062
ARMCM4_STM32F3/rs232.c	
Bootloader RS232 communication interface source file	3063
ARMCM4_STM32F4/rs232.c	
Bootloader RS232 communication interface source file	3063
ARMCM4_STM32L4/rs232.c	
Bootloader RS232 communication interface source file	3064
ARMCM4_TM4C/rs232.c	
Bootloader RS232 communication interface source file	3064
ARMCM4_XMC4/rs232.c	
Bootloader RS232 communication interface source file	3065
ARMCM7_STM32F7/rs232.c	
Bootloader RS232 communication interface source file	3065
ARMCM7_STM32H7/rs232.c	
Bootloader RS232 communication interface source file	3066
HCS12/rs232.c	
Bootloader RS232 communication interface source file	3066
rs232.h	
Bootloader RS232 communication interface header file	3067
ARMCM3_LM3S_EK_LM3S6965_GCC/Boot/shared_params.c	
Shared RAM parameters source file	3067
ARMCM3_LM3S_EK_LM3S6965_GCC/Prog/shared_params.c	
Shared RAM parameters source file	3071

ARMCM3_LM3S_EK_LM3S6965_IAR/Boot/shared_params.c	
Shared RAM parameters source file	3076
ARMCM3_LM3S_EK_LM3S6965_IAR/Prog/shared_params.c	
Shared RAM parameters source file	3080
ARMCM4_STM32F4_Nucleo_F429ZI_CubeIDE/Boot/App/shared_params.c	
Shared RAM parameters source file	3084
ARMCM4_STM32F4_Nucleo_F429ZI_CubeIDE/Prog/App/shared_params.c	
Shared RAM parameters source file	3088
ARMCM4_STM32F4_Nucleo_F429ZI_GCC/Boot/shared_params.c	
Shared RAM parameters source file	3092
ARMCM4_STM32F4_Nucleo_F429ZI_GCC/Prog/shared_params.c	
Shared RAM parameters source file	3097
ARMCM4_STM32F4_Nucleo_F429ZI_IAR/Boot/shared_params.c	
Shared RAM parameters source file	3101
ARMCM4_STM32F4_Nucleo_F429ZI_IAR/Prog/shared_params.c	
Shared RAM parameters source file	3105
ARMCM4_STM32F4_Nucleo_F429ZI_Keil/Boot/shared_params.c	
Shared RAM parameters source file	3110
ARMCM4_STM32F4_Nucleo_F429ZI_Keil/Prog/shared_params.c	
Shared RAM parameters source file	3114
ARMCM4_XMC4_XMC4700_Relax_Kit_GCC/Boot/shared_params.c	
Shared RAM parameters source file	3118
ARMCM4_XMC4_XMC4700_Relax_Kit_GCC/Prog/shared_params.c	
Shared RAM parameters source file	3123
ARMCM4_XMC4_XMC4700_Relax_Kit_IAR/Boot/shared_params.c	
Shared RAM parameters source file	3127
ARMCM4_XMC4_XMC4700_Relax_Kit_IAR/Prog/shared_params.c	
Shared RAM parameters source file	3131
ARMCM7_STM32F7_Nucleo_F767ZI_CubeIDE/Boot/App/shared_params.c	
Shared RAM parameters source file	3136
ARMCM7_STM32F7_Nucleo_F767ZI_CubeIDE/Prog/App/shared_params.c	
Shared RAM parameters source file	3140
ARMCM7_STM32F7_Nucleo_F767ZI_GCC/Boot/shared_params.c	
Shared RAM parameters source file	3144
ARMCM7_STM32F7_Nucleo_F767ZI_GCC/Prog/shared_params.c	
Shared RAM parameters source file	3149
ARMCM7_STM32F7_Nucleo_F767ZI_IAR/Boot/shared_params.c	
Shared RAM parameters source file	3153
ARMCM7_STM32F7_Nucleo_F767ZI_IAR/Prog/shared_params.c	
Shared RAM parameters source file	3157
ARMCM7_STM32F7_Nucleo_F767ZI_Keil/Boot/shared_params.c	
Shared RAM parameters source file	3162
ARMCM7_STM32F7_Nucleo_F767ZI_Keil/Prog/shared_params.c	
Shared RAM parameters source file	3166
ARMCM3_LM3S_EK_LM3S6965_GCC/Boot/shared_params.h	
Shared RAM parameters header file	3170
ARMCM3_LM3S_EK_LM3S6965_GCC/Prog/shared_params.h	
Shared RAM parameters header file	3172
ARMCM3_LM3S_EK_LM3S6965_IAR/Boot/shared_params.h	
Shared RAM parameters header file	3175
ARMCM3_LM3S_EK_LM3S6965_IAR/Prog/shared_params.h	
Shared RAM parameters header file	3177
ARMCM4_STM32F4_Nucleo_F429ZI_CubeIDE/Boot/App/shared_params.h	
Shared RAM parameters header file	3179
ARMCM4_STM32F4_Nucleo_F429ZI_CubeIDE/Prog/App/shared_params.h	
Shared RAM parameters header file	3182
ARMCM4_STM32F4_Nucleo_F429ZI_GCC/Prog/shared_params.h	
Shared RAM parameters header file	3184

ARMCM4_STM32F4_Nucleo_F429ZI_IAR/Prog/shared_params.h	
Shared RAM parameters header file	3186
ARMCM4_STM32F4_Nucleo_F429ZI_Keil/Prog/shared_params.h	
Shared RAM parameters header file	3189
ARMCM4_XMC4_XMC4700_Relax_Kit_GCC/Boot/shared_params.h	
Shared RAM parameters header file	3191
ARMCM4_XMC4_XMC4700_Relax_Kit_GCC/Prog/shared_params.h	
Shared RAM parameters header file	3193
ARMCM4_XMC4_XMC4700_Relax_Kit_IAR/Boot/shared_params.h	
Shared RAM parameters header file	3196
ARMCM4_XMC4_XMC4700_Relax_Kit_IAR/Prog/shared_params.h	
Shared RAM parameters header file	3198
ARMCM7_STM32F7_Nucleo_F767ZI_CubelDE/Boot/App/shared_params.h	
Shared RAM parameters header file	3200
ARMCM7_STM32F7_Nucleo_F767ZI_CubelDE/Prog/App/shared_params.h	
Shared RAM parameters header file	3203
ARMCM7_STM32F7_Nucleo_F767ZI_GCC/Boot/shared_params.h	
Shared RAM parameters header file	3205
ARMCM7_STM32F7_Nucleo_F767ZI_GCC/Prog/shared_params.h	
Shared RAM parameters header file	3207
ARMCM7_STM32F7_Nucleo_F767ZI_IAR/Boot/shared_params.h	
Shared RAM parameters header file	3210
ARMCM7_STM32F7_Nucleo_F767ZI_IAR/Prog/shared_params.h	
Shared RAM parameters header file	3212
ARMCM7_STM32F7_Nucleo_F767ZI_Keil/Boot/shared_params.h	
Shared RAM parameters header file	3214
ARMCM7_STM32F7_Nucleo_F767ZI_Keil/Prog/shared_params.h	
Shared RAM parameters header file	3217
ARMCM3_LM3S_EK_LM3S6965_GCC/Prog/time.c	
Timer driver source file	3219
ARMCM3_LM3S_EK_LM3S6965_IAR/Prog/time.c	
Timer driver source file	3221
ARMCM3_LM3S_EK_LM3S8962_GCC/Prog/time.c	
Timer driver source file	3223
ARMCM3_LM3S_EK_LM3S8962_IAR/Prog/time.c	
Timer driver source file	3226
ARMCM4_TM4C_DK_TM4C123G_IAR/Prog/time.c	
Timer driver source file	3228
ARMCM3_LM3S_EK_LM3S6965_GCC/Prog/time.h	
Timer driver header file	3230
ARMCM3_LM3S_EK_LM3S6965_IAR/Prog/time.h	
Timer driver header file	3232
ARMCM3_LM3S_EK_LM3S8962_GCC/Prog/time.h	
Timer driver header file	3235
ARMCM3_LM3S_EK_LM3S8962_IAR/Prog/time.h	
Timer driver header file	3237
ARMCM4_TM4C_DK_TM4C123G_IAR/Prog/time.h	
Timer driver header file	3239
Demo/_template/Prog/timer.c	
Timer driver source file	3242
Demo/ARMCM0_STM32F0_Discovery_STM32F051_CubelDE/Prog/App/timer.c	
Timer driver source file	3244
Demo/ARMCM0_STM32F0_Discovery_STM32F051_GCC/Prog/timer.c	
Timer driver source file	3245
Demo/ARMCM0_STM32F0_Discovery_STM32F051_IAR/Prog/timer.c	
Timer driver source file	3246
Demo/ARMCM0_STM32F0_Discovery_STM32F051_Keil/Prog/timer.c	
Timer driver source file	3248

Demo/ARMCM0_STM32F0_Nucleo_F091RC_CubeIDE/Prog/App/timer.c	
Timer driver source file	3249
Demo/ARMCM0_STM32F0_Nucleo_F091RC_GCC/Prog/timer.c	
Timer driver source file	3251
Demo/ARMCM0_STM32F0_Nucleo_F091RC_IAR/Prog/timer.c	
Timer driver source file	3252
Demo/ARMCM0_STM32F0_Nucleo_F091RC_Keil/Prog/timer.c	
Timer driver source file	3254
Demo/ARMCM0_STM32G0_Nucleo_G071RB_CubeIDE/Prog/App/timer.c	
Timer driver source file	3255
Demo/ARMCM0_STM32G0_Nucleo_G071RB_GCC/Prog/timer.c	
Timer driver source file	3256
Demo/ARMCM0_STM32G0_Nucleo_G071RB_IAR/Prog/timer.c	
Timer driver source file	3258
Demo/ARMCM0_STM32G0_Nucleo_G071RB_Keil/Prog/timer.c	
Timer driver source file	3259
Demo/ARMCM0_XMC1_XMC1400_Boot_Kit_GCC/Prog/timer.c	
Timer driver source file	3261
Demo/ARMCM0_XMC1_XMC1400_Boot_Kit_IAR/Prog/timer.c	
Timer driver source file	3263
Demo/ARMCM33_STM32L5_Nucleo_L552ZE_CubeIDE/Prog/App/timer.c	
Timer driver source file	3265
Demo/ARMCM33_STM32L5_Nucleo_L552ZE_GCC/Prog/timer.c	
Timer driver source file	3267
Demo/ARMCM33_STM32L5_Nucleo_L552ZE_IAR/Prog/timer.c	
Timer driver source file	3268
Demo/ARMCM33_STM32L5_Nucleo_L552ZE_Keil/Prog/timer.c	
Timer driver source file	3270
Demo/ARMCM3_EFM32_Olimex_EM32G880F128STK_GCC/Prog/timer.c	
Timer driver source file	3271
Demo/ARMCM3_EFM32_Olimex_EM32G880F128STK_IAR/Prog/timer.c	
Timer driver source file	3273
Demo/ARMCM3_STM32F1_Nucleo_F103RB_CubeIDE/Prog/App/timer.c	
Timer driver source file	3276
Demo/ARMCM3_STM32F1_Nucleo_F103RB_GCC/Prog/timer.c	
Timer driver source file	3277
Demo/ARMCM3_STM32F1_Nucleo_F103RB_IAR/Prog/timer.c	
Timer driver source file	3278
Demo/ARMCM3_STM32F1_Nucleo_F103RB_Keil/Prog/timer.c	
Timer driver source file	3280
Demo/ARMCM3_STM32F1_Olimex_STM32H103_CubeIDE/Prog/App/timer.c	
Timer driver source file	3282
Demo/ARMCM3_STM32F1_Olimex_STM32H103_GCC/Prog/timer.c	
Timer driver source file	3283
Demo/ARMCM3_STM32F1_Olimex_STM32H103_IAR/Prog/timer.c	
Timer driver source file	3285
Demo/ARMCM3_STM32F1_Olimex_STM32H103_Keil/Prog/timer.c	
Timer driver source file	3286
Demo/ARMCM3_STM32F1_Olimex_STM32P103_CubeIDE/Prog/App/timer.c	
Timer driver source file	3288
Demo/ARMCM3_STM32F1_Olimex_STM32P103_GCC/Prog/timer.c	
Timer driver source file	3289
Demo/ARMCM3_STM32F1_Olimex_STM32P103_IAR/Prog/timer.c	
Timer driver source file	3291
Demo/ARMCM3_STM32F1_Olimex_STM32P103_Keil/Prog/timer.c	
Timer driver source file	3292
Demo/ARMCM3_STM32F1_Olimexino_STM32_CubeIDE/Prog/App/timer.c	
Timer driver source file	3294

Demo/ARMCM3_STM32F1_Olimexino_STM32_GCC/Prog/timer.c	
Timer driver source file	3295
Demo/ARMCM3_STM32F1_Olimexino_STM32_IAR/Prog/timer.c	
Timer driver source file	3296
Demo/ARMCM3_STM32F1_Olimexino_STM32_Keil/Prog/timer.c	
Timer driver source file	3298
Demo/ARMCM3_STM32F2_Olimex_STM32P207_CubeIDE/Prog/App/timer.c	
Timer driver source file	3299
Demo/ARMCM3_STM32F2_Olimex_STM32P207_GCC/Prog/timer.c	
Timer driver source file	3301
Demo/ARMCM3_STM32F2_Olimex_STM32P207_IAR/Prog/timer.c	
Timer driver source file	3302
Demo/ARMCM3_STM32F2_Olimex_STM32P207_Keil/Prog/timer.c	
Timer driver source file	3304
Demo/ARMCM4_S32K14_S32K144EVB_GCC/Prog/timer.c	
Timer driver source file	3305
Demo/ARMCM4_S32K14_S32K144EVB_IAR/Prog/timer.c	
Timer driver source file	3307
Demo/ARMCM4_STM32F3_Discovery_F303VC_CubeIDE/Prog/App/timer.c	
Timer driver source file	3308
Demo/ARMCM4_STM32F3_Discovery_F303VC_GCC/Prog/timer.c	
Timer driver source file	3310
Demo/ARMCM4_STM32F3_Discovery_F303VC_IAR/Prog/timer.c	
Timer driver source file	3311
Demo/ARMCM4_STM32F3_Discovery_F303VC_Keil/Prog/timer.c	
Timer driver source file	3313
Demo/ARMCM4_STM32F3_Nucleo_F303K8_CubeIDE/Prog/App/timer.c	
Timer driver source file	3315
Demo/ARMCM4_STM32F3_Nucleo_F303K8_GCC/Prog/timer.c	
Timer driver source file	3316
Demo/ARMCM4_STM32F3_Nucleo_F303K8_IAR/Prog/timer.c	
Timer driver source file	3317
Demo/ARMCM4_STM32F3_Nucleo_F303K8_Keil/Prog/timer.c	
Timer driver source file	3319
Demo/ARMCM4_STM32F4_Nucleo_F429ZI_CubeIDE/Prog/App/timer.c	
Timer driver source file	3320
Demo/ARMCM4_STM32F4_Nucleo_F429ZI_GCC/Prog/timer.c	
Timer driver source file	3322
Demo/ARMCM4_STM32F4_Nucleo_F429ZI_IAR/Prog/timer.c	
Timer driver source file	3323
Demo/ARMCM4_STM32F4_Nucleo_F429ZI_Keil/Prog/timer.c	
Timer driver source file	3325
Demo/ARMCM4_STM32F4_Olimex_STM32P405_CubeIDE/Prog/App/timer.c	
Timer driver source file	3326
Demo/ARMCM4_STM32F4_Olimex_STM32P405_GCC/Prog/timer.c	
Timer driver source file	3327
Demo/ARMCM4_STM32F4_Olimex_STM32P405_IAR/Prog/timer.c	
Timer driver source file	3329
Demo/ARMCM4_STM32F4_Olimex_STM32P405_Keil/Prog/timer.c	
Timer driver source file	3330
Demo/ARMCM4_STM32L4_Nucleo_L476RG_CubeIDE/Prog/App/timer.c	
Timer driver source file	3332
Demo/ARMCM4_STM32L4_Nucleo_L476RG_GCC/Prog/timer.c	
Timer driver source file	3333
Demo/ARMCM4_STM32L4_Nucleo_L476RG_IAR/Prog/timer.c	
Timer driver source file	3335
Demo/ARMCM4_STM32L4_Nucleo_L476RG_Keil/Prog/timer.c	
Timer driver source file	3336

Demo/ARMCM4_XMC4_XMC4700_Relax_Kit_GCC/Prog/timer.c	
Timer driver source file	3338
Demo/ARMCM4_XMC4_XMC4700_Relax_Kit_IAR/Prog/timer.c	
Timer driver source file	3340
Demo/ARMCM7_STM32F7_Nucleo_F746ZG_CubeIDE/Prog/App/timer.c	
Timer driver source file	3342
Demo/ARMCM7_STM32F7_Nucleo_F746ZG_GCC/Prog/timer.c	
Timer driver source file	3344
Demo/ARMCM7_STM32F7_Nucleo_F746ZG_IAR/Prog/timer.c	
Timer driver source file	3345
Demo/ARMCM7_STM32F7_Nucleo_F746ZG_Keil/Prog/timer.c	
Timer driver source file	3347
Demo/ARMCM7_STM32F7_Nucleo_F767ZI_CubeIDE/Prog/App/timer.c	
Timer driver source file	3348
Demo/ARMCM7_STM32F7_Nucleo_F767ZI_GCC/Prog/timer.c	
Timer driver source file	3349
Demo/ARMCM7_STM32F7_Nucleo_F767ZI_IAR/Prog/timer.c	
Timer driver source file	3351
Demo/ARMCM7_STM32F7_Nucleo_F767ZI_Keil/Prog/timer.c	
Timer driver source file	3352
Demo/ARMCM7_STM32H7_Nucleo_H743ZI_CubeIDE/Prog/App/timer.c	
Timer driver source file	3354
Demo/ARMCM7_STM32H7_Nucleo_H743ZI_GCC/Prog/timer.c	
Timer driver source file	3355
Demo/ARMCM7_STM32H7_Nucleo_H743ZI_IAR/Prog/timer.c	
Timer driver source file	3357
Demo/ARMCM7_STM32H7_Nucleo_H743ZI_Keil/Prog/timer.c	
Timer driver source file	3358
Demo/HCS12_DevKit_S12G128_CodeWarrior/Prog/timer.c	
Timer driver source file	3360
Demo/HCS12_Evbplus_Dragon12p_CodeWarrior/Prog/timer.c	
Timer driver source file	3362
Source/_template/timer.c	
Bootloader timer driver source file	3364
Source/ARMCM0_S32K11/timer.c	
Bootloader timer driver source file	3366
Source/ARMCM0_STM32F0/timer.c	
Bootloader timer driver source file	3368
Source/ARMCM0_STM32G0/timer.c	
Bootloader timer driver source file	3371
Source/ARMCM0_XMC1/timer.c	
Bootloader timer driver source file	3374
Source/ARMCM33_STM32L5/timer.c	
Bootloader timer driver source file	3377
Source/ARMCM3_EFM32/timer.c	
Bootloader timer driver source file	3380
Source/ARMCM3_LM3S/timer.c	
Bootloader timer driver source file	3382
Source/ARMCM3_STM32F1/timer.c	
Bootloader timer driver source file	3385
Source/ARMCM3_STM32F2/timer.c	
Bootloader timer driver source file	3388
Source/ARMCM4_S32K14/timer.c	
Bootloader timer driver source file	3391
Source/ARMCM4_STM32F3/timer.c	
Bootloader timer driver source file	3393
Source/ARMCM4_STM32F4/timer.c	
Bootloader timer driver source file	3396

Source/ARMCM4_STM32L4/timer.c	
Bootloader timer driver source file	3399
Source/ARMCM4_TM4C/timer.c	
Bootloader timer driver source file	3402
Source/ARMCM4_XMC4/timer.c	
Bootloader timer driver source file	3404
Source/ARMCM7_STM32F7/timer.c	
Bootloader timer driver source file	3407
Source/ARMCM7_STM32H7/timer.c	
Bootloader timer driver source file	3410
Source/HCS12/timer.c	
Bootloader timer driver source file	3413
Demo/_template/Prog/timer.h	
Timer driver header file	3415
Demo/ARMCM0_STM32F0_Discovery_STM32F051_CubeIDE/Prog/App/timer.h	
Timer driver header file	3417
Demo/ARMCM0_STM32F0_Discovery_STM32F051_GCC/Prog/timer.h	
Timer driver header file	3418
Demo/ARMCM0_STM32F0_Discovery_STM32F051_IAR/Prog/timer.h	
Timer driver header file	3419
Demo/ARMCM0_STM32F0_Discovery_STM32F051_Keil/Prog/timer.h	
Timer driver header file	3421
Demo/ARMCM0_STM32F0_Nucleo_F091RC_CubeIDE/Prog/App/timer.h	
Timer driver header file	3422
Demo/ARMCM0_STM32F0_Nucleo_F091RC_GCC/Prog/timer.h	
Timer driver header file	3423
Demo/ARMCM0_STM32F0_Nucleo_F091RC_IAR/Prog/timer.h	
Timer driver header file	3425
Demo/ARMCM0_STM32F0_Nucleo_F091RC_Keil/Prog/timer.h	
Timer driver header file	3426
Demo/ARMCM0_STM32G0_Nucleo_G071RB_CubeIDE/Prog/App/timer.h	
Timer driver header file	3427
Demo/ARMCM0_STM32G0_Nucleo_G071RB_GCC/Prog/timer.h	
Timer driver header file	3429
Demo/ARMCM0_STM32G0_Nucleo_G071RB_IAR/Prog/timer.h	
Timer driver header file	3430
Demo/ARMCM0_STM32G0_Nucleo_G071RB_Keil/Prog/timer.h	
Timer driver header file	3431
Demo/ARMCM0_XMC1_XMC1400_Boot_Kit_GCC/Prog/timer.h	
Timer driver header file	3433
Demo/ARMCM0_XMC1_XMC1400_Boot_Kit_IAR/Prog/timer.h	
Timer driver header file	3435
Demo/ARMCM33_STM32L5_Nucleo_L552ZE_CubeIDE/Prog/App/timer.h	
Timer driver header file	3437
Demo/ARMCM33_STM32L5_Nucleo_L552ZE_GCC/Prog/timer.h	
Timer driver header file	3438
Demo/ARMCM33_STM32L5_Nucleo_L552ZE_IAR/Prog/timer.h	
Timer driver header file	3440
Demo/ARMCM33_STM32L5_Nucleo_L552ZE_Keil/Prog/timer.h	
Timer driver header file	3441
Demo/ARMCM3_EFM32_Olimex_EM32G880F128STK_GCC/Prog/timer.h	
Timer driver header file	3442
Demo/ARMCM3_EFM32_Olimex_EM32G880F128STK_IAR/Prog/timer.h	
Timer driver header file	3445
Demo/ARMCM3_STM32F1_Nucleo_F103RB_CubeIDE/Prog/App/timer.h	
Timer driver header file	3447
Demo/ARMCM3_STM32F1_Nucleo_F103RB_GCC/Prog/timer.h	
Timer driver header file	3448

Demo/ARMCM3_STM32F1_Nucleo_F103RB_IAR/Prog/timer.h	
Timer driver header file	3450
Demo/ARMCM3_STM32F1_Nucleo_F103RB_Keil/Prog/timer.h	
Timer driver header file	3451
Demo/ARMCM3_STM32F1_Olimex_STM32H103_CubeIDE/Prog/App/timer.h	
Timer driver header file	3452
Demo/ARMCM3_STM32F1_Olimex_STM32H103_GCC/Prog/timer.h	
Timer driver header file	3454
Demo/ARMCM3_STM32F1_Olimex_STM32H103_IAR/Prog/timer.h	
Timer driver header file	3455
Demo/ARMCM3_STM32F1_Olimex_STM32H103_Keil/Prog/timer.h	
Timer driver header file	3456
Demo/ARMCM3_STM32F1_Olimex_STM32P103_CubeIDE/Prog/App/timer.h	
Timer driver header file	3458
Demo/ARMCM3_STM32F1_Olimex_STM32P103_GCC/Prog/timer.h	
Timer driver header file	3459
Demo/ARMCM3_STM32F1_Olimex_STM32P103_IAR/Prog/timer.h	
Timer driver header file	3460
Demo/ARMCM3_STM32F1_Olimex_STM32P103_Keil/Prog/timer.h	
Timer driver header file	3462
Demo/ARMCM3_STM32F1_Olimexino_STM32_CubeIDE/Prog/App/timer.h	
Timer driver header file	3463
Demo/ARMCM3_STM32F1_Olimexino_STM32_GCC/Prog/timer.h	
Timer driver header file	3464
Demo/ARMCM3_STM32F1_Olimexino_STM32_IAR/Prog/timer.h	
Timer driver header file	3466
Demo/ARMCM3_STM32F1_Olimexino_STM32_Keil/Prog/timer.h	
Timer driver header file	3467
Demo/ARMCM3_STM32F2_Olimex_STM32P207_CubeIDE/Prog/App/timer.h	
Timer driver header file	3468
Demo/ARMCM3_STM32F2_Olimex_STM32P207_GCC/Prog/timer.h	
Timer driver header file	3470
Demo/ARMCM3_STM32F2_Olimex_STM32P207_IAR/Prog/timer.h	
Timer driver header file	3471
Demo/ARMCM3_STM32F2_Olimex_STM32P207_Keil/Prog/timer.h	
Timer driver header file	3472
Demo/ARMCM4_S32K14_S32K144EVB_GCC/Prog/timer.h	
Timer driver header file	3474
Demo/ARMCM4_S32K14_S32K144EVB_IAR/Prog/timer.h	
Timer driver header file	3475
Demo/ARMCM4_STM32F3_Discovery_F303VC_CubeIDE/Prog/App/timer.h	
Timer driver header file	3476
Demo/ARMCM4_STM32F3_Discovery_F303VC_GCC/Prog/timer.h	
Timer driver header file	3478
Demo/ARMCM4_STM32F3_Discovery_F303VC_IAR/Prog/timer.h	
Timer driver header file	3479
Demo/ARMCM4_STM32F3_Discovery_F303VC_Keil/Prog/timer.h	
Timer driver header file	3480
Demo/ARMCM4_STM32F3_Nucleo_F303K8_CubeIDE/Prog/App/timer.h	
Timer driver header file	3482
Demo/ARMCM4_STM32F3_Nucleo_F303K8_GCC/Prog/timer.h	
Timer driver header file	3483
Demo/ARMCM4_STM32F3_Nucleo_F303K8_IAR/Prog/timer.h	
Timer driver header file	3484
Demo/ARMCM4_STM32F3_Nucleo_F303K8_Keil/Prog/timer.h	
Timer driver header file	3486
Demo/ARMCM4_STM32F4_Nucleo_F429ZI_CubeIDE/Prog/App/timer.h	
Timer driver header file	3487

Demo/ARMCM4_STM32F4_Nucleo_F429ZI_GCC/Prog/timer.h	
Timer driver header file	3488
Demo/ARMCM4_STM32F4_Nucleo_F429ZI_IAR/Prog/timer.h	
Timer driver header file	3490
Demo/ARMCM4_STM32F4_Nucleo_F429ZI_Keil/Prog/timer.h	
Timer driver header file	3491
Demo/ARMCM4_STM32F4_Olimex_STM32P405_CubeIDE/Prog/App/timer.h	
Timer driver header file	3492
Demo/ARMCM4_STM32F4_Olimex_STM32P405_GCC/Prog/timer.h	
Timer driver header file	3494
Demo/ARMCM4_STM32F4_Olimex_STM32P405_IAR/Prog/timer.h	
Timer driver header file	3495
Demo/ARMCM4_STM32F4_Olimex_STM32P405_Keil/Prog/timer.h	
Timer driver header file	3496
Demo/ARMCM4_STM32L4_Nucleo_L476RG_CubeIDE/Prog/App/timer.h	
Timer driver header file	3498
Demo/ARMCM4_STM32L4_Nucleo_L476RG_GCC/Prog/timer.h	
Timer driver header file	3499
Demo/ARMCM4_STM32L4_Nucleo_L476RG_IAR/Prog/timer.h	
Timer driver header file	3500
Demo/ARMCM4_STM32L4_Nucleo_L476RG_Keil/Prog/timer.h	
Timer driver header file	3502
Demo/ARMCM4_XMC4_XMC4700_Relax_Kit_GCC/Prog/timer.h	
Timer driver header file	3503
Demo/ARMCM4_XMC4_XMC4700_Relax_Kit_IAR/Prog/timer.h	
Timer driver header file	3505
Demo/ARMCM7_STM32F7_Nucleo_F746ZG_CubeIDE/Prog/App/timer.h	
Timer driver header file	3507
Demo/ARMCM7_STM32F7_Nucleo_F746ZG_GCC/Prog/timer.h	
Timer driver header file	3508
Demo/ARMCM7_STM32F7_Nucleo_F746ZG_IAR/Prog/timer.h	
Timer driver header file	3510
Demo/ARMCM7_STM32F7_Nucleo_F746ZG_Keil/Prog/timer.h	
Timer driver header file	3511
Demo/ARMCM7_STM32F7_Nucleo_F767ZI_CubeIDE/Prog/App/timer.h	
Timer driver header file	3512
Demo/ARMCM7_STM32F7_Nucleo_F767ZI_GCC/Prog/timer.h	
Timer driver header file	3514
Demo/ARMCM7_STM32F7_Nucleo_F767ZI_IAR/Prog/timer.h	
Timer driver header file	3515
Demo/ARMCM7_STM32F7_Nucleo_F767ZI_Keil/Prog/timer.h	
Timer driver header file	3516
Demo/ARMCM7_STM32H7_Nucleo_H743ZI_CubeIDE/Prog/App/timer.h	
Timer driver header file	3518
Demo/ARMCM7_STM32H7_Nucleo_H743ZI_GCC/Prog/timer.h	
Timer driver header file	3519
Demo/ARMCM7_STM32H7_Nucleo_H743ZI_IAR/Prog/timer.h	
Timer driver header file	3520
Demo/ARMCM7_STM32H7_Nucleo_H743ZI_Keil/Prog/timer.h	
Timer driver header file	3522
Demo/HCS12_DevKit_S12G128_CodeWarrior/Prog/timer.h	
Timer driver header file	3523
Demo/HCS12_Evbplus_Dragon12p_CodeWarrior/Prog/timer.h	
Timer driver header file	3525
Source/timer.h	
Bootloader timer driver header file	3528
_template/types.h	
Bootloader types header file	3530

ARMCM0_S32K11/types.h	
Bootloader types header file	3531
ARMCM0_STM32F0/types.h	
Bootloader types header file	3533
ARMCM0_STM32G0/types.h	
Bootloader types header file	3535
ARMCM0_XMC1/types.h	
Bootloader types header file	3538
ARMCM33_STM32L5/types.h	
Bootloader types header file	3539
ARMCM3_EFM32/types.h	
Bootloader types header file	3542
ARMCM3_LM3S/types.h	
Bootloader types header file	3543
ARMCM3_STM32F1/types.h	
Bootloader types header file	3545
ARMCM3_STM32F2/types.h	
Bootloader types header file	3547
ARMCM4_S32K14/types.h	
Bootloader types header file	3549
ARMCM4_STM32F3/types.h	
Bootloader types header file	3551
ARMCM4_STM32F4/types.h	
Bootloader types header file	3553
ARMCM4_STM32L4/types.h	
Bootloader types header file	3555
ARMCM4_TM4C/types.h	
Bootloader types header file	3557
ARMCM4_XMC4/types.h	
Bootloader types header file	3559
ARMCM7_STM32F7/types.h	
Bootloader types header file	3561
ARMCM7_STM32H7/types.h	
Bootloader types header file	3563
HCS12/types.h	
Bootloader types header file	3565
_template/usb.c	
Bootloader USB communication interface source file	3568
ARMCM33_STM32L5/usb.c	
Bootloader USB communication interface source file	3574
ARMCM3_STM32F1/usb.c	
Bootloader USB communication interface source file	3580
ARMCM4_STM32F3/usb.c	
Bootloader USB communication interface source file	3587
ARMCM4_STM32F4/usb.c	
Bootloader USB communication interface source file	3593
ARMCM4_TM4C/usb.c	
Bootloader USB communication interface source file	3599
ARMCM7_STM32F7/usb.c	
Bootloader USB communication interface source file	3607
ARMCM7_STM32H7/usb.c	
Bootloader USB communication interface source file	3613
usb.h	
Bootloader USB communication interface header file	3620
ARMCM3_EFM32_Olimex_EM32G880F128STK_GCC/Boot/vectors.c	
Bootloader interrupt vector table source file	3623
ARMCM3_EFM32_Olimex_EM32G880F128STK_GCC/Prog/vectors.c	
Demo program interrupt vectors source file	3624

ARMCM3_EFM32_Olimex_EM32G880F128STK_IAR/Boot/vectors.c	
Bootloader interrupt vector table source file	3626
ARMCM3_EFM32_Olimex_EM32G880F128STK_IAR/Prog/vectors.c	
Demo program interrupt vectors source file	3627
ARMCM3_LM3S_EK_LM3S6965_GCC/Boot/vectors.c	
Bootloader application source file	3628
ARMCM3_LM3S_EK_LM3S6965_GCC/Prog/vectors.c	
Demo program interrupt vectors source file	3630
ARMCM3_LM3S_EK_LM3S6965_IAR/Boot/vectors.c	
Bootloader interrupt vector table source file	3631
ARMCM3_LM3S_EK_LM3S6965_IAR/Prog/vectors.c	
Demo program interrupt vectors source file	3633
ARMCM3_LM3S_EK_LM3S8962_GCC/Boot/vectors.c	
Bootloader interrupt vector table source file	3634
ARMCM3_LM3S_EK_LM3S8962_GCC/Prog/vectors.c	
Demo program interrupt vectors source file	3635
ARMCM3_LM3S_EK_LM3S8962_IAR/Boot/vectors.c	
Bootloader interrupt vector table source file	3637
ARMCM3_LM3S_EK_LM3S8962_IAR/Prog/vectors.c	
Demo program interrupt vectors source file	3638
ARMCM4_TM4C_DK_TM4C123G_IAR/Boot/vectors.c	
Bootloader interrupt vector table source file	3639
ARMCM4_TM4C_DK_TM4C123G_IAR/Prog/vectors.c	
Demo program interrupt vectors source file	3641
HCS12_DevKit_S12G128_CodeWarrior/Boot/vectors.c	
Demo program interrupt vectors source file	3642
HCS12_DevKit_S12G128_CodeWarrior/Prog/vectors.c	
Demo program interrupt vectors source file	3663
HCS12_Evbplus_Dragon12p_CodeWarrior/Boot/vectors.c	
Bootloader interrupt vector table source file	3664
HCS12_Evbplus_Dragon12p_CodeWarrior/Prog/vectors.c	
Demo program interrupt vectors source file	3686
xcp.c	
XCP 1.0 protocol core source file	3687
xcp.h	
XCP 1.0 protocol core header file	3700

Chapter 5

Module Documentation

5.1 Bootloader

Bootloader.

Files

- file [_template/Boot/blt_conf.h](#)
Bootloader configuration header file.
- file [_template/Boot/hooks.c](#)
Bootloader callback source file.
- file [_template/Boot/led.c](#)
LED driver source file.
- file [_template/Boot/led.h](#)
LED driver header file.
- file [_template/Boot/main.c](#)
Bootloader application source file.

5.1.1 Detailed Description

Bootloader.

5.2 Template for demo programs

Template for creating OpenBLT demo programs.

Modules

- [Bootloader](#)
Bootloader.
- [User Program](#)
User Program.

5.2.1 Detailed Description

Template for creating OpenBLT demo programs.

These template programs for a foundation when creating OpenBLT demo programs. For compilation testing these template demo programs are preconfigured for building with GCC and a Makefile. Other already existing demo programs can serve as a reference when creating OpenBLT demo programs based on this template. The demo bootloader and demo user program templates contain instructions in source code comments of what functionality needs to be implemented where. Search for "TODO ##Boot" and "TODO ##Prog" to find these comments in the template source files.

5.3 User Program

User Program.

Files

- file [Demo/_template/Prog/boot.c](#)
Demo program bootloader interface source file.
- file [Demo/_template/Prog/boot.h](#)
Demo program bootloader interface header file.
- file [_template/Prog/header.h](#)
Generic header file.
- file [_template/Prog/led.c](#)
LED driver source file.
- file [_template/Prog/led.h](#)
LED driver header file.
- file [_template/Prog/main.c](#)
Demo program application source file.
- file [Demo/_template/Prog/timer.c](#)
Timer driver source file.
- file [Demo/_template/Prog/timer.h](#)
Timer driver header file.

5.3.1 Detailed Description

User Program.

The intention of the demo user program is two-fold. (1) To test the bootloader, you need some sort of firmware to see if you can perform a firmware update with the bootloader. This program can be used for this purpose. (2) To make firmware programmable by the bootloader, a few adjustments to the firmware are required. The demo user program serves as an example for how these adjustments can be implemented. This demo user program is a template that can be used as a starting point for creating your own demo user program.

5.4 Bootloader

Bootloader.

Bootloader.

5.5 Demo for S32K118EVB/GCC

Preconfigured programs for the NXP S32K118EVB board and the S32 Design Studio development environment, which is based on the ARM GCC toolchain.

Modules

- [Bootloader](#)
Bootloader.
- [User Program](#)
User Program.

5.5.1 Detailed Description

Preconfigured programs for the NXP S32K118EVB board and the S32 Design Studio development environment, which is based on the ARM GCC toolchain.

5.6 User Program

User Program.

User Program.

The intention of the demo user program is two-fold. (1) To test the bootloader, you need some sort of firmware to see if you can perform a firmware update with the bootloader. This program can be used for this purpose. (2) To make firmware programmable by the bootloader, a few adjustments to the firmware are required. The demo user program serves as an example for how these adjustments can be implemented. This demo user program is a template that can be used as a starting point for creating your own demo user program.

5.7 Bootloader

Bootloader.

Bootloader.

5.8 Demo for S32K118EVB/IAR

Preconfigured programs for the NXP S32K118EVB board and the IAR Embedded Workbench for ARM.

Modules

- [Bootloader](#)
Bootloader.
- [User Program](#)
User Program.

5.8.1 Detailed Description

Preconfigured programs for the NXP S32K118EVB board and the IAR Embedded Workbench for ARM.

5.9 User Program

User Program.

User Program.

The intention of the demo user program is two-fold. (1) To test the bootloader, you need some sort of firmware to see if you can perform a firmware update with the bootloader. This program can be used for this purpose. (2) To make firmware programmable by the bootloader, a few adjustments to the firmware are required. The demo user program serves as an example for how these adjustments can be implemented. This demo user program is a template that can be used as a starting point for creating your own demo user program.

5.10 Bootloader

Bootloader.

Files

- file [ARMCM0_STM32F0_Discovery_STM32F051_CubeIDE/Boot/App/app.c](#)
Bootloader application source file.
- file [ARMCM0_STM32F0_Discovery_STM32F051_CubeIDE/Boot/App/app.h](#)
Bootloader application header file.
- file [ARMCM0_STM32F0_Discovery_STM32F051_CubeIDE/Boot/App/blt_conf.h](#)
Bootloader configuration header file.
- file [ARMCM0_STM32F0_Discovery_STM32F051_CubeIDE/Boot/App/flash_layout.c](#)
Custom flash layout table source file.
- file [ARMCM0_STM32F0_Discovery_STM32F051_CubeIDE/Boot/App/hooks.c](#)
Bootloader callback source file.
- file [ARMCM0_STM32F0_Discovery_STM32F051_CubeIDE/Boot/App/led.c](#)
LED driver source file.
- file [ARMCM0_STM32F0_Discovery_STM32F051_CubeIDE/Boot/App/led.h](#)
LED driver header file.

5.10.1 Detailed Description

Bootloader.

5.11 Demo for STM32F0-Discovery/STM32CubeIDE

Preconfigured programs for the STM32F0-Discovery board and the ST STM32CubeIDE toolchain.

Modules

- [Bootloader](#)
Bootloader.
- [User Program](#)
User Program.

5.11.1 Detailed Description

Preconfigured programs for the STM32F0-Discovery board and the ST STM32CubeIDE toolchain.

Refer to <http://feaser.com/openblt/doku.php?id=manual:demos>

For detailed getting started instructions, refer to: https://www.feaser.com/openblt/doku.php?id=manual:demos:stm32f0_discovery_cubeide.

5.12 User Program

User Program.

Files

- file [ARMCM0_STM32F0_Discovery_STM32F051_CubeIDE/Prog/App/app.c](#)
User program application source file.
- file [ARMCM0_STM32F0_Discovery_STM32F051_CubeIDE/Prog/App/app.h](#)
User program application header file.
- file [Demo/ARMCM0_STM32F0_Discovery_STM32F051_CubeIDE/Prog/App/boot.c](#)
Demo program bootloader interface source file.
- file [Demo/ARMCM0_STM32F0_Discovery_STM32F051_CubeIDE/Prog/App/boot.h](#)
Demo program bootloader interface header file.
- file [ARMCM0_STM32F0_Discovery_STM32F051_CubeIDE/Prog/App/header.h](#)
Generic header file.
- file [ARMCM0_STM32F0_Discovery_STM32F051_CubeIDE/Prog/App/led.c](#)
LED driver source file.
- file [ARMCM0_STM32F0_Discovery_STM32F051_CubeIDE/Prog/App/led.h](#)
LED driver header file.
- file [Demo/ARMCM0_STM32F0_Discovery_STM32F051_CubeIDE/Prog/App/timer.c](#)
Timer driver source file.
- file [Demo/ARMCM0_STM32F0_Discovery_STM32F051_CubeIDE/Prog/App/timer.h](#)
Timer driver header file.

5.12.1 Detailed Description

User Program.

The intention of the demo user program is two-fold. (1) To test the bootloader, you need some sort of firmware to see if you can perform a firmware update with the bootloader. This program can be used for this purpose. (2) To make firmware programmable by the bootloader, a few adjustments to the firmware are required. The demo user program serves as an example for how these adjustments can be implemented. Additional details on this subject can be found in the port specifics documentation, which is available at: https://www.feaser.com/openblt/doku.php?id=manual:ports:armcm0_stm32.

5.13 Bootloader

Bootloader.

Files

- file [ARMCM0_STM32F0_Discovery_STM32F051_GCC/Boot/blt_conf.h](#)
Bootloader configuration header file.
- file [ARMCM0_STM32F0_Discovery_STM32F051_GCC/Boot/flash_layout.c](#)
Custom flash layout table source file.
- file [ARMCM0_STM32F0_Discovery_STM32F051_GCC/Boot/hooks.c](#)
Bootloader callback source file.
- file [ARMCM0_STM32F0_Discovery_STM32F051_GCC/Boot/led.c](#)
LED driver source file.
- file [ARMCM0_STM32F0_Discovery_STM32F051_GCC/Boot/led.h](#)
LED driver header file.
- file [ARMCM0_STM32F0_Discovery_STM32F051_GCC/Boot/main.c](#)
Bootloader application source file.

5.13.1 Detailed Description

Bootloader.

5.14 Demo for STM32F0-Discovery/GCC

Preconfigured programs for the STM32F0-Discovery board and the GCC compiler.

Modules

- [Bootloader](#)
Bootloader.
- [User Program](#)
User Program.

5.14.1 Detailed Description

Preconfigured programs for the STM32F0-Discovery board and the GCC compiler.

Refer to <http://feaser.com/openblt/doku.php?id=manual:demos>

For detailed getting started instructions, refer to: https://www.feaser.com/openblt/doku.php?id=manual:demos:stm32f0_discovery_gcc.

5.15 User Program

User Program.

Files

- file [Demo/ARMCM0_STM32F0_Discovery_STM32F051_GCC/Prog/boot.c](#)
Demo program bootloader interface source file.
- file [Demo/ARMCM0_STM32F0_Discovery_STM32F051_GCC/Prog/boot.h](#)
Demo program bootloader interface header file.
- file [ARMCM0_STM32F0_Discovery_STM32F051_GCC/Prog/header.h](#)
Generic header file.
- file [ARMCM0_STM32F0_Discovery_STM32F051_GCC/Prog/led.c](#)
LED driver source file.
- file [ARMCM0_STM32F0_Discovery_STM32F051_GCC/Prog/led.h](#)
LED driver header file.
- file [ARMCM0_STM32F0_Discovery_STM32F051_GCC/Prog/main.c](#)
Demo program application source file.
- file [Demo/ARMCM0_STM32F0_Discovery_STM32F051_GCC/Prog/timer.c](#)
Timer driver source file.
- file [Demo/ARMCM0_STM32F0_Discovery_STM32F051_GCC/Prog/timer.h](#)
Timer driver header file.

5.15.1 Detailed Description

User Program.

The intention of the demo user program is two-fold. (1) To test the bootloader, you need some sort of firmware to see if you can perform a firmware update with the bootloader. This program can be used for this purpose. (2) To make firmware programmable by the bootloader, a few adjustments to the firmware are required. The demo user program serves as an example for how these adjustments can be implemented. Additional details on this subject can be found in the port specifics documentation, which is available at: https://www.feaser.com/openblt/doku.php?id=manual:ports:armcm0_stm32.

5.16 Bootloader

Bootloader.

Files

- file [ARMCM0_STM32F0_Discovery_STM32F051_IAR/Boot/blt_conf.h](#)
Bootloader configuration header file.
- file [ARMCM0_STM32F0_Discovery_STM32F051_IAR/Boot/flash_layout.c](#)
Custom flash layout table source file.
- file [ARMCM0_STM32F0_Discovery_STM32F051_IAR/Boot/hooks.c](#)
Bootloader callback source file.
- file [ARMCM0_STM32F0_Discovery_STM32F051_IAR/Boot/led.c](#)
LED driver source file.
- file [ARMCM0_STM32F0_Discovery_STM32F051_IAR/Boot/led.h](#)
LED driver header file.
- file [ARMCM0_STM32F0_Discovery_STM32F051_IAR/Boot/main.c](#)
Bootloader application source file.

5.16.1 Detailed Description

Bootloader.

5.17 Demo for STM32F0-Discovery/IAR

Preconfigured programs for the STM32F0-Discovery board and the IAR Embedded Workbench IDE.

Modules

- [Bootloader](#)
Bootloader.
- [User Program](#)
User Program.

5.17.1 Detailed Description

Preconfigured programs for the STM32F0-Discovery board and the IAR Embedded Workbench IDE.

For detailed getting started instructions, refer to: https://www.feaser.com/openblt/doku.php?id=manual:demos:stm32f0_discovery_iar.

5.18 User Program

User Program.

Files

- file [Demo/ARMCM0_STM32F0_Discovery_STM32F051_IAR/Prog/boot.c](#)
Demo program bootloader interface source file.
- file [Demo/ARMCM0_STM32F0_Discovery_STM32F051_IAR/Prog/boot.h](#)
Demo program bootloader interface header file.
- file [ARMCM0_STM32F0_Discovery_STM32F051_IAR/Prog/header.h](#)
Generic header file.
- file [ARMCM0_STM32F0_Discovery_STM32F051_IAR/Prog/led.c](#)
LED driver source file.
- file [ARMCM0_STM32F0_Discovery_STM32F051_IAR/Prog/led.h](#)
LED driver header file.
- file [ARMCM0_STM32F0_Discovery_STM32F051_IAR/Prog/main.c](#)
Demo program application source file.
- file [Demo/ARMCM0_STM32F0_Discovery_STM32F051_IAR/Prog/timer.c](#)
Timer driver source file.
- file [Demo/ARMCM0_STM32F0_Discovery_STM32F051_IAR/Prog/timer.h](#)
Timer driver header file.

5.18.1 Detailed Description

User Program.

The intention of the demo user program is two-fold. (1) To test the bootloader, you need some sort of firmware to see if you can perform a firmware update with the bootloader. This program can be used for this purpose. (2) To make firmware programmable by the bootloader, a few adjustments to the firmware are required. The demo user program serves as an example for how these adjustments can be implemented. Additional details on this subject can be found in the port specifics documentation, which is available at: https://www.feaser.com/openblt/doku.php?id=manual:ports:armcm0_stm32.

5.19 Bootloader

Bootloader.

Files

- file [ARMCM0_STM32F0_Discovery_STM32F051_Keil/Boot/blt_conf.h](#)
Bootloader configuration header file.
- file [ARMCM0_STM32F0_Discovery_STM32F051_Keil/Boot/flash_layout.c](#)
Custom flash layout table source file.
- file [ARMCM0_STM32F0_Discovery_STM32F051_Keil/Boot/hooks.c](#)
Bootloader callback source file.
- file [ARMCM0_STM32F0_Discovery_STM32F051_Keil/Boot/led.c](#)
LED driver source file.
- file [ARMCM0_STM32F0_Discovery_STM32F051_Keil/Boot/led.h](#)
LED driver header file.
- file [ARMCM0_STM32F0_Discovery_STM32F051_Keil/Boot/main.c](#)
Bootloader application source file.

5.19.1 Detailed Description

Bootloader.

5.20 Demo for STM32F0-Discovery/Keil

Preconfigured programs for the STM32F0-Discovery board and the Keil MDK toolchain.

Modules

- [Bootloader](#)
Bootloader.
- [User Program](#)
User Program.

5.20.1 Detailed Description

Preconfigured programs for the STM32F0-Discovery board and the Keil MDK toolchain.

Refer to <http://feaser.com/openblt/doku.php?id=manual:demos>

For detailed getting started instructions, refer to: https://www.feaser.com/openblt/doku.php?id=manual:demos:stm32f0_discovery_keil.

5.21 User Program

User Program.

Files

- file [Demo/ARMCM0_STM32F0_Discovery_STM32F051_Keil/Prog/boot.c](#)
Demo program bootloader interface source file.
- file [Demo/ARMCM0_STM32F0_Discovery_STM32F051_Keil/Prog/boot.h](#)
Demo program bootloader interface header file.
- file [ARMCM0_STM32F0_Discovery_STM32F051_Keil/Prog/header.h](#)
Generic header file.
- file [ARMCM0_STM32F0_Discovery_STM32F051_Keil/Prog/led.c](#)
LED driver source file.
- file [ARMCM0_STM32F0_Discovery_STM32F051_Keil/Prog/led.h](#)
LED driver header file.
- file [ARMCM0_STM32F0_Discovery_STM32F051_Keil/Prog/main.c](#)
Demo program application source file.
- file [Demo/ARMCM0_STM32F0_Discovery_STM32F051_Keil/Prog/timer.c](#)
Timer driver source file.
- file [Demo/ARMCM0_STM32F0_Discovery_STM32F051_Keil/Prog/timer.h](#)
Timer driver header file.

5.21.1 Detailed Description

User Program.

The intention of the demo user program is two-fold. (1) To test the bootloader, you need some sort of firmware to see if you can perform a firmware update with the bootloader. This program can be used for this purpose. (2) To make firmware programmable by the bootloader, a few adjustments to the firmware are required. The demo user program serves as an example for how these adjustments can be implemented. Additional details on this subject can be found in the port specifics documentation, which is available at: https://www.feaser.com/openblt/doku.php?id=manual:ports:armcm0_stm32.

5.22 Bootloader

Bootloader.

Files

- file [ARMCM0_STM32F0_Nucleo_F091RC_CubeIDE/Boot/App/app.c](#)
Bootloader application source file.
- file [ARMCM0_STM32F0_Nucleo_F091RC_CubeIDE/Boot/App/app.h](#)
Bootloader application header file.
- file [ARMCM0_STM32F0_Nucleo_F091RC_CubeIDE/Boot/App/blt_conf.h](#)
Bootloader configuration header file.
- file [ARMCM0_STM32F0_Nucleo_F091RC_CubeIDE/Boot/App/hooks.c](#)
Bootloader callback source file.
- file [ARMCM0_STM32F0_Nucleo_F091RC_CubeIDE/Boot/App/led.c](#)
LED driver source file.
- file [ARMCM0_STM32F0_Nucleo_F091RC_CubeIDE/Boot/App/led.h](#)
LED driver header file.

5.22.1 Detailed Description

Bootloader.

5.23 Demo for Nucleo-F091RC/STM32CubeIDE

Preconfigured programs for the Nucleo-F091RC and the ST STM32CubeIDE toolchain.

Modules

- [Bootloader](#)
Bootloader.
- [User Program](#)
User Program.

5.23.1 Detailed Description

Preconfigured programs for the Nucleo-F091RC and the ST STM32CubeIDE toolchain.

Refer to <http://feaser.com/openblt/doku.php?id=manual:demos>

For detailed getting started instructions, refer to: https://www.feaser.com/openblt/doku.php?id=manual:demos:nucleo_f091rc_cubeide.

5.24 User Program

User Program.

Files

- file [ARMCM0_STM32F0_Nucleo_F091RC_CubeIDE/Prog/App/app.c](#)
User program application source file.
- file [ARMCM0_STM32F0_Nucleo_F091RC_CubeIDE/Prog/App/app.h](#)
User program application header file.
- file [Demo/ARMCM0_STM32F0_Nucleo_F091RC_CubeIDE/Prog/App/boot.c](#)
Demo program bootloader interface source file.
- file [Demo/ARMCM0_STM32F0_Nucleo_F091RC_CubeIDE/Prog/App/boot.h](#)
Demo program bootloader interface header file.
- file [ARMCM0_STM32F0_Nucleo_F091RC_CubeIDE/Prog/App/header.h](#)
Generic header file.
- file [ARMCM0_STM32F0_Nucleo_F091RC_CubeIDE/Prog/App/led.c](#)
LED driver source file.
- file [ARMCM0_STM32F0_Nucleo_F091RC_CubeIDE/Prog/App/led.h](#)
LED driver header file.
- file [Demo/ARMCM0_STM32F0_Nucleo_F091RC_CubeIDE/Prog/App/timer.c](#)
Timer driver source file.
- file [Demo/ARMCM0_STM32F0_Nucleo_F091RC_CubeIDE/Prog/App/timer.h](#)
Timer driver header file.

5.24.1 Detailed Description

User Program.

The intention of the demo user program is two-fold. (1) To test the bootloader, you need some sort of firmware to see if you can perform a firmware update with the bootloader. This program can be used for this purpose. (2) To make firmware programmable by the bootloader, a few adjustments to the firmware are required. The demo user program serves as an example for how these adjustments can be implemented. Additional details on this subject can be found in the port specifics documentation, which is available at: https://www.feaser.com/openblt/doku.php?id=manual:ports:armcm0_stm32.

5.25 Bootloader

Bootloader.

Files

- file [ARMCM0_STM32F0_Nucleo_F091RC_GCC/Boot/blt_conf.h](#)
Bootloader configuration header file.
- file [ARMCM0_STM32F0_Nucleo_F091RC_GCC/Boot/hooks.c](#)
Bootloader callback source file.
- file [ARMCM0_STM32F0_Nucleo_F091RC_GCC/Boot/led.c](#)
LED driver source file.
- file [ARMCM0_STM32F0_Nucleo_F091RC_GCC/Boot/led.h](#)
LED driver header file.
- file [ARMCM0_STM32F0_Nucleo_F091RC_GCC/Boot/main.c](#)
Bootloader application source file.

5.25.1 Detailed Description

Bootloader.

5.26 Demo for Nucleo-F091RC/GCC

Preconfigured programs for the Nucleo-F091RC and the GNU ARM GCC compiler.

Modules

- [Bootloader](#)
Bootloader.
- [User Program](#)
User Program.

5.26.1 Detailed Description

Preconfigured programs for the Nucleo-F091RC and the GNU ARM GCC compiler.

Refer to <http://feaser.com/openblt/doku.php?id=manual:demos>

For detailed getting started instructions, refer to: https://www.feaser.com/openblt/doku.php?id=manual:demos:nucleo_f091rc_gcc.

5.27 User Program

User Program.

Files

- file [Demo/ARMCM0_STM32F0_Nucleo_F091RC_GCC/Prog/boot.c](#)
Demo program bootloader interface source file.
- file [Demo/ARMCM0_STM32F0_Nucleo_F091RC_GCC/Prog/boot.h](#)
Demo program bootloader interface header file.
- file [ARMCM0_STM32F0_Nucleo_F091RC_GCC/Prog/header.h](#)
Generic header file.
- file [ARMCM0_STM32F0_Nucleo_F091RC_GCC/Prog/led.c](#)
LED driver source file.
- file [ARMCM0_STM32F0_Nucleo_F091RC_GCC/Prog/led.h](#)
LED driver header file.
- file [ARMCM0_STM32F0_Nucleo_F091RC_GCC/Prog/main.c](#)
Demo program application source file.
- file [Demo/ARMCM0_STM32F0_Nucleo_F091RC_GCC/Prog/timer.c](#)
Timer driver source file.
- file [Demo/ARMCM0_STM32F0_Nucleo_F091RC_GCC/Prog/timer.h](#)
Timer driver header file.

5.27.1 Detailed Description

User Program.

The intention of the demo user program is two-fold. (1) To test the bootloader, you need some sort of firmware to see if you can perform a firmware update with the bootloader. This program can be used for this purpose. (2) To make firmware programmable by the bootloader, a few adjustments to the firmware are required. The demo user program serves as an example for how these adjustments can be implemented. Additional details on this subject can be found in the port specifics documentation, which is available at: https://www.feaser.com/openblt/doku.php?id=manual:ports:armcm0_stm32.

5.28 Bootloader

Bootloader.

Files

- file [ARMCM0_STM32F0_Nucleo_F091RC_IAR/Boot/blt_conf.h](#)
Bootloader configuration header file.
- file [ARMCM0_STM32F0_Nucleo_F091RC_IAR/Boot/hooks.c](#)
Bootloader callback source file.
- file [ARMCM0_STM32F0_Nucleo_F091RC_IAR/Boot/led.c](#)
LED driver source file.
- file [ARMCM0_STM32F0_Nucleo_F091RC_IAR/Boot/led.h](#)
LED driver header file.
- file [ARMCM0_STM32F0_Nucleo_F091RC_IAR/Boot/main.c](#)
Bootloader application source file.

5.28.1 Detailed Description

Bootloader.

5.29 Demo for Nucleo-F091RC/IAR

Preconfigured programs for the Nucleo-F091RC board and the IAR Embedded Workbench IDE.

Modules

- [Bootloader](#)
Bootloader.
- [User Program](#)
User Program.

5.29.1 Detailed Description

Preconfigured programs for the Nucleo-F091RC board and the IAR Embedded Workbench IDE.

For detailed getting started instructions, refer to: https://www.feaser.com/openblt/doku.php?id=manual:demos:nucleo_f091rc_iar.

5.30 User Program

User Program.

Files

- file [Demo/ARMCM0_STM32F0_Nucleo_F091RC_IAR/Prog/boot.c](#)
Demo program bootloader interface source file.
- file [Demo/ARMCM0_STM32F0_Nucleo_F091RC_IAR/Prog/boot.h](#)
Demo program bootloader interface header file.
- file [ARMCM0_STM32F0_Nucleo_F091RC_IAR/Prog/header.h](#)
Generic header file.
- file [ARMCM0_STM32F0_Nucleo_F091RC_IAR/Prog/led.c](#)
LED driver source file.
- file [ARMCM0_STM32F0_Nucleo_F091RC_IAR/Prog/led.h](#)
LED driver header file.
- file [ARMCM0_STM32F0_Nucleo_F091RC_IAR/Prog/main.c](#)
Demo program application source file.
- file [Demo/ARMCM0_STM32F0_Nucleo_F091RC_IAR/Prog/timer.c](#)
Timer driver source file.
- file [Demo/ARMCM0_STM32F0_Nucleo_F091RC_IAR/Prog/timer.h](#)
Timer driver header file.

5.30.1 Detailed Description

User Program.

The intention of the demo user program is two-fold. (1) To test the bootloader, you need some sort of firmware to see if you can perform a firmware update with the bootloader. This program can be used for this purpose. (2) To make firmware programmable by the bootloader, a few adjustments to the firmware are required. The demo user program serves as an example for how these adjustments can be implemented. Additional details on this subject can be found in the port specifics documentation, which is available at: https://www.feaser.com/openblt/doku.php?id=manual:ports:armcm0_stm32.

5.31 Bootloader

Bootloader.

Files

- file [ARMCM0_STM32F0_Nucleo_F091RC_Keil/Boot/blt_conf.h](#)
Bootloader configuration header file.
- file [ARMCM0_STM32F0_Nucleo_F091RC_Keil/Boot/hooks.c](#)
Bootloader callback source file.
- file [ARMCM0_STM32F0_Nucleo_F091RC_Keil/Boot/led.c](#)
LED driver source file.
- file [ARMCM0_STM32F0_Nucleo_F091RC_Keil/Boot/led.h](#)
LED driver header file.
- file [ARMCM0_STM32F0_Nucleo_F091RC_Keil/Boot/main.c](#)
Bootloader application source file.

5.31.1 Detailed Description

Bootloader.

5.32 Demo for Nucleo-F091RC/Keil

Preconfigured programs for the Nucleo-F091RC and the Keil MDK toolchain.

Modules

- [Bootloader](#)
Bootloader.
- [User Program](#)
User Program.

5.32.1 Detailed Description

Preconfigured programs for the Nucleo-F091RC and the Keil MDK toolchain.

For detailed getting started instructions, refer to: https://www.feaser.com/openblt/doku.php?id=manual:demos:demos:nucleo_f091rc_keil.

5.33 User Program

User Program.

Files

- file [Demo/ARMCM0_STM32F0_Nucleo_F091RC_Keil/Prog/boot.c](#)
Demo program bootloader interface source file.
- file [Demo/ARMCM0_STM32F0_Nucleo_F091RC_Keil/Prog/boot.h](#)
Demo program bootloader interface header file.
- file [ARMCM0_STM32F0_Nucleo_F091RC_Keil/Prog/header.h](#)
Generic header file.
- file [ARMCM0_STM32F0_Nucleo_F091RC_Keil/Prog/led.c](#)
LED driver source file.
- file [ARMCM0_STM32F0_Nucleo_F091RC_Keil/Prog/led.h](#)
LED driver header file.
- file [ARMCM0_STM32F0_Nucleo_F091RC_Keil/Prog/main.c](#)
Demo program application source file.
- file [Demo/ARMCM0_STM32F0_Nucleo_F091RC_Keil/Prog/timer.c](#)
Timer driver source file.
- file [Demo/ARMCM0_STM32F0_Nucleo_F091RC_Keil/Prog/timer.h](#)
Timer driver header file.

5.33.1 Detailed Description

User Program.

The intention of the demo user program is two-fold. (1) To test the bootloader, you need some sort of firmware to see if you can perform a firmware update with the bootloader. This program can be used for this purpose. (2) To make firmware programmable by the bootloader, a few adjustments to the firmware are required. The demo user program serves as an example for how these adjustments can be implemented. Additional details on this subject can be found in the port specifics documentation, which is available at: https://www.feaser.com/openblt/doku.php?id=manual:ports:armcm0_stm32.

5.34 Bootloader

Bootloader.

Files

- file [ARMCM0_STM32G0_Nucleo_G071RB_CubeIDE/Boot/App/app.c](#)
Bootloader application source file.
- file [ARMCM0_STM32G0_Nucleo_G071RB_CubeIDE/Boot/App/app.h](#)
Bootloader application header file.
- file [ARMCM0_STM32G0_Nucleo_G071RB_CubeIDE/Boot/App/blt_conf.h](#)
Bootloader configuration header file.
- file [ARMCM0_STM32G0_Nucleo_G071RB_CubeIDE/Boot/App/hooks.c](#)
Bootloader callback source file.
- file [ARMCM0_STM32G0_Nucleo_G071RB_CubeIDE/Boot/App/led.c](#)
LED driver source file.
- file [ARMCM0_STM32G0_Nucleo_G071RB_CubeIDE/Boot/App/led.h](#)
LED driver header file.

5.34.1 Detailed Description

Bootloader.

5.35 Demo for Nucleo-G071RB/STM32CubeIDE

Preconfigured programs for the Nucleo-G071RB and the ST STM32CubeIDE toolchain.

Modules

- [Bootloader](#)
Bootloader.
- [User Program](#)
User Program.

5.35.1 Detailed Description

Preconfigured programs for the Nucleo-G071RB and the ST STM32CubeIDE toolchain.

For detailed getting started instructions, refer to: https://www.feaser.com/openblt/doku.php?id=manual:demos:nucleo_g071rb_cubeide.

5.36 User Program

User Program.

Files

- file [ARMCM0_STM32G0_Nucleo_G071RB_CubeIDE/Prog/App/app.c](#)
User program application source file.
- file [ARMCM0_STM32G0_Nucleo_G071RB_CubeIDE/Prog/App/app.h](#)
User program application header file.
- file [Demo/ARMCM0_STM32G0_Nucleo_G071RB_CubeIDE/Prog/App/boot.c](#)
Demo program bootloader interface source file.
- file [Demo/ARMCM0_STM32G0_Nucleo_G071RB_CubeIDE/Prog/App/boot.h](#)
Demo program bootloader interface header file.
- file [ARMCM0_STM32G0_Nucleo_G071RB_CubeIDE/Prog/App/header.h](#)
Generic header file.
- file [ARMCM0_STM32G0_Nucleo_G071RB_CubeIDE/Prog/App/led.c](#)
LED driver source file.
- file [ARMCM0_STM32G0_Nucleo_G071RB_CubeIDE/Prog/App/led.h](#)
LED driver header file.
- file [Demo/ARMCM0_STM32G0_Nucleo_G071RB_CubeIDE/Prog/App/timer.c](#)
Timer driver source file.
- file [Demo/ARMCM0_STM32G0_Nucleo_G071RB_CubeIDE/Prog/App/timer.h](#)
Timer driver header file.

5.36.1 Detailed Description

User Program.

The intention of the demo user program is two-fold. (1) To test the bootloader, you need some sort of firmware to see if you can perform a firmware update with the bootloader. This program can be used for this purpose. (2) To make firmware programmable by the bootloader, a few adjustments to the firmware are required. The demo user program serves as an example for how these adjustments can be implemented. This demo user program is a template that can be used as a starting point for creating your own demo user program.

5.37 Bootloader

Bootloader.

Files

- file [ARMCM0_STM32G0_Nucleo_G071RB_GCC/Boot/blt_conf.h](#)
Bootloader configuration header file.
- file [ARMCM0_STM32G0_Nucleo_G071RB_GCC/Boot/hooks.c](#)
Bootloader callback source file.
- file [ARMCM0_STM32G0_Nucleo_G071RB_GCC/Boot/led.c](#)
LED driver source file.
- file [ARMCM0_STM32G0_Nucleo_G071RB_GCC/Boot/led.h](#)
LED driver header file.
- file [ARMCM0_STM32G0_Nucleo_G071RB_GCC/Boot/main.c](#)
Bootloader application source file.

5.37.1 Detailed Description

Bootloader.

5.38 Demo for Nucleo-G071RB/GCC

Preconfigured programs for the Nucleo-G071RB and the GNU ARM GCC compiler.

Modules

- [Bootloader](#)
Bootloader.
- [User Program](#)
User Program.

5.38.1 Detailed Description

Preconfigured programs for the Nucleo-G071RB and the GNU ARM GCC compiler.

For detailed getting started instructions, refer to: https://www.feaser.com/openblt/doku.php?id=manual:demos:nucleo_g071rb_gcc.

5.39 User Program

User Program.

Files

- file [Demo/ARMCM0_STM32G0_Nucleo_G071RB_GCC/Prog/boot.c](#)
Demo program bootloader interface source file.
- file [Demo/ARMCM0_STM32G0_Nucleo_G071RB_GCC/Prog/boot.h](#)
Demo program bootloader interface header file.
- file [ARMCM0_STM32G0_Nucleo_G071RB_GCC/Prog/header.h](#)
Generic header file.
- file [ARMCM0_STM32G0_Nucleo_G071RB_GCC/Prog/led.c](#)
LED driver source file.
- file [ARMCM0_STM32G0_Nucleo_G071RB_GCC/Prog/led.h](#)
LED driver header file.
- file [ARMCM0_STM32G0_Nucleo_G071RB_GCC/Prog/main.c](#)
Demo program application source file.
- file [Demo/ARMCM0_STM32G0_Nucleo_G071RB_GCC/Prog/timer.c](#)
Timer driver source file.
- file [Demo/ARMCM0_STM32G0_Nucleo_G071RB_GCC/Prog/timer.h](#)
Timer driver header file.

5.39.1 Detailed Description

User Program.

The intention of the demo user program is two-fold. (1) To test the bootloader, you need some sort of firmware to see if you can perform a firmware update with the bootloader. This program can be used for this purpose. (2) To make firmware programmable by the bootloader, a few adjustments to the firmware are required. The demo user program serves as an example for how these adjustments can be implemented. This demo user program is a template that can be used as a starting point for creating your own demo user program.

5.40 Bootloader

Bootloader.

Files

- file [ARMCM0_STM32G0_Nucleo_G071RB_IAR/Boot/blt_conf.h](#)
Bootloader configuration header file.
- file [ARMCM0_STM32G0_Nucleo_G071RB_IAR/Boot/hooks.c](#)
Bootloader callback source file.
- file [ARMCM0_STM32G0_Nucleo_G071RB_IAR/Boot/led.c](#)
LED driver source file.
- file [ARMCM0_STM32G0_Nucleo_G071RB_IAR/Boot/led.h](#)
LED driver header file.
- file [ARMCM0_STM32G0_Nucleo_G071RB_IAR/Boot/main.c](#)
Bootloader application source file.

5.40.1 Detailed Description

Bootloader.

5.41 Demo for Nucleo-G071RB/IAR

Preconfigured programs for the Nucleo-G071RB and the IAR Embedded Workbench IDE.

Modules

- [Bootloader](#)
Bootloader.
- [User Program](#)
User Program.

5.41.1 Detailed Description

Preconfigured programs for the Nucleo-G071RB and the IAR Embedded Workbench IDE.

For detailed getting started instructions, refer to: https://www.feaser.com/openblt/doku.php?id=manual:demos:nucleo_g071rb_iar.

5.42 User Program

User Program.

Files

- file [Demo/ARMCM0_STM32G0_Nucleo_G071RB_IAR/Prog/boot.c](#)
Demo program bootloader interface source file.
- file [Demo/ARMCM0_STM32G0_Nucleo_G071RB_IAR/Prog/boot.h](#)
Demo program bootloader interface header file.
- file [ARMCM0_STM32G0_Nucleo_G071RB_IAR/Prog/header.h](#)
Generic header file.
- file [ARMCM0_STM32G0_Nucleo_G071RB_IAR/Prog/led.c](#)
LED driver source file.
- file [ARMCM0_STM32G0_Nucleo_G071RB_IAR/Prog/led.h](#)
LED driver header file.
- file [ARMCM0_STM32G0_Nucleo_G071RB_IAR/Prog/main.c](#)
Demo program application source file.
- file [Demo/ARMCM0_STM32G0_Nucleo_G071RB_IAR/Prog/timer.c](#)
Timer driver source file.
- file [Demo/ARMCM0_STM32G0_Nucleo_G071RB_IAR/Prog/timer.h](#)
Timer driver header file.

5.42.1 Detailed Description

User Program.

The intention of the demo user program is two-fold. (1) To test the bootloader, you need some sort of firmware to see if you can perform a firmware update with the bootloader. This program can be used for this purpose. (2) To make firmware programmable by the bootloader, a few adjustments to the firmware are required. The demo user program serves as an example for how these adjustments can be implemented. This demo user program is a template that can be used as a starting point for creating your own demo user program.

5.43 Bootloader

Bootloader.

Files

- file [ARMCM0_STM32G0_Nucleo_G071RB_Keil/Boot/blt_conf.h](#)
Bootloader configuration header file.
- file [ARMCM0_STM32G0_Nucleo_G071RB_Keil/Boot/hooks.c](#)
Bootloader callback source file.
- file [ARMCM0_STM32G0_Nucleo_G071RB_Keil/Boot/led.c](#)
LED driver source file.
- file [ARMCM0_STM32G0_Nucleo_G071RB_Keil/Boot/led.h](#)
LED driver header file.
- file [ARMCM0_STM32G0_Nucleo_G071RB_Keil/Boot/main.c](#)
Bootloader application source file.

5.43.1 Detailed Description

Bootloader.

5.44 Demo for Nucleo-G071RB/Keil

Preconfigured programs for the Nucleo-G071RB and the Keil MDK toolchain.

Modules

- [Bootloader](#)
Bootloader.
- [User Program](#)
User Program.

5.44.1 Detailed Description

Preconfigured programs for the Nucleo-G071RB and the Keil MDK toolchain.

For detailed getting started instructions, refer to: https://www.feaser.com/openblt/doku.php?id=manual:demos:nucleo_g071rb_keil.

5.45 User Program

User Program.

Files

- file [Demo/ARMCM0_STM32G0_Nucleo_G071RB_Keil/Prog/boot.c](#)
Demo program bootloader interface source file.
- file [Demo/ARMCM0_STM32G0_Nucleo_G071RB_Keil/Prog/boot.h](#)
Demo program bootloader interface header file.
- file [ARMCM0_STM32G0_Nucleo_G071RB_Keil/Prog/header.h](#)
Generic header file.
- file [ARMCM0_STM32G0_Nucleo_G071RB_Keil/Prog/led.c](#)
LED driver source file.
- file [ARMCM0_STM32G0_Nucleo_G071RB_Keil/Prog/led.h](#)
LED driver header file.
- file [ARMCM0_STM32G0_Nucleo_G071RB_Keil/Prog/main.c](#)
Demo program application source file.
- file [Demo/ARMCM0_STM32G0_Nucleo_G071RB_Keil/Prog/timer.c](#)
Timer driver source file.
- file [Demo/ARMCM0_STM32G0_Nucleo_G071RB_Keil/Prog/timer.h](#)
Timer driver header file.

5.45.1 Detailed Description

User Program.

The intention of the demo user program is two-fold. (1) To test the bootloader, you need some sort of firmware to see if you can perform a firmware update with the bootloader. This program can be used for this purpose. (2) To make firmware programmable by the bootloader, a few adjustments to the firmware are required. The demo user program serves as an example for how these adjustments can be implemented. This demo user program is a template that can be used as a starting point for creating your own demo user program.

5.46 Bootloader

Bootloader.

Files

- file [ARMCM0_XMC1_XMC1400_Boot_Kit_GCC/Boot/blt_conf.h](#)
Bootloader configuration header file.
- file [ARMCM0_XMC1_XMC1400_Boot_Kit_GCC/Boot/hooks.c](#)
Bootloader callback source file.
- file [ARMCM0_XMC1_XMC1400_Boot_Kit_GCC/Boot/led.c](#)
LED driver source file.
- file [ARMCM0_XMC1_XMC1400_Boot_Kit_GCC/Boot/led.h](#)
LED driver header file.
- file [ARMCM0_XMC1_XMC1400_Boot_Kit_GCC/Boot/main.c](#)
Bootloader application source file.

5.46.1 Detailed Description

Bootloader.

5.47 Demo for XMC1400 Boot Kit/GCC

Preconfigured programs for the Infineon XMC1400 Boot Kit board and the Dave 4 development environment, which is based on the ARM GCC toolchain.

Modules

- [Bootloader](#)
Bootloader.
- [User Program](#)
User Program.

5.47.1 Detailed Description

Preconfigured programs for the Infineon XMC1400 Boot Kit board and the Dave 4 development environment, which is based on the ARM GCC toolchain.

For detailed getting started instructions, refer to: https://www.feaser.com/openblt/doku.php?id=manual:demos:infineon_xmc1400_boot_kit_gcc.

5.48 User Program

User Program.

Files

- file [Demo/ARMCM0_XMC1_XMC1400_Boot_Kit_GCC/Prog/boot.c](#)
Demo program bootloader interface source file.
- file [Demo/ARMCM0_XMC1_XMC1400_Boot_Kit_GCC/Prog/boot.h](#)
Demo program bootloader interface header file.
- file [ARMCM0_XMC1_XMC1400_Boot_Kit_GCC/Prog/header.h](#)
Generic header file.
- file [ARMCM0_XMC1_XMC1400_Boot_Kit_GCC/Prog/led.c](#)
LED driver source file.
- file [ARMCM0_XMC1_XMC1400_Boot_Kit_GCC/Prog/led.h](#)
LED driver header file.
- file [ARMCM0_XMC1_XMC1400_Boot_Kit_GCC/Prog/main.c](#)
Demo program application source file.
- file [Demo/ARMCM0_XMC1_XMC1400_Boot_Kit_GCC/Prog/timer.c](#)
Timer driver source file.
- file [Demo/ARMCM0_XMC1_XMC1400_Boot_Kit_GCC/Prog/timer.h](#)
Timer driver header file.

5.48.1 Detailed Description

User Program.

The intention of the demo user program is two-fold. (1) To test the bootloader, you need some sort of firmware to see if you can perform a firmware update with the bootloader. This program can be used for this purpose. (2) To make firmware programmable by the bootloader, a few adjustments to the firmware are required. The demo user program serves as an example for how these adjustments can be implemented. Additional details on this subject can be found in the port specifics documentation, which is available at: https://www.feaser.com/openblt/doku.php?id=manual:ports:armcm0_xmc1.

5.49 Bootloader

Bootloader.

Files

- file [ARMCM0_XMC1_XMC1400_Boot_Kit_IAR/Boot/blt_conf.h](#)
Bootloader configuration header file.
- file [ARMCM0_XMC1_XMC1400_Boot_Kit_IAR/Boot/hooks.c](#)
Bootloader callback source file.
- file [ARMCM0_XMC1_XMC1400_Boot_Kit_IAR/Boot/led.c](#)
LED driver source file.
- file [ARMCM0_XMC1_XMC1400_Boot_Kit_IAR/Boot/led.h](#)
LED driver header file.
- file [ARMCM0_XMC1_XMC1400_Boot_Kit_IAR/Boot/main.c](#)
Bootloader application source file.

5.49.1 Detailed Description

Bootloader.

5.50 Demo for XMC1400 Boot Kit/IAR

Preconfigured programs for the Infineon XMC1400 Boot Kit board and the IAR compiler.

Modules

- [Bootloader](#)
Bootloader.
- [User Program](#)
User Program.

5.50.1 Detailed Description

Preconfigured programs for the Infineon XMC1400 Boot Kit board and the IAR compiler.

For detailed getting started instructions, refer to: https://www.feaser.com/openblt/doku.php?id=manual:demos:infineon_xmc1400_boot_kit_iar.

5.51 User Program

User Program.

Files

- file [Demo/ARMCM0_XMC1_XMC1400_Boot_Kit_IAR/Prog/boot.c](#)
Demo program bootloader interface source file.
- file [Demo/ARMCM0_XMC1_XMC1400_Boot_Kit_IAR/Prog/boot.h](#)
Demo program bootloader interface header file.
- file [ARMCM0_XMC1_XMC1400_Boot_Kit_IAR/Prog/header.h](#)
Generic header file.
- file [ARMCM0_XMC1_XMC1400_Boot_Kit_IAR/Prog/led.c](#)
LED driver source file.
- file [ARMCM0_XMC1_XMC1400_Boot_Kit_IAR/Prog/led.h](#)
LED driver header file.
- file [ARMCM0_XMC1_XMC1400_Boot_Kit_IAR/Prog/main.c](#)
Demo program application source file.
- file [Demo/ARMCM0_XMC1_XMC1400_Boot_Kit_IAR/Prog/timer.c](#)
Timer driver source file.
- file [Demo/ARMCM0_XMC1_XMC1400_Boot_Kit_IAR/Prog/timer.h](#)
Timer driver header file.

5.51.1 Detailed Description

User Program.

The intention of the demo user program is two-fold. (1) To test the bootloader, you need some sort of firmware to see if you can perform a firmware update with the bootloader. This program can be used for this purpose. (2) To make firmware programmable by the bootloader, a few adjustments to the firmware are required. The demo user program serves as an example for how these adjustments can be implemented. Additional details on this subject can be found in the port specifics documentation, which is available at: https://www.feaser.com/openblt/doku.php?id=manual:ports:armcm0_xmc1.

5.52 Bootloader

Bootloader.

Files

- file [ARMCM33_STM32L5_Nucleo_L552ZE_CubeIDE/Boot/App/app.c](#)
Bootloader application source file.
- file [ARMCM33_STM32L5_Nucleo_L552ZE_CubeIDE/Boot/App/app.h](#)
Bootloader application header file.
- file [ARMCM33_STM32L5_Nucleo_L552ZE_CubeIDE/Boot/App/blt_conf.h](#)
Bootloader configuration header file.
- file [ARMCM33_STM32L5_Nucleo_L552ZE_CubeIDE/Boot/App/hooks.c](#)
Bootloader callback source file.
- file [ARMCM33_STM32L5_Nucleo_L552ZE_CubeIDE/Boot/App/led.c](#)
LED driver source file.
- file [ARMCM33_STM32L5_Nucleo_L552ZE_CubeIDE/Boot/App/led.h](#)
LED driver header file.

5.52.1 Detailed Description

Bootloader.

5.53 Demo for Nucleo-L552ZE/STM32CubeIDE

Preconfigured programs for the Nucleo-L552ZE board and the ST STM32CubeIDE toolchain.

Modules

- [Bootloader](#)
Bootloader.
- [User Program](#)
User Program.

5.53.1 Detailed Description

Preconfigured programs for the Nucleo-L552ZE board and the ST STM32CubeIDE toolchain.

For detailed getting started instructions, refer to: https://www.feaser.com/openblt/doku.php?id=manual:demos:nucleo_l552ze_cubeide.

5.54 User Program

User Program.

Files

- file [ARMCM33_STM32L5_Nucleo_L552ZE_CubeIDE/Prog/App/app.c](#)
User program application source file.
- file [ARMCM33_STM32L5_Nucleo_L552ZE_CubeIDE/Prog/App/app.h](#)
User program application header file.
- file [Demo/ARMCM33_STM32L5_Nucleo_L552ZE_CubeIDE/Prog/App/boot.c](#)
Demo program bootloader interface source file.
- file [Demo/ARMCM33_STM32L5_Nucleo_L552ZE_CubeIDE/Prog/App/boot.h](#)
Demo program bootloader interface header file.
- file [ARMCM33_STM32L5_Nucleo_L552ZE_CubeIDE/Prog/App/header.h](#)
Generic header file.
- file [ARMCM33_STM32L5_Nucleo_L552ZE_CubeIDE/Prog/App/led.c](#)
LED driver source file.
- file [ARMCM33_STM32L5_Nucleo_L552ZE_CubeIDE/Prog/App/led.h](#)
LED driver header file.
- file [Demo/ARMCM33_STM32L5_Nucleo_L552ZE_CubeIDE/Prog/App/timer.c](#)
Timer driver source file.
- file [Demo/ARMCM33_STM32L5_Nucleo_L552ZE_CubeIDE/Prog/App/timer.h](#)
Timer driver header file.

5.54.1 Detailed Description

User Program.

The intention of the demo user program is two-fold. (1) To test the bootloader, you need some sort of firmware to see if you can perform a firmware update with the bootloader. This program can be used for this purpose. (2) To make firmware programmable by the bootloader, a few adjustments to the firmware are required. The demo user program serves as an example for how these adjustments can be implemented. Additional details on this subject can be found in the port specifics documentation, which is available at: <https://www.feaser.com/openblt/doku.php?id=manual:ports:armcm33-stm32l5>.

5.55 Bootloader

Bootloader.

Files

- file [ARMCM33_STM32L5_Nucleo_L552ZE_GCC/Boot/blt_conf.h](#)
Bootloader configuration header file.
- file [ARMCM33_STM32L5_Nucleo_L552ZE_GCC/Boot/hooks.c](#)
Bootloader callback source file.
- file [ARMCM33_STM32L5_Nucleo_L552ZE_GCC/Boot/led.c](#)
LED driver source file.
- file [ARMCM33_STM32L5_Nucleo_L552ZE_GCC/Boot/led.h](#)
LED driver header file.
- file [ARMCM33_STM32L5_Nucleo_L552ZE_GCC/Boot/main.c](#)
Bootloader application source file.

5.55.1 Detailed Description

Bootloader.

5.56 Demo for Nucleo-L552ZE/GCC

Preconfigured programs for the Nucleo-L552ZE board and the GNU ARM GCC compiler.

Modules

- [Bootloader](#)
Bootloader.
- [User Program](#)
User Program.

5.56.1 Detailed Description

Preconfigured programs for the Nucleo-L552ZE board and the GNU ARM GCC compiler.

For detailed getting started instructions, refer to: https://www.feaser.com/openblt/doku.php?id=manual:demos:nucleo_l552ze_gcc.

5.57 User Program

User Program.

Files

- file [Demo/ARMCM33_STM32L5_Nucleo_L552ZE_GCC/Prog/boot.c](#)
Demo program bootloader interface source file.
- file [Demo/ARMCM33_STM32L5_Nucleo_L552ZE_GCC/Prog/boot.h](#)
Demo program bootloader interface header file.
- file [ARMCM33_STM32L5_Nucleo_L552ZE_GCC/Prog/header.h](#)
Generic header file.
- file [ARMCM33_STM32L5_Nucleo_L552ZE_GCC/Prog/led.c](#)
LED driver source file.
- file [ARMCM33_STM32L5_Nucleo_L552ZE_GCC/Prog/led.h](#)
LED driver header file.
- file [ARMCM33_STM32L5_Nucleo_L552ZE_GCC/Prog/main.c](#)
Demo program application source file.
- file [Demo/ARMCM33_STM32L5_Nucleo_L552ZE_GCC/Prog/timer.c](#)
Timer driver source file.
- file [Demo/ARMCM33_STM32L5_Nucleo_L552ZE_GCC/Prog/timer.h](#)
Timer driver header file.

5.57.1 Detailed Description

User Program.

The intention of the demo user program is two-fold. (1) To test the bootloader, you need some sort of firmware to see if you can perform a firmware update with the bootloader. This program can be used for this purpose. (2) To make firmware programmable by the bootloader, a few adjustments to the firmware are required. The demo user program serves as an example for how these adjustments can be implemented. Additional details on this subject can be found in the port specifics documentation, which is available at: https://www.feaser.com/openblt/doku.php?id=manual:ports:armcm33_stm32l5.

5.58 Bootloader

Bootloader.

Files

- file [ARMCM33_STM32L5_Nucleo_L552ZE_IAR/Boot/blt_conf.h](#)
Bootloader configuration header file.
- file [ARMCM33_STM32L5_Nucleo_L552ZE_IAR/Boot/hooks.c](#)
Bootloader callback source file.
- file [ARMCM33_STM32L5_Nucleo_L552ZE_IAR/Boot/led.c](#)
LED driver source file.
- file [ARMCM33_STM32L5_Nucleo_L552ZE_IAR/Boot/led.h](#)
LED driver header file.
- file [ARMCM33_STM32L5_Nucleo_L552ZE_IAR/Boot/main.c](#)
Bootloader application source file.

5.58.1 Detailed Description

Bootloader.

5.59 Demo for Nucleo-L552ZE/IAR

Preconfigured programs for the Nucleo-L552ZE board and the IAR Embedded Workbench IDE.

Modules

- [Bootloader](#)
Bootloader.
- [User Program](#)
User Program.

5.59.1 Detailed Description

Preconfigured programs for the Nucleo-L552ZE board and the IAR Embedded Workbench IDE.

For detailed getting started instructions, refer to: https://www.feaser.com/openblt/doku.php?id=manual:demos:nucleo_l552ze_iar.

5.60 User Program

User Program.

Files

- file [Demo/ARMCM33_STM32L5_Nucleo_L552ZE_IAR/Prog/boot.c](#)
Demo program bootloader interface source file.
- file [Demo/ARMCM33_STM32L5_Nucleo_L552ZE_IAR/Prog/boot.h](#)
Demo program bootloader interface header file.
- file [ARMCM33_STM32L5_Nucleo_L552ZE_IAR/Prog/header.h](#)
Generic header file.
- file [ARMCM33_STM32L5_Nucleo_L552ZE_IAR/Prog/led.c](#)
LED driver source file.
- file [ARMCM33_STM32L5_Nucleo_L552ZE_IAR/Prog/led.h](#)
LED driver header file.
- file [ARMCM33_STM32L5_Nucleo_L552ZE_IAR/Prog/main.c](#)
Demo program application source file.
- file [Demo/ARMCM33_STM32L5_Nucleo_L552ZE_IAR/Prog/timer.c](#)
Timer driver source file.
- file [Demo/ARMCM33_STM32L5_Nucleo_L552ZE_IAR/Prog/timer.h](#)
Timer driver header file.

5.60.1 Detailed Description

User Program.

The intention of the demo user program is two-fold. (1) To test the bootloader, you need some sort of firmware to see if you can perform a firmware update with the bootloader. This program can be used for this purpose. (2) To make firmware programmable by the bootloader, a few adjustments to the firmware are required. The demo user program serves as an example for how these adjustments can be implemented. Additional details on this subject can be found in the port specifics documentation, which is available at: https://www.feaser.com/openblt/doku.php?id=manual:ports:armcm33_stm32l5.

5.61 Bootloader

Bootloader.

Files

- file [ARMCM33_STM32L5_Nucleo_L552ZE_Keil/Boot/blt_conf.h](#)
Bootloader configuration header file.
- file [ARMCM33_STM32L5_Nucleo_L552ZE_Keil/Boot/hooks.c](#)
Bootloader callback source file.
- file [ARMCM33_STM32L5_Nucleo_L552ZE_Keil/Boot/led.c](#)
LED driver source file.
- file [ARMCM33_STM32L5_Nucleo_L552ZE_Keil/Boot/led.h](#)
LED driver header file.
- file [ARMCM33_STM32L5_Nucleo_L552ZE_Keil/Boot/main.c](#)
Bootloader application source file.

5.61.1 Detailed Description

Bootloader.

5.62 Demo for Nucleo-L552ZE/Keil

Preconfigured programs for the Nucleo-L552ZE board and the Keil MDK for ARM IDE.

Modules

- [Bootloader](#)
Bootloader.
- [User Program](#)
User Program.

5.62.1 Detailed Description

Preconfigured programs for the Nucleo-L552ZE board and the Keil MDK for ARM IDE.

For detailed getting started instructions, refer to: https://www.feaser.com/openblt/doku.php?id=manual:demos:nucleo_l552ze_keil.

5.63 User Program

User Program.

Files

- file [Demo/ARMCM33_STM32L5_Nucleo_L552ZE_Keil/Prog/boot.c](#)
Demo program bootloader interface source file.
- file [Demo/ARMCM33_STM32L5_Nucleo_L552ZE_Keil/Prog/boot.h](#)
Demo program bootloader interface header file.
- file [ARMCM33_STM32L5_Nucleo_L552ZE_Keil/Prog/header.h](#)
Generic header file.
- file [ARMCM33_STM32L5_Nucleo_L552ZE_Keil/Prog/led.c](#)
LED driver source file.
- file [ARMCM33_STM32L5_Nucleo_L552ZE_Keil/Prog/led.h](#)
LED driver header file.
- file [ARMCM33_STM32L5_Nucleo_L552ZE_Keil/Prog/main.c](#)
Demo program application source file.
- file [Demo/ARMCM33_STM32L5_Nucleo_L552ZE_Keil/Prog/timer.c](#)
Timer driver source file.
- file [Demo/ARMCM33_STM32L5_Nucleo_L552ZE_Keil/Prog/timer.h](#)
Timer driver header file.

5.63.1 Detailed Description

User Program.

The intention of the demo user program is two-fold. (1) To test the bootloader, you need some sort of firmware to see if you can perform a firmware update with the bootloader. This program can be used for this purpose. (2) To make firmware programmable by the bootloader, a few adjustments to the firmware are required. The demo user program serves as an example for how these adjustments can be implemented. Additional details on this subject can be found in the port specifics documentation, which is available at: https://www.feaser.com/openblt/doku.php?id=manual:ports:armcm33_stm32l5.

5.64 Bootloader

Bootloader.

Files

- file [ARMCM3_EFM32_Olimex_EM32G880F128STK_GCC/Boot/blt_conf.h](#)
Bootloader configuration header file.
- file [ARMCM3_EFM32_Olimex_EM32G880F128STK_GCC/Boot/cstart.c](#)
Bootloader C startup source file.
- file [ARMCM3_EFM32_Olimex_EM32G880F128STK_GCC/Boot/hooks.c](#)
Bootloader callback source file.
- file [ARMCM3_EFM32_Olimex_EM32G880F128STK_GCC/Boot/main.c](#)
Bootloader application source file.
- file [ARMCM3_EFM32_Olimex_EM32G880F128STK_GCC/Boot/vectors.c](#)
Bootloader interrupt vector table source file.

5.64.1 Detailed Description

Bootloader.

5.65 Demo for Olimex EM-32G880F128-STK/GCC

Preconfigured programs for the Olimex EM-32G880F128-STK and the GCC compiler.

Modules

- [Bootloader](#)
Bootloader.
- [User Program](#)
User Program.

5.65.1 Detailed Description

Preconfigured programs for the Olimex EM-32G880F128-STK and the GCC compiler.

For detailed getting started instructions, refer to: https://www.feaser.com/openblt/doku.php?id=manual:demos:olimex_em32g880f128stk_gcc.

5.66 User Program

User Program.

Files

- file [Demo/ARMCM3_EFM32_Olimex_EM32G880F128STK_GCC/Prog/boot.c](#)
Demo program bootloader interface source file.
- file [Demo/ARMCM3_EFM32_Olimex_EM32G880F128STK_GCC/Prog/boot.h](#)
Demo program bootloader interface header file.
- file [ARMCM3_EFM32_Olimex_EM32G880F128STK_GCC/Prog/cstart.c](#)
Demo program C startup source file.
- file [ARMCM3_EFM32_Olimex_EM32G880F128STK_GCC/Prog/header.h](#)
Generic header file.
- file [ARMCM3_EFM32_Olimex_EM32G880F128STK_GCC/Prog/led.c](#)
LED driver source file.
- file [ARMCM3_EFM32_Olimex_EM32G880F128STK_GCC/Prog/led.h](#)
LED driver header file.
- file [ARMCM3_EFM32_Olimex_EM32G880F128STK_GCC/Prog/main.c](#)
Demo program application source file.
- file [Demo/ARMCM3_EFM32_Olimex_EM32G880F128STK_GCC/Prog/timer.c](#)
Timer driver source file.
- file [Demo/ARMCM3_EFM32_Olimex_EM32G880F128STK_GCC/Prog/timer.h](#)
Timer driver header file.
- file [ARMCM3_EFM32_Olimex_EM32G880F128STK_GCC/Prog/vectors.c](#)
Demo program interrupt vectors source file.

5.66.1 Detailed Description

User Program.

The intention of the demo user program is two-fold. (1) To test the bootloader, you need some sort of firmware to see if you can perform a firmware update with the bootloader. This program can be used for this purpose. (2) To make firmware programmable by the bootloader, a few adjustments to the firmware are required. The demo user program serves as an example for how these adjustments can be implemented. Additional details on this subject can be found in the port specifics documentation, which is available at: https://www.feaser.com/openblt/doku.php?id=manual:ports:armcm3_efm32.

5.67 Bootloader

Bootloader.

Files

- file [ARMCM3_EFM32_Olimex_EM32G880F128STK_IAR/Boot/blt_conf.h](#)
Bootloader configuration header file.
- file [ARMCM3_EFM32_Olimex_EM32G880F128STK_IAR/Boot/hooks.c](#)
Bootloader callback source file.
- file [ARMCM3_EFM32_Olimex_EM32G880F128STK_IAR/Boot/main.c](#)
Bootloader application source file.
- file [ARMCM3_EFM32_Olimex_EM32G880F128STK_IAR/Boot/vectors.c](#)
Bootloader interrupt vector table source file.

5.67.1 Detailed Description

Bootloader.

5.68 Demo for Olimex EM-32G880F128-STK/IAR

Preconfigured programs for the Olimex EM-32G880F128-STK and the IAR Embedded Workbench IDE.

Modules

- [Bootloader](#)
Bootloader.
- [User Program](#)
User Program.

5.68.1 Detailed Description

Preconfigured programs for the Olimex EM-32G880F128-STK and the IAR Embedded Workbench IDE.

For detailed getting started instructions, refer to: https://www.feaser.com/openblt/doku.php?id=manual:demos:olimex_em32g880f128stk_iar.

5.69 User Program

User Program.

Files

- file [Demo/ARMCM3_EFM32_Olimex_EM32G880F128STK_IAR/Prog/boot.c](#)
Demo program bootloader interface source file.
- file [Demo/ARMCM3_EFM32_Olimex_EM32G880F128STK_IAR/Prog/boot.h](#)
Demo program bootloader interface header file.
- file [ARMCM3_EFM32_Olimex_EM32G880F128STK_IAR/Prog/header.h](#)
Generic header file.
- file [ARMCM3_EFM32_Olimex_EM32G880F128STK_IAR/Prog/led.c](#)
LED driver source file.
- file [ARMCM3_EFM32_Olimex_EM32G880F128STK_IAR/Prog/led.h](#)
LED driver header file.
- file [ARMCM3_EFM32_Olimex_EM32G880F128STK_IAR/Prog/main.c](#)
Demo program application source file.
- file [Demo/ARMCM3_EFM32_Olimex_EM32G880F128STK_IAR/Prog/timer.c](#)
Timer driver source file.
- file [Demo/ARMCM3_EFM32_Olimex_EM32G880F128STK_IAR/Prog/timer.h](#)
Timer driver header file.
- file [ARMCM3_EFM32_Olimex_EM32G880F128STK_IAR/Prog/vectors.c](#)
Demo program interrupt vectors source file.

5.69.1 Detailed Description

User Program.

The intention of the demo user program is two-fold. (1) To test the bootloader, you need some sort of firmware to see if you can perform a firmware update with the bootloader. This program can be used for this purpose. (2) To make firmware programmable by the bootloader, a few adjustments to the firmware are required. The demo user program serves as an example for how these adjustments can be implemented. Additional details on this subject can be found in the port specifics documentation, which is available at: https://www.feaser.com/openblt/doku.php?id=manual:ports:armcm3_efm32.

5.70 Bootloader

Bootloader.

Files

- file [ARMCM3_LM3S_EK_LM3S6965_GCC/Boot/blt_conf.h](#)
Bootloader configuration header file.
- file [ARMCM3_LM3S_EK_LM3S6965_GCC/Boot/cstart.c](#)
Bootloader C startup source file.
- file [ARMCM3_LM3S_EK_LM3S6965_GCC/Boot/hooks.c](#)
Bootloader callback source file.
- file [ARMCM3_LM3S_EK_LM3S6965_GCC/Boot/main.c](#)
Bootloader application source file.
- file [ARMCM3_LM3S_EK_LM3S6965_GCC/Boot/shared_params.c](#)
Shared RAM parameters source file.
- file [ARMCM3_LM3S_EK_LM3S6965_GCC/Boot/shared_params.h](#)
Shared RAM parameters header file.
- file [ARMCM3_LM3S_EK_LM3S6965_GCC/Boot/vectors.c](#)
Bootloader application source file.

5.70.1 Detailed Description

Bootloader.

5.71 Demo for Texas Instruments EK-LM3S6965/GCC

Preconfigured programs for the Texas Instruments EK-LM3S6965 and the GCC compiler.

Modules

- [Bootloader](#)
Bootloader.
- [User Program](#)
User Program.

5.71.1 Detailed Description

Preconfigured programs for the Texas Instruments EK-LM3S6965 and the GCC compiler.

For detailed getting started instructions, refer to: https://www.feaser.com/openblt/doku.php?id=manual:demos:ek_lm3s6965_gcc.

5.72 User Program

User Program.

Files

- file [Demo/ARMCM3_LM3S_EK_LM3S6965_GCC/Prog/boot.c](#)
Demo program bootloader interface source file.
- file [Demo/ARMCM3_LM3S_EK_LM3S6965_GCC/Prog/boot.h](#)
Demo program bootloader interface header file.
- file [ARMCM3_LM3S_EK_LM3S6965_GCC/Prog/cstart.c](#)
Demo program C startup source file.
- file [ARMCM3_LM3S_EK_LM3S6965_GCC/Prog/header.h](#)
Generic header file.
- file [ARMCM3_LM3S_EK_LM3S6965_GCC/Prog/led.c](#)
LED driver source file.
- file [ARMCM3_LM3S_EK_LM3S6965_GCC/Prog/led.h](#)
LED driver header file.
- file [ARMCM3_LM3S_EK_LM3S6965_GCC/Prog/main.c](#)
Demo program application source file.
- file [Demo/ARMCM3_LM3S_EK_LM3S6965_GCC/Prog/net.c](#)
Network application for the uIP TCP/IP stack.
- file [Demo/ARMCM3_LM3S_EK_LM3S6965_GCC/Prog/net.h](#)
Network application for the uIP TCP/IP stack.
- file [ARMCM3_LM3S_EK_LM3S6965_GCC/Prog/shared_params.c](#)
Shared RAM parameters source file.
- file [ARMCM3_LM3S_EK_LM3S6965_GCC/Prog/shared_params.h](#)
Shared RAM parameters header file.
- file [ARMCM3_LM3S_EK_LM3S6965_GCC/Prog/time.c](#)
Timer driver source file.
- file [ARMCM3_LM3S_EK_LM3S6965_GCC/Prog/time.h](#)
Timer driver header file.
- file [ARMCM3_LM3S_EK_LM3S6965_GCC/Prog/vectors.c](#)
Demo program interrupt vectors source file.

5.72.1 Detailed Description

User Program.

The intention of the demo user program is two-fold. (1) To test the bootloader, you need some sort of firmware to see if you can perform a firmware update with the bootloader. This program can be used for this purpose. (2) To make firmware programmable by the bootloader, a few adjustments to the firmware are required. The demo user program serves as an example for how these adjustments can be implemented. Additional details on this subject can be found in the port specifics documentation, which is available at: https://www.feaser.com/openblt/doku.php?id=manual:ports:armcm3_lm3s.

5.73 Bootloader

Bootloader.

Files

- file [ARMCM3_LM3S_EK_LM3S6965_IAR/Boot/blt_conf.h](#)
Bootloader configuration header file.
- file [ARMCM3_LM3S_EK_LM3S6965_IAR/Boot/hooks.c](#)
Bootloader callback source file.
- file [ARMCM3_LM3S_EK_LM3S6965_IAR/Boot/main.c](#)
Bootloader application source file.
- file [ARMCM3_LM3S_EK_LM3S6965_IAR/Boot/shared_params.c](#)
Shared RAM parameters source file.
- file [ARMCM3_LM3S_EK_LM3S6965_IAR/Boot/shared_params.h](#)
Shared RAM parameters header file.
- file [ARMCM3_LM3S_EK_LM3S6965_IAR/Boot/vectors.c](#)
Bootloader interrupt vector table source file.

5.73.1 Detailed Description

Bootloader.

5.74 Demo for Texas Instruments EK-LM3S6965/IAR

Preconfigured programs for the Texas Instruments EK-LM3S6965 and the IAR Embedded Workbench IDE.

Modules

- [Bootloader](#)
Bootloader.
- [User Program](#)
User Program.

5.74.1 Detailed Description

Preconfigured programs for the Texas Instruments EK-LM3S6965 and the IAR Embedded Workbench IDE.

For detailed getting started instructions, refer to: https://www.feaser.com/openblt/doku.php?id=manual:demos:ek_lm3s6965_iar.

5.75 User Program

User Program.

Files

- file [Demo/ARMCM3_LM3S_EK_LM3S6965_IAR/Prog/boot.c](#)
Demo program bootloader interface source file.
- file [Demo/ARMCM3_LM3S_EK_LM3S6965_IAR/Prog/boot.h](#)
Demo program bootloader interface header file.
- file [ARMCM3_LM3S_EK_LM3S6965_IAR/Prog/header.h](#)
Generic header file.
- file [ARMCM3_LM3S_EK_LM3S6965_IAR/Prog/led.c](#)
LED driver source file.
- file [ARMCM3_LM3S_EK_LM3S6965_IAR/Prog/led.h](#)
LED driver header file.
- file [ARMCM3_LM3S_EK_LM3S6965_IAR/Prog/main.c](#)
Demo program application source file.
- file [Demo/ARMCM3_LM3S_EK_LM3S6965_IAR/Prog/net.c](#)
Network application for the uIP TCP/IP stack.
- file [Demo/ARMCM3_LM3S_EK_LM3S6965_IAR/Prog/net.h](#)
Network application for the uIP TCP/IP stack.
- file [ARMCM3_LM3S_EK_LM3S6965_IAR/Prog/shared_params.c](#)
Shared RAM parameters source file.
- file [ARMCM3_LM3S_EK_LM3S6965_IAR/Prog/shared_params.h](#)
Shared RAM parameters header file.
- file [ARMCM3_LM3S_EK_LM3S6965_IAR/Prog/time.c](#)
Timer driver source file.
- file [ARMCM3_LM3S_EK_LM3S6965_IAR/Prog/time.h](#)
Timer driver header file.
- file [ARMCM3_LM3S_EK_LM3S6965_IAR/Prog/vectors.c](#)
Demo program interrupt vectors source file.

5.75.1 Detailed Description

User Program.

The intention of the demo user program is two-fold. (1) To test the bootloader, you need some sort of firmware to see if you can perform a firmware update with the bootloader. This program can be used for this purpose. (2) To make firmware programmable by the bootloader, a few adjustments to the firmware are required. The demo user program serves as an example for how these adjustments can be implemented. Additional details on this subject can be found in the port specifics documentation, which is available at: https://www.feaser.com/openblt/doku.php?id=manual:ports:armcm3_lm3s.

5.76 Bootloader

Bootloader.

Files

- file [ARMCM3_LM3S_EK_LM3S8962_GCC/Boot/blt_conf.h](#)
Bootloader configuration header file.
- file [ARMCM3_LM3S_EK_LM3S8962_GCC/Boot/cstart.c](#)
Bootloader C startup source file.
- file [ARMCM3_LM3S_EK_LM3S8962_GCC/Boot/flash_layout.c](#)
Custom flash layout table source file.
- file [ARMCM3_LM3S_EK_LM3S8962_GCC/Boot/hooks.c](#)
Bootloader callback source file.
- file [ARMCM3_LM3S_EK_LM3S8962_GCC/Boot/main.c](#)
Bootloader application source file.
- file [ARMCM3_LM3S_EK_LM3S8962_GCC/Boot/vectors.c](#)
Bootloader interrupt vector table source file.

5.76.1 Detailed Description

Bootloader.

5.77 Demo for Texas Instruments EK-LM3S8962/GCC

Preconfigured programs for the Texas Instruments EK-LM3S8962 and the GCC compiler.

Modules

- [Bootloader](#)
Bootloader.
- [User Program](#)
User Program.

5.77.1 Detailed Description

Preconfigured programs for the Texas Instruments EK-LM3S8962 and the GCC compiler.

For detailed getting started instructions, refer to: https://www.feaser.com/openblt/doku.php?id=manual:demos:ek_lm3s8962_gcc.

5.78 User Program

User Program.

Files

- file [Demo/ARMCM3_LM3S_EK_LM3S8962_GCC/Prog/boot.c](#)
Demo program bootloader interface source file.
- file [Demo/ARMCM3_LM3S_EK_LM3S8962_GCC/Prog/boot.h](#)
Demo program bootloader interface header file.
- file [ARMCM3_LM3S_EK_LM3S8962_GCC/Prog/cstart.c](#)
Demo program C startup source file.
- file [ARMCM3_LM3S_EK_LM3S8962_GCC/Prog/header.h](#)
Generic header file.
- file [ARMCM3_LM3S_EK_LM3S8962_GCC/Prog/led.c](#)
LED driver source file.
- file [ARMCM3_LM3S_EK_LM3S8962_GCC/Prog/led.h](#)
LED driver header file.
- file [ARMCM3_LM3S_EK_LM3S8962_GCC/Prog/main.c](#)
Demo program application source file.
- file [ARMCM3_LM3S_EK_LM3S8962_GCC/Prog/time.c](#)
Timer driver source file.
- file [ARMCM3_LM3S_EK_LM3S8962_GCC/Prog/time.h](#)
Timer driver header file.
- file [ARMCM3_LM3S_EK_LM3S8962_GCC/Prog/vectors.c](#)
Demo program interrupt vectors source file.

5.78.1 Detailed Description

User Program.

The intention of the demo user program is two-fold. (1) To test the bootloader, you need some sort of firmware to see if you can perform a firmware update with the bootloader. This program can be used for this purpose. (2) To make firmware programmable by the bootloader, a few adjustments to the firmware are required. The demo user program serves as an example for how these adjustments can be implemented. Additional details on this subject can be found in the port specifics documentation, which is available at: https://www.feaser.com/openblt/doku.php?id=manual:ports:armcm3_lm3s.

5.79 Bootloader

Bootloader.

Files

- file [ARMCM3_LM3S_EK_LM3S8962_IAR/Boot/blt_conf.h](#)
Bootloader configuration header file.
- file [ARMCM3_LM3S_EK_LM3S8962_IAR/Boot/flash_layout.c](#)
Custom flash layout table source file.
- file [ARMCM3_LM3S_EK_LM3S8962_IAR/Boot/hooks.c](#)
Bootloader callback source file.
- file [ARMCM3_LM3S_EK_LM3S8962_IAR/Boot/main.c](#)
Bootloader application source file.
- file [ARMCM3_LM3S_EK_LM3S8962_IAR/Boot/vectors.c](#)
Bootloader interrupt vector table source file.

5.79.1 Detailed Description

Bootloader.

5.80 Demo for Texas Instruments EK-LM3S8962/IAR

Preconfigured programs for the Texas Instruments EK-LM3S8962 and the IAR Embedded Workbench IDE.

Modules

- [Bootloader](#)
Bootloader.
- [User Program](#)
User Program.

5.80.1 Detailed Description

Preconfigured programs for the Texas Instruments EK-LM3S8962 and the IAR Embedded Workbench IDE.

For detailed getting started instructions, refer to: https://www.feaser.com/openblt/doku.php?id=manual:demos:ek_lm3s8962_iar.

5.81 User Program

User Program.

Files

- file [Demo/ARMCM3_LM3S_EK_LM3S8962_IAR/Prog/boot.c](#)
Demo program bootloader interface source file.
- file [Demo/ARMCM3_LM3S_EK_LM3S8962_IAR/Prog/boot.h](#)
Demo program bootloader interface header file.
- file [ARMCM3_LM3S_EK_LM3S8962_IAR/Prog/header.h](#)
Generic header file.
- file [ARMCM3_LM3S_EK_LM3S8962_IAR/Prog/led.c](#)
LED driver source file.
- file [ARMCM3_LM3S_EK_LM3S8962_IAR/Prog/led.h](#)
LED driver header file.
- file [ARMCM3_LM3S_EK_LM3S8962_IAR/Prog/main.c](#)
Demo program application source file.
- file [ARMCM3_LM3S_EK_LM3S8962_IAR/Prog/time.c](#)
Timer driver source file.
- file [ARMCM3_LM3S_EK_LM3S8962_IAR/Prog/time.h](#)
Timer driver header file.
- file [ARMCM3_LM3S_EK_LM3S8962_IAR/Prog/vectors.c](#)
Demo program interrupt vectors source file.

5.81.1 Detailed Description

User Program.

The intention of the demo user program is two-fold. (1) To test the bootloader, you need some sort of firmware to see if you can perform a firmware update with the bootloader. This program can be used for this purpose. (2) To make firmware programmable by the bootloader, a few adjustments to the firmware are required. The demo user program serves as an example for how these adjustments can be implemented. Additional details on this subject can be found in the port specifics documentation, which is available at: https://www.feaser.com/openblt/doku.php?id=manual:ports:armcm3_lm3s.

5.82 Bootloader

Bootloader.

Files

- file [ARMCM3_STM32F1_Nucleo_F103RB_CubeIDE/Boot/App/app.c](#)
Bootloader application source file.
- file [ARMCM3_STM32F1_Nucleo_F103RB_CubeIDE/Boot/App/app.h](#)
Bootloader application header file.
- file [ARMCM3_STM32F1_Nucleo_F103RB_CubeIDE/Boot/App/blt_conf.h](#)
Bootloader configuration header file.
- file [ARMCM3_STM32F1_Nucleo_F103RB_CubeIDE/Boot/App/flash_layout.c](#)
Custom flash layout table source file.
- file [ARMCM3_STM32F1_Nucleo_F103RB_CubeIDE/Boot/App/hooks.c](#)
Bootloader callback source file.
- file [ARMCM3_STM32F1_Nucleo_F103RB_CubeIDE/Boot/App/led.c](#)
LED driver source file.
- file [ARMCM3_STM32F1_Nucleo_F103RB_CubeIDE/Boot/App/led.h](#)
LED driver header file.

5.82.1 Detailed Description

Bootloader.

5.83 Demo for Nucleo-F103RB/STM32CubeIDE

Preconfigured programs for the Nucleo-F103RB and the ST STM32CubeIDE toolchain.

Modules

- [Bootloader](#)
Bootloader.
- [User Program](#)
User Program.

5.83.1 Detailed Description

Preconfigured programs for the Nucleo-F103RB and the ST STM32CubeIDE toolchain.

For detailed getting started instructions, refer to: https://www.feaser.com/openblt/doku.php?id=manual:demos:nucleo_f103rb_cubeide.

5.84 User Program

User Program.

Files

- file [ARMCM3_STM32F1_Nucleo_F103RB_CubeIDE/Prog/App/app.c](#)
User program application source file.
- file [ARMCM3_STM32F1_Nucleo_F103RB_CubeIDE/Prog/App/app.h](#)
User program application header file.
- file [Demo/ARMCM3_STM32F1_Nucleo_F103RB_CubeIDE/Prog/App/boot.c](#)
Demo program bootloader interface source file.
- file [Demo/ARMCM3_STM32F1_Nucleo_F103RB_CubeIDE/Prog/App/boot.h](#)
Demo program bootloader interface header file.
- file [ARMCM3_STM32F1_Nucleo_F103RB_CubeIDE/Prog/App/header.h](#)
Generic header file.
- file [ARMCM3_STM32F1_Nucleo_F103RB_CubeIDE/Prog/App/led.c](#)
LED driver source file.
- file [ARMCM3_STM32F1_Nucleo_F103RB_CubeIDE/Prog/App/led.h](#)
LED driver header file.
- file [Demo/ARMCM3_STM32F1_Nucleo_F103RB_CubeIDE/Prog/App/timer.c](#)
Timer driver source file.
- file [Demo/ARMCM3_STM32F1_Nucleo_F103RB_CubeIDE/Prog/App/timer.h](#)
Timer driver header file.

5.84.1 Detailed Description

User Program.

The intention of the demo user program is two-fold. (1) To test the bootloader, you need some sort of firmware to see if you can perform a firmware update with the bootloader. This program can be used for this purpose. (2) To make firmware programmable by the bootloader, a few adjustments to the firmware are required. The demo user program serves as an example for how these adjustments can be implemented. Additional details on this subject can be found in the port specifics documentation, which is available at: https://www.feaser.com/openblt/doku.php?id=manual:ports:armcm3_stm32.

5.85 Bootloader

Bootloader.

Files

- file [ARMCM3_STM32F1_Nucleo_F103RB_GCC/Boot/blt_conf.h](#)
Bootloader configuration header file.
- file [ARMCM3_STM32F1_Nucleo_F103RB_GCC/Boot/flash_layout.c](#)
Custom flash layout table source file.
- file [ARMCM3_STM32F1_Nucleo_F103RB_GCC/Boot/hooks.c](#)
Bootloader callback source file.
- file [ARMCM3_STM32F1_Nucleo_F103RB_GCC/Boot/led.c](#)
LED driver source file.
- file [ARMCM3_STM32F1_Nucleo_F103RB_GCC/Boot/led.h](#)
LED driver header file.
- file [ARMCM3_STM32F1_Nucleo_F103RB_GCC/Boot/main.c](#)
Bootloader application source file.

5.85.1 Detailed Description

Bootloader.

5.86 Demo for Nucleo-F103RB/GCC

Preconfigured programs for the Nucleo-F103RB and the GNU ARM GCC compiler.

Modules

- [Bootloader](#)
Bootloader.
- [User Program](#)
User Program.

5.86.1 Detailed Description

Preconfigured programs for the Nucleo-F103RB and the GNU ARM GCC compiler.

For detailed getting started instructions, refer to: https://www.feaser.com/openblt/doku.php?id=manual:demos:nucleo_f103rb_gcc.

5.87 User Program

User Program.

Files

- file [Demo/ARMCM3_STM32F1_Nucleo_F103RB_GCC/Prog/boot.c](#)
Demo program bootloader interface source file.
- file [Demo/ARMCM3_STM32F1_Nucleo_F103RB_GCC/Prog/boot.h](#)
Demo program bootloader interface header file.
- file [ARMCM3_STM32F1_Nucleo_F103RB_GCC/Prog/header.h](#)
Generic header file.
- file [ARMCM3_STM32F1_Nucleo_F103RB_GCC/Prog/led.c](#)
LED driver source file.
- file [ARMCM3_STM32F1_Nucleo_F103RB_GCC/Prog/led.h](#)
LED driver header file.
- file [ARMCM3_STM32F1_Nucleo_F103RB_GCC/Prog/main.c](#)
Demo program application source file.
- file [Demo/ARMCM3_STM32F1_Nucleo_F103RB_GCC/Prog/timer.c](#)
Timer driver source file.
- file [Demo/ARMCM3_STM32F1_Nucleo_F103RB_GCC/Prog/timer.h](#)
Timer driver header file.

5.87.1 Detailed Description

User Program.

The intention of the demo user program is two-fold. (1) To test the bootloader, you need some sort of firmware to see if you can perform a firmware update with the bootloader. This program can be used for this purpose. (2) To make firmware programmable by the bootloader, a few adjustments to the firmware are required. The demo user program serves as an example for how these adjustments can be implemented. Additional details on this subject can be found in the port specifics documentation, which is available at: https://www.feaser.com/openblt/doku.php?id=manual:ports:armcm3_stm32.

5.88 Bootloader

Bootloader.

Files

- file [ARMCM3_STM32F1_Nucleo_F103RB_IAR/Boot/blt_conf.h](#)
Bootloader configuration header file.
- file [ARMCM3_STM32F1_Nucleo_F103RB_IAR/Boot/flash_layout.c](#)
Custom flash layout table source file.
- file [ARMCM3_STM32F1_Nucleo_F103RB_IAR/Boot/hooks.c](#)
Bootloader callback source file.
- file [ARMCM3_STM32F1_Nucleo_F103RB_IAR/Boot/led.c](#)
LED driver source file.
- file [ARMCM3_STM32F1_Nucleo_F103RB_IAR/Boot/led.h](#)
LED driver header file.
- file [ARMCM3_STM32F1_Nucleo_F103RB_IAR/Boot/main.c](#)
Bootloader application source file.

5.88.1 Detailed Description

Bootloader.

5.89 Demo for Nucleo-F103RB

Preconfigured programs for the Nucleo-F103RB and the IAR Embedded Workbench IDE.

Modules

- [Bootloader](#)
Bootloader.
- [User Program](#)
User Program.

5.89.1 Detailed Description

Preconfigured programs for the Nucleo-F103RB and the IAR Embedded Workbench IDE.

For detailed getting started instructions, refer to: https://www.feaser.com/openblt/doku.php?id=manual:demos:nucleo_f103rb_iar.

5.90 User Program

User Program.

Files

- file [Demo/ARMCM3_STM32F1_Nucleo_F103RB_IAR/Prog/boot.c](#)
Demo program bootloader interface source file.
- file [Demo/ARMCM3_STM32F1_Nucleo_F103RB_IAR/Prog/boot.h](#)
Demo program bootloader interface header file.
- file [ARMCM3_STM32F1_Nucleo_F103RB_IAR/Prog/header.h](#)
Generic header file.
- file [ARMCM3_STM32F1_Nucleo_F103RB_IAR/Prog/led.c](#)
LED driver source file.
- file [ARMCM3_STM32F1_Nucleo_F103RB_IAR/Prog/led.h](#)
LED driver header file.
- file [ARMCM3_STM32F1_Nucleo_F103RB_IAR/Prog/main.c](#)
Demo program application source file.
- file [Demo/ARMCM3_STM32F1_Nucleo_F103RB_IAR/Prog/timer.c](#)
Timer driver source file.
- file [Demo/ARMCM3_STM32F1_Nucleo_F103RB_IAR/Prog/timer.h](#)
Timer driver header file.

5.90.1 Detailed Description

User Program.

The intention of the demo user program is two-fold. (1) To test the bootloader, you need some sort of firmware to see if you can perform a firmware update with the bootloader. This program can be used for this purpose. (2) To make firmware programmable by the bootloader, a few adjustments to the firmware are required. The demo user program serves as an example for how these adjustments can be implemented. Additional details on this subject can be found in the port specifics documentation, which is available at: https://www.feaser.com/openblt/doku.php?id=manual:ports:armcm3_stm32.

5.91 Bootloader

Bootloader.

Files

- file [ARMCM3_STM32F1_Nucleo_F103RB_Keil/Boot/blt_conf.h](#)
Bootloader configuration header file.
- file [ARMCM3_STM32F1_Nucleo_F103RB_Keil/Boot/flash_layout.c](#)
Custom flash layout table source file.
- file [ARMCM3_STM32F1_Nucleo_F103RB_Keil/Boot/hooks.c](#)
Bootloader callback source file.
- file [ARMCM3_STM32F1_Nucleo_F103RB_Keil/Boot/led.c](#)
LED driver source file.
- file [ARMCM3_STM32F1_Nucleo_F103RB_Keil/Boot/led.h](#)
LED driver header file.
- file [ARMCM3_STM32F1_Nucleo_F103RB_Keil/Boot/main.c](#)
Bootloader application source file.

5.91.1 Detailed Description

Bootloader.

5.92 Demo for Nucleo-F103RB/Keil

Preconfigured programs for the Nucleo-F103RB and the Keil MDK toolchain.

Modules

- [Bootloader](#)
Bootloader.
- [User Program](#)
User Program.

5.92.1 Detailed Description

Preconfigured programs for the Nucleo-F103RB and the Keil MDK toolchain.

For detailed getting started instructions, refer to: https://www.feaser.com/openblt/doku.php?id=manual:demos:nucleo_f103rb_keil.

5.93 User Program

User Program.

Files

- file [Demo/ARMCM3_STM32F1_Nucleo_F103RB_Keil/Prog/boot.c](#)
Demo program bootloader interface source file.
- file [Demo/ARMCM3_STM32F1_Nucleo_F103RB_Keil/Prog/boot.h](#)
Demo program bootloader interface header file.
- file [ARMCM3_STM32F1_Nucleo_F103RB_Keil/Prog/header.h](#)
Generic header file.
- file [ARMCM3_STM32F1_Nucleo_F103RB_Keil/Prog/led.c](#)
LED driver source file.
- file [ARMCM3_STM32F1_Nucleo_F103RB_Keil/Prog/led.h](#)
LED driver header file.
- file [ARMCM3_STM32F1_Nucleo_F103RB_Keil/Prog/main.c](#)
Demo program application source file.
- file [Demo/ARMCM3_STM32F1_Nucleo_F103RB_Keil/Prog/timer.c](#)
Timer driver source file.
- file [Demo/ARMCM3_STM32F1_Nucleo_F103RB_Keil/Prog/timer.h](#)
Timer driver header file.

5.93.1 Detailed Description

User Program.

The intention of the demo user program is two-fold. (1) To test the bootloader, you need some sort of firmware to see if you can perform a firmware update with the bootloader. This program can be used for this purpose. (2) To make firmware programmable by the bootloader, a few adjustments to the firmware are required. The demo user program serves as an example for how these adjustments can be implemented. Additional details on this subject can be found in the port specifics documentation, which is available at: https://www.feaser.com/openblt/doku.php?id=manual:ports:armcm3_stm32.

5.94 Bootloader

Bootloader.

Files

- file [ARMCM3_STM32F1_Olimex_STM32H103_CubeIDE/Boot/App/app.c](#)
Bootloader application source file.
- file [ARMCM3_STM32F1_Olimex_STM32H103_CubeIDE/Boot/App/app.h](#)
Bootloader application header file.
- file [ARMCM3_STM32F1_Olimex_STM32H103_CubeIDE/Boot/App/blt_conf.h](#)
Bootloader configuration header file.
- file [ARMCM3_STM32F1_Olimex_STM32H103_CubeIDE/Boot/App/flash_layout.c](#)
Custom flash layout table source file.
- file [ARMCM3_STM32F1_Olimex_STM32H103_CubeIDE/Boot/App/hooks.c](#)
Bootloader callback source file.
- file [ARMCM3_STM32F1_Olimex_STM32H103_CubeIDE/Boot/App/led.c](#)
LED driver source file.
- file [ARMCM3_STM32F1_Olimex_STM32H103_CubeIDE/Boot/App/led.h](#)
LED driver header file.

5.94.1 Detailed Description

Bootloader.

5.95 Demo for Olimex STM32-H103/STM32CubeIDE

Preconfigured programs for the Olimex STM32-H103 and the ST STM32CubeIDE toolchain.

Modules

- [Bootloader](#)
Bootloader.
- [User Program](#)
User Program.

5.95.1 Detailed Description

Preconfigured programs for the Olimex STM32-H103 and the ST STM32CubeIDE toolchain.

For detailed getting started instructions, refer to: https://www.feaser.com/openblt/doku.php?id=manual:demos:olimex_stm32h103_cubeide.

5.96 User Program

User Program.

Files

- file [ARMCM3_STM32F1_Olimex_STM32H103_CubeIDE/Prog/App/app.c](#)
User program application source file.
- file [ARMCM3_STM32F1_Olimex_STM32H103_CubeIDE/Prog/App/app.h](#)
User program application header file.
- file [ARMCM3_STM32F1_Olimex_STM32H103_CubeIDE/Prog/App/header.h](#)
Generic header file.
- file [ARMCM3_STM32F1_Olimex_STM32H103_CubeIDE/Prog/App/led.c](#)
LED driver source file.
- file [ARMCM3_STM32F1_Olimex_STM32H103_CubeIDE/Prog/App/led.h](#)
LED driver header file.
- file [Demo/ARMCM3_STM32F1_Olimex_STM32H103_CubeIDE/Prog/App/timer.c](#)
Timer driver source file.
- file [Demo/ARMCM3_STM32F1_Olimex_STM32H103_CubeIDE/Prog/App/timer.h](#)
Timer driver header file.

5.96.1 Detailed Description

User Program.

The intention of the demo user program is two-fold. (1) To test the bootloader, you need some sort of firmware to see if you can perform a firmware update with the bootloader. This program can be used for this purpose. (2) To make firmware programmable by the bootloader, a few adjustments to the firmware are required. The demo user program serves as an example for how these adjustments can be implemented. Additional details on this subject can be found in the port specifics documentation, which is available at: https://www.feaser.com/openblt/doku.php?id=manual:ports:armcm3_stm32.

5.97 Bootloader

Bootloader.

Files

- file [ARMCM3_STM32F1_Olimex_STM32H103_GCC/Boot/blt_conf.h](#)
Bootloader configuration header file.
- file [ARMCM3_STM32F1_Olimex_STM32H103_GCC/Boot/flash_layout.c](#)
Custom flash layout table source file.
- file [ARMCM3_STM32F1_Olimex_STM32H103_GCC/Boot/hooks.c](#)
Bootloader callback source file.
- file [ARMCM3_STM32F1_Olimex_STM32H103_GCC/Boot/led.c](#)
LED driver source file.
- file [ARMCM3_STM32F1_Olimex_STM32H103_GCC/Boot/led.h](#)
LED driver header file.
- file [ARMCM3_STM32F1_Olimex_STM32H103_GCC/Boot/main.c](#)
Bootloader application source file.

5.97.1 Detailed Description

Bootloader.

5.98 Demo for Olimex STM32-H103/GCC

Preconfigured programs for the Olimex STM32-H103 and the GCC compiler.

Modules

- [Bootloader](#)
Bootloader.
- [User Program](#)
User Program.

5.98.1 Detailed Description

Preconfigured programs for the Olimex STM32-H103 and the GCC compiler.

For detailed getting started instructions, refer to: https://www.feaser.com/openblt/doku.php?id=manual:demos:olimex_stm32h103_gcc.

5.99 User Program

User Program.

Files

- file [ARMCM3_STM32F1_Olimex_STM32H103_GCC/Prog/header.h](#)
Generic header file.
- file [ARMCM3_STM32F1_Olimex_STM32H103_GCC/Prog/led.c](#)
LED driver source file.
- file [ARMCM3_STM32F1_Olimex_STM32H103_GCC/Prog/led.h](#)
LED driver header file.
- file [ARMCM3_STM32F1_Olimex_STM32H103_GCC/Prog/main.c](#)
Demo program application source file.
- file [Demo/ARMCM3_STM32F1_Olimex_STM32H103_GCC/Prog/timer.c](#)
Timer driver source file.
- file [Demo/ARMCM3_STM32F1_Olimex_STM32H103_GCC/Prog/timer.h](#)
Timer driver header file.

5.99.1 Detailed Description

User Program.

The intention of the demo user program is two-fold. (1) To test the bootloader, you need some sort of firmware to see if you can perform a firmware update with the bootloader. This program can be used for this purpose. (2) To make firmware programmable by the bootloader, a few adjustments to the firmware are required. The demo user program serves as an example for how these adjustments can be implemented. Additional details on this subject can be found in the port specifics documentation, which is available at: <https://www.feaser.com/openblt/doku.php?id=manual:ports:armcm3-stm32>.

5.100 Bootloader

Bootloader.

Files

- file [ARMCM3_STM32F1_Olimex_STM32H103_IAR/Boot/blt_conf.h](#)
Bootloader configuration header file.
- file [ARMCM3_STM32F1_Olimex_STM32H103_IAR/Boot/flash_layout.c](#)
Custom flash layout table source file.
- file [ARMCM3_STM32F1_Olimex_STM32H103_IAR/Boot/hooks.c](#)
Bootloader callback source file.
- file [ARMCM3_STM32F1_Olimex_STM32H103_IAR/Boot/led.c](#)
LED driver source file.
- file [ARMCM3_STM32F1_Olimex_STM32H103_IAR/Boot/led.h](#)
LED driver header file.
- file [ARMCM3_STM32F1_Olimex_STM32H103_IAR/Boot/main.c](#)
Bootloader application source file.

5.100.1 Detailed Description

Bootloader.

5.101 Demo for Olimex STM32-H103/IAR

Preconfigured programs for the Olimex STM32-H103 and the IAR Embedded Workbench IDE.

Modules

- [Bootloader](#)
Bootloader.
- [User Program](#)
User Program.

5.101.1 Detailed Description

Preconfigured programs for the Olimex STM32-H103 and the IAR Embedded Workbench IDE.

For detailed getting started instructions, refer to: https://www.feaser.com/openblt/doku.php?id=manual:demos:olimex_stm32h103_iar.

5.102 User Program

User Program.

Files

- file [ARMCM3_STM32F1_Olimex_STM32H103_IAR/Prog/header.h](#)
Generic header file.
- file [ARMCM3_STM32F1_Olimex_STM32H103_IAR/Prog/led.c](#)
LED driver source file.
- file [ARMCM3_STM32F1_Olimex_STM32H103_IAR/Prog/led.h](#)
LED driver header file.
- file [ARMCM3_STM32F1_Olimex_STM32H103_IAR/Prog/main.c](#)
Demo program application source file.
- file [Demo/ARMCM3_STM32F1_Olimex_STM32H103_IAR/Prog/timer.c](#)
Timer driver source file.
- file [Demo/ARMCM3_STM32F1_Olimex_STM32H103_IAR/Prog/timer.h](#)
Timer driver header file.

5.102.1 Detailed Description

User Program.

The intention of the demo user program is two-fold. (1) To test the bootloader, you need some sort of firmware to see if you can perform a firmware update with the bootloader. This program can be used for this purpose. (2) To make firmware programmable by the bootloader, a few adjustments to the firmware are required. The demo user program serves as an example for how these adjustments can be implemented. Additional details on this subject can be found in the port specifics documentation, which is available at: https://www.feaser.com/openblt/doku.php?id=manual:ports:armcm3_stm32.

5.103 Bootloader

Bootloader.

Files

- file [ARMCM3_STM32F1_Olimex_STM32H103_Keil/Boot/blt_conf.h](#)
Bootloader configuration header file.
- file [ARMCM3_STM32F1_Olimex_STM32H103_Keil/Boot/flash_layout.c](#)
Custom flash layout table source file.
- file [ARMCM3_STM32F1_Olimex_STM32H103_Keil/Boot/hooks.c](#)
Bootloader callback source file.
- file [ARMCM3_STM32F1_Olimex_STM32H103_Keil/Boot/led.c](#)
LED driver source file.
- file [ARMCM3_STM32F1_Olimex_STM32H103_Keil/Boot/led.h](#)
LED driver header file.
- file [ARMCM3_STM32F1_Olimex_STM32H103_Keil/Boot/main.c](#)
Bootloader application source file.

5.103.1 Detailed Description

Bootloader.

5.104 Demo for Olimex STM32-H103/Keil

Preconfigured programs for the Olimex STM32-H103 and the Keil MDK for ARM IDE.

Modules

- [Bootloader](#)
Bootloader.
- [User Program](#)
User Program.

5.104.1 Detailed Description

Preconfigured programs for the Olimex STM32-H103 and the Keil MDK for ARM IDE.

For detailed getting started instructions, refer to: https://www.feaser.com/openblt/doku.php?id=manual:demos:olimex_stm32h103_keil.

5.105 User Program

User Program.

Files

- file [ARMCM3_STM32F1_Olimex_STM32H103_Keil/Prog/header.h](#)
Generic header file.
- file [ARMCM3_STM32F1_Olimex_STM32H103_Keil/Prog/led.c](#)
LED driver source file.
- file [ARMCM3_STM32F1_Olimex_STM32H103_Keil/Prog/led.h](#)
LED driver header file.
- file [ARMCM3_STM32F1_Olimex_STM32H103_Keil/Prog/main.c](#)
Demo program application source file.
- file [Demo/ARMCM3_STM32F1_Olimex_STM32H103_Keil/Prog/timer.c](#)
Timer driver source file.
- file [Demo/ARMCM3_STM32F1_Olimex_STM32H103_Keil/Prog/timer.h](#)
Timer driver header file.

5.105.1 Detailed Description

User Program.

The intention of the demo user program is two-fold. (1) To test the bootloader, you need some sort of firmware to see if you can perform a firmware update with the bootloader. This program can be used for this purpose. (2) To make firmware programmable by the bootloader, a few adjustments to the firmware are required. The demo user program serves as an example for how these adjustments can be implemented. Additional details on this subject can be found in the port specifics documentation, which is available at: <https://www.feaser.com/openblt/doku.php?id=manual:ports:armcm3-stm32>.

5.106 Bootloader

Bootloader.

Files

- file [ARMCM3_STM32F1_Olimex_STM32P103_CubeIDE/Boot/App/app.c](#)
Bootloader application source file.
- file [ARMCM3_STM32F1_Olimex_STM32P103_CubeIDE/Boot/App/app.h](#)
Bootloader application header file.
- file [ARMCM3_STM32F1_Olimex_STM32P103_CubeIDE/Boot/App/blt_conf.h](#)
Bootloader configuration header file.
- file [ARMCM3_STM32F1_Olimex_STM32P103_CubeIDE/Boot/App/hooks.c](#)
Bootloader callback source file.
- file [ARMCM3_STM32F1_Olimex_STM32P103_CubeIDE/Boot/App/led.c](#)
LED driver source file.
- file [ARMCM3_STM32F1_Olimex_STM32P103_CubeIDE/Boot/App/led.h](#)
LED driver header file.

5.106.1 Detailed Description

Bootloader.

5.107 Demo for Olimex STM32-P103/STM32CubeIDE

Preconfigured programs for the Olimex STM32-P103 and the ST STM32CubeIDE toolchain.

Modules

- [Bootloader](#)
Bootloader.
- [User Program](#)
User Program.

5.107.1 Detailed Description

Preconfigured programs for the Olimex STM32-P103 and the ST STM32CubeIDE toolchain.

For detailed getting started instructions, refer to: https://www.feaser.com/openblt/doku.php?id=manual:demos:olimex_stm32p103_cubeide.

5.108 User Program

User Program.

Files

- file [ARMCM3_STM32F1_Olimex_STM32P103_CubeIDE/Prog/App/app.c](#)
User program application source file.
- file [ARMCM3_STM32F1_Olimex_STM32P103_CubeIDE/Prog/App/app.h](#)
User program application header file.
- file [Demo/ARMCM3_STM32F1_Olimex_STM32P103_CubeIDE/Prog/App/boot.c](#)
Demo program bootloader interface source file.
- file [Demo/ARMCM3_STM32F1_Olimex_STM32P103_CubeIDE/Prog/App/boot.h](#)
Demo program bootloader interface header file.
- file [ARMCM3_STM32F1_Olimex_STM32P103_CubeIDE/Prog/App/header.h](#)
Generic header file.
- file [ARMCM3_STM32F1_Olimex_STM32P103_CubeIDE/Prog/App/led.c](#)
LED driver source file.
- file [ARMCM3_STM32F1_Olimex_STM32P103_CubeIDE/Prog/App/led.h](#)
LED driver header file.
- file [Demo/ARMCM3_STM32F1_Olimex_STM32P103_CubeIDE/Prog/App/timer.c](#)
Timer driver source file.
- file [Demo/ARMCM3_STM32F1_Olimex_STM32P103_CubeIDE/Prog/App/timer.h](#)
Timer driver header file.

5.108.1 Detailed Description

User Program.

The intention of the demo user program is two-fold. (1) To test the bootloader, you need some sort of firmware to see if you can perform a firmware update with the bootloader. This program can be used for this purpose. (2) To make firmware programmable by the bootloader, a few adjustments to the firmware are required. The demo user program serves as an example for how these adjustments can be implemented. Additional details on this subject can be found in the port specifics documentation, which is available at: https://www.feaser.com/openblt/doku.php?id=manual:ports:armcm3_stm32.

5.109 Bootloader

Bootloader.

Files

- file [ARMCM3_STM32F1_Olimex_STM32P103_GCC/Boot/blt_conf.h](#)
Bootloader configuration header file.
- file [ARMCM3_STM32F1_Olimex_STM32P103_GCC/Boot/hooks.c](#)
Bootloader callback source file.
- file [ARMCM3_STM32F1_Olimex_STM32P103_GCC/Boot/led.c](#)
LED driver source file.
- file [ARMCM3_STM32F1_Olimex_STM32P103_GCC/Boot/led.h](#)
LED driver header file.
- file [ARMCM3_STM32F1_Olimex_STM32P103_GCC/Boot/main.c](#)
Bootloader application source file.

5.109.1 Detailed Description

Bootloader.

5.110 Demo for Olimex STM32-P103/GCC

Preconfigured programs for the Olimex STM32-P103 and the GCC compiler.

Modules

- [Bootloader](#)
Bootloader.
- [User Program](#)
User Program.

5.110.1 Detailed Description

Preconfigured programs for the Olimex STM32-P103 and the GCC compiler.

For detailed getting started instructions, refer to: https://www.feaser.com/openblt/doku.php?id=manual:demos:olimex_stm32p103_gcc.

5.111 User Program

User Program.

Files

- file [Demo/ARMCM3_STM32F1_Olimex_STM32P103_GCC/Prog/boot.c](#)
Demo program bootloader interface source file.
- file [Demo/ARMCM3_STM32F1_Olimex_STM32P103_GCC/Prog/boot.h](#)
Demo program bootloader interface header file.
- file [ARMCM3_STM32F1_Olimex_STM32P103_GCC/Prog/header.h](#)
Generic header file.
- file [ARMCM3_STM32F1_Olimex_STM32P103_GCC/Prog/led.c](#)
LED driver source file.
- file [ARMCM3_STM32F1_Olimex_STM32P103_GCC/Prog/led.h](#)
LED driver header file.
- file [ARMCM3_STM32F1_Olimex_STM32P103_GCC/Prog/main.c](#)
Demo program application source file.
- file [Demo/ARMCM3_STM32F1_Olimex_STM32P103_GCC/Prog/timer.c](#)
Timer driver source file.
- file [Demo/ARMCM3_STM32F1_Olimex_STM32P103_GCC/Prog/timer.h](#)
Timer driver header file.

5.111.1 Detailed Description

User Program.

The intention of the demo user program is two-fold. (1) To test the bootloader, you need some sort of firmware to see if you can perform a firmware update with the bootloader. This program can be used for this purpose. (2) To make firmware programmable by the bootloader, a few adjustments to the firmware are required. The demo user program serves as an example for how these adjustments can be implemented. Additional details on this subject can be found in the port specifics documentation, which is available at: https://www.feaser.com/openblt/doku.php?id=manual:ports:armcm3_stm32.

5.112 Bootloader

Bootloader.

Files

- file [ARMCM3_STM32F1_Olimex_STM32P103_IAR/Boot/blt_conf.h](#)
Bootloader configuration header file.
- file [ARMCM3_STM32F1_Olimex_STM32P103_IAR/Boot/hooks.c](#)
Bootloader callback source file.
- file [ARMCM3_STM32F1_Olimex_STM32P103_IAR/Boot/led.c](#)
LED driver source file.
- file [ARMCM3_STM32F1_Olimex_STM32P103_IAR/Boot/led.h](#)
LED driver header file.
- file [ARMCM3_STM32F1_Olimex_STM32P103_IAR/Boot/main.c](#)
Bootloader application source file.

5.112.1 Detailed Description

Bootloader.

5.113 Demo for Olimex STM32-P103/IAR

Preconfigured programs for the Olimex STM32-P103 and the IAR Embedded Workbench IDE.

Modules

- [Bootloader](#)
Bootloader.
- [User Program](#)
User Program.

5.113.1 Detailed Description

Preconfigured programs for the Olimex STM32-P103 and the IAR Embedded Workbench IDE.

For detailed getting started instructions, refer to: https://www.feaser.com/openblt/doku.php?id=manual:demos:olimex_stm32p103_iar.

5.114 User Program

User Program.

Files

- file [Demo/ARMCM3_STM32F1_Olimex_STM32P103_IAR/Prog/boot.c](#)
Demo program bootloader interface source file.
- file [Demo/ARMCM3_STM32F1_Olimex_STM32P103_IAR/Prog/boot.h](#)
Demo program bootloader interface header file.
- file [ARMCM3_STM32F1_Olimex_STM32P103_IAR/Prog/header.h](#)
Generic header file.
- file [ARMCM3_STM32F1_Olimex_STM32P103_IAR/Prog/led.c](#)
LED driver source file.
- file [ARMCM3_STM32F1_Olimex_STM32P103_IAR/Prog/led.h](#)
LED driver header file.
- file [ARMCM3_STM32F1_Olimex_STM32P103_IAR/Prog/main.c](#)
Demo program application source file.
- file [Demo/ARMCM3_STM32F1_Olimex_STM32P103_IAR/Prog/timer.c](#)
Timer driver source file.
- file [Demo/ARMCM3_STM32F1_Olimex_STM32P103_IAR/Prog/timer.h](#)
Timer driver header file.

5.114.1 Detailed Description

User Program.

The intention of the demo user program is two-fold. (1) To test the bootloader, you need some sort of firmware to see if you can perform a firmware update with the bootloader. This program can be used for this purpose. (2) To make firmware programmable by the bootloader, a few adjustments to the firmware are required. The demo user program serves as an example for how these adjustments can be implemented. Additional details on this subject can be found in the port specifics documentation, which is available at: https://www.feaser.com/openblt/doku.php?id=manual:ports:armcm3_stm32.

5.115 Bootloader

Bootloader.

Files

- file [ARMCM3_STM32F1_Olimex_STM32P103_Keil/Boot/blt_conf.h](#)
Bootloader configuration header file.
- file [ARMCM3_STM32F1_Olimex_STM32P103_Keil/Boot/hooks.c](#)
Bootloader callback source file.
- file [ARMCM3_STM32F1_Olimex_STM32P103_Keil/Boot/led.c](#)
LED driver source file.
- file [ARMCM3_STM32F1_Olimex_STM32P103_Keil/Boot/led.h](#)
LED driver header file.
- file [ARMCM3_STM32F1_Olimex_STM32P103_Keil/Boot/main.c](#)
Bootloader application source file.

5.115.1 Detailed Description

Bootloader.

5.116 Demo for Olimex STM32-P103/Keil

Preconfigured programs for the Olimex STM32-P103 and the Keil MDK toolchain.

Modules

- [Bootloader](#)
Bootloader.
- [User Program](#)
User Program.

5.116.1 Detailed Description

Preconfigured programs for the Olimex STM32-P103 and the Keil MDK toolchain.

For detailed getting started instructions, refer to: https://www.feaser.com/openblt/doku.php?id=manual:demos:olimex_stm32p103_keil.

5.117 User Program

User Program.

Files

- file [Demo/ARMCM3_STM32F1_Olimex_STM32P103_Keil/Prog/boot.c](#)
Demo program bootloader interface source file.
- file [Demo/ARMCM3_STM32F1_Olimex_STM32P103_Keil/Prog/boot.h](#)
Demo program bootloader interface header file.
- file [ARMCM3_STM32F1_Olimex_STM32P103_Keil/Prog/header.h](#)
Generic header file.
- file [ARMCM3_STM32F1_Olimex_STM32P103_Keil/Prog/led.c](#)
LED driver source file.
- file [ARMCM3_STM32F1_Olimex_STM32P103_Keil/Prog/led.h](#)
LED driver header file.
- file [ARMCM3_STM32F1_Olimex_STM32P103_Keil/Prog/main.c](#)
Demo program application source file.
- file [Demo/ARMCM3_STM32F1_Olimex_STM32P103_Keil/Prog/timer.c](#)
Timer driver source file.
- file [Demo/ARMCM3_STM32F1_Olimex_STM32P103_Keil/Prog/timer.h](#)
Timer driver header file.

5.117.1 Detailed Description

User Program.

The intention of the demo user program is two-fold. (1) To test the bootloader, you need some sort of firmware to see if you can perform a firmware update with the bootloader. This program can be used for this purpose. (2) To make firmware programmable by the bootloader, a few adjustments to the firmware are required. The demo user program serves as an example for how these adjustments can be implemented. Additional details on this subject can be found in the port specifics documentation, which is available at: https://www.feaser.com/openblt/doku.php?id=manual:ports:armcm3_stm32.

5.118 Bootloader

Bootloader.

Files

- file [ARMCM3_STM32F1_Olimexino_STM32_CubeIDE/Boot/App/app.c](#)
Bootloader application source file.
- file [ARMCM3_STM32F1_Olimexino_STM32_CubeIDE/Boot/App/app.h](#)
Bootloader application header file.
- file [ARMCM3_STM32F1_Olimexino_STM32_CubeIDE/Boot/App/blt_conf.h](#)
Bootloader configuration header file.
- file [ARMCM3_STM32F1_Olimexino_STM32_CubeIDE/Boot/App/flash_layout.c](#)
Custom flash layout table source file.
- file [ARMCM3_STM32F1_Olimexino_STM32_CubeIDE/Boot/App/hooks.c](#)
Bootloader callback source file.
- file [ARMCM3_STM32F1_Olimexino_STM32_CubeIDE/Boot/App/led.c](#)
LED driver source file.
- file [ARMCM3_STM32F1_Olimexino_STM32_CubeIDE/Boot/App/led.h](#)
LED driver header file.

5.118.1 Detailed Description

Bootloader.

5.119 Demo for Olimexino-STM32/STM32CubeIDE

Preconfigured programs for the Olimexino-STM32 and the ST STM32CubeIDE toolchain.

Modules

- [Bootloader](#)
Bootloader.
- [User Program](#)
User Program.

5.119.1 Detailed Description

Preconfigured programs for the Olimexino-STM32 and the ST STM32CubeIDE toolchain.

For detailed getting started instructions, refer to: https://www.feaser.com/openblt/doku.php?id=manual:demos:olimexino_stm32_cubeide.

5.120 User Program

User Program.

Files

- file [ARMCM3_STM32F1_Olimexino_STM32_CubeIDE/Prog/App/app.c](#)
User program application source file.
- file [ARMCM3_STM32F1_Olimexino_STM32_CubeIDE/Prog/App/app.h](#)
User program application header file.
- file [Demo/ARMCM3_STM32F1_Olimexino_STM32_CubeIDE/Prog/App/boot.c](#)
Demo program bootloader interface source file.
- file [Demo/ARMCM3_STM32F1_Olimexino_STM32_CubeIDE/Prog/App/boot.h](#)
Demo program bootloader interface header file.
- file [ARMCM3_STM32F1_Olimexino_STM32_CubeIDE/Prog/App/header.h](#)
Generic header file.
- file [ARMCM3_STM32F1_Olimexino_STM32_CubeIDE/Prog/App/led.c](#)
LED driver source file.
- file [ARMCM3_STM32F1_Olimexino_STM32_CubeIDE/Prog/App/led.h](#)
LED driver header file.
- file [Demo/ARMCM3_STM32F1_Olimexino_STM32_CubeIDE/Prog/App/timer.c](#)
Timer driver source file.
- file [Demo/ARMCM3_STM32F1_Olimexino_STM32_CubeIDE/Prog/App/timer.h](#)
Timer driver header file.

5.120.1 Detailed Description

User Program.

The intention of the demo user program is two-fold. (1) To test the bootloader, you need some sort of firmware to see if you can perform a firmware update with the bootloader. This program can be used for this purpose. (2) To make firmware programmable by the bootloader, a few adjustments to the firmware are required. The demo user program serves as an example for how these adjustments can be implemented. Additional details on this subject can be found in the port specifics documentation, which is available at: https://www.feaser.com/openblt/doku.php?id=manual:ports:armcm3_stm32.

5.121 Bootloader

Bootloader.

Files

- file [ARMCM3_STM32F1_Olimexino_STM32_GCC/Boot/blt_conf.h](#)
Bootloader configuration header file.
- file [ARMCM3_STM32F1_Olimexino_STM32_GCC/Boot/flash_layout.c](#)
Custom flash layout table source file.
- file [ARMCM3_STM32F1_Olimexino_STM32_GCC/Boot/hooks.c](#)
Bootloader callback source file.
- file [ARMCM3_STM32F1_Olimexino_STM32_GCC/Boot/led.c](#)
LED driver source file.
- file [ARMCM3_STM32F1_Olimexino_STM32_GCC/Boot/led.h](#)
LED driver header file.
- file [ARMCM3_STM32F1_Olimexino_STM32_GCC/Boot/main.c](#)
Bootloader application source file.

5.121.1 Detailed Description

Bootloader.

5.122 Demo for Olimexino-STM32/GCC

Preconfigured programs for the Olimexino-STM32 and the GCC compiler.

Modules

- [Bootloader](#)
Bootloader.
- [User Program](#)
User Program.

5.122.1 Detailed Description

Preconfigured programs for the Olimexino-STM32 and the GCC compiler.

For detailed getting started instructions, refer to: https://www.feaser.com/openblt/doku.php?id=manual:demos:olimexino_stm32_gcc.

5.123 User Program

User Program.

Files

- file [Demo/ARMCM3_STM32F1_Olimexino_STM32_GCC/Prog/boot.c](#)
Demo program bootloader interface source file.
- file [Demo/ARMCM3_STM32F1_Olimexino_STM32_GCC/Prog/boot.h](#)
Demo program bootloader interface header file.
- file [ARMCM3_STM32F1_Olimexino_STM32_GCC/Prog/header.h](#)
Generic header file.
- file [ARMCM3_STM32F1_Olimexino_STM32_GCC/Prog/led.c](#)
LED driver source file.
- file [ARMCM3_STM32F1_Olimexino_STM32_GCC/Prog/led.h](#)
LED driver header file.
- file [ARMCM3_STM32F1_Olimexino_STM32_GCC/Prog/main.c](#)
Demo program application source file.
- file [Demo/ARMCM3_STM32F1_Olimexino_STM32_GCC/Prog/timer.c](#)
Timer driver source file.
- file [Demo/ARMCM3_STM32F1_Olimexino_STM32_GCC/Prog/timer.h](#)
Timer driver header file.

5.123.1 Detailed Description

User Program.

The intention of the demo user program is two-fold. (1) To test the bootloader, you need some sort of firmware to see if you can perform a firmware update with the bootloader. This program can be used for this purpose. (2) To make firmware programmable by the bootloader, a few adjustments to the firmware are required. The demo user program serves as an example for how these adjustments can be implemented. Additional details on this subject can be found in the port specifics documentation, which is available at: https://www.feaser.com/openblt/doku.php?id=manual:ports:armcm3_stm32.

5.124 Bootloader

Bootloader.

Files

- file [ARMCM3_STM32F1_Olimexino_STM32_IAR/Boot/blt_conf.h](#)
Bootloader configuration header file.
- file [ARMCM3_STM32F1_Olimexino_STM32_IAR/Boot/flash_layout.c](#)
Custom flash layout table source file.
- file [ARMCM3_STM32F1_Olimexino_STM32_IAR/Boot/hooks.c](#)
Bootloader callback source file.
- file [ARMCM3_STM32F1_Olimexino_STM32_IAR/Boot/led.c](#)
LED driver source file.
- file [ARMCM3_STM32F1_Olimexino_STM32_IAR/Boot/led.h](#)
LED driver header file.
- file [ARMCM3_STM32F1_Olimexino_STM32_IAR/Boot/main.c](#)
Bootloader application source file.

5.124.1 Detailed Description

Bootloader.

5.125 Demo for Olimexino-STM32/IAR

Preconfigured programs for the Olimexino-STM32 and the IAR Embedded Workbench.

Modules

- [Bootloader](#)
Bootloader.
- [User Program](#)
User Program.

5.125.1 Detailed Description

Preconfigured programs for the Olimexino-STM32 and the IAR Embedded Workbench.

For detailed getting started instructions, refer to: https://www.feaser.com/openblt/doku.php?id=manual:demos:olimexino_stm32_iar.

5.126 User Program

User Program.

Files

- file [Demo/ARMCM3_STM32F1_Olimexino_STM32_IAR/Prog/boot.c](#)
Demo program bootloader interface source file.
- file [Demo/ARMCM3_STM32F1_Olimexino_STM32_IAR/Prog/boot.h](#)
Demo program bootloader interface header file.
- file [ARMCM3_STM32F1_Olimexino_STM32_IAR/Prog/header.h](#)
Generic header file.
- file [ARMCM3_STM32F1_Olimexino_STM32_IAR/Prog/led.c](#)
LED driver source file.
- file [ARMCM3_STM32F1_Olimexino_STM32_IAR/Prog/led.h](#)
LED driver header file.
- file [ARMCM3_STM32F1_Olimexino_STM32_IAR/Prog/main.c](#)
Demo program application source file.
- file [Demo/ARMCM3_STM32F1_Olimexino_STM32_IAR/Prog/timer.c](#)
Timer driver source file.
- file [Demo/ARMCM3_STM32F1_Olimexino_STM32_IAR/Prog/timer.h](#)
Timer driver header file.

5.126.1 Detailed Description

User Program.

The intention of the demo user program is two-fold. (1) To test the bootloader, you need some sort of firmware to see if you can perform a firmware update with the bootloader. This program can be used for this purpose. (2) To make firmware programmable by the bootloader, a few adjustments to the firmware are required. The demo user program serves as an example for how these adjustments can be implemented. Additional details on this subject can be found in the port specifics documentation, which is available at: https://www.feaser.com/openblt/doku.php?id=manual:ports:armcm3_stm32.

5.127 Bootloader

Bootloader.

Files

- file [ARMCM3_STM32F1_Olimexino_STM32_Keil/Boot/blt_conf.h](#)
Bootloader configuration header file.
- file [ARMCM3_STM32F1_Olimexino_STM32_Keil/Boot/flash_layout.c](#)
Custom flash layout table source file.
- file [ARMCM3_STM32F1_Olimexino_STM32_Keil/Boot/hooks.c](#)
Bootloader callback source file.
- file [ARMCM3_STM32F1_Olimexino_STM32_Keil/Boot/led.c](#)
LED driver source file.
- file [ARMCM3_STM32F1_Olimexino_STM32_Keil/Boot/led.h](#)
LED driver header file.
- file [ARMCM3_STM32F1_Olimexino_STM32_Keil/Boot/main.c](#)
Bootloader application source file.

5.127.1 Detailed Description

Bootloader.

5.128 Demo for Olimexino-STM32/Keil

Preconfigured programs for the Olimexino-STM32 and the Keil MDK toolchain.

Modules

- [Bootloader](#)
Bootloader.
- [User Program](#)
User Program.

5.128.1 Detailed Description

Preconfigured programs for the Olimexino-STM32 and the Keil MDK toolchain.

For detailed getting started instructions, refer to: https://www.feaser.com/openblt/doku.php?id=manual:demos:olimexino_stm32_keil.

5.129 User Program

User Program.

Files

- file [Demo/ARMCM3_STM32F1_Olimexino_STM32_Keil/Prog/boot.c](#)
Demo program bootloader interface source file.
- file [Demo/ARMCM3_STM32F1_Olimexino_STM32_Keil/Prog/boot.h](#)
Demo program bootloader interface header file.
- file [ARMCM3_STM32F1_Olimexino_STM32_Keil/Prog/header.h](#)
Generic header file.
- file [ARMCM3_STM32F1_Olimexino_STM32_Keil/Prog/led.c](#)
LED driver source file.
- file [ARMCM3_STM32F1_Olimexino_STM32_Keil/Prog/led.h](#)
LED driver header file.
- file [ARMCM3_STM32F1_Olimexino_STM32_Keil/Prog/main.c](#)
Demo program application source file.
- file [Demo/ARMCM3_STM32F1_Olimexino_STM32_Keil/Prog/timer.c](#)
Timer driver source file.
- file [Demo/ARMCM3_STM32F1_Olimexino_STM32_Keil/Prog/timer.h](#)
Timer driver header file.

5.129.1 Detailed Description

User Program.

The intention of the demo user program is two-fold. (1) To test the bootloader, you need some sort of firmware to see if you can perform a firmware update with the bootloader. This program can be used for this purpose. (2) To make firmware programmable by the bootloader, a few adjustments to the firmware are required. The demo user program serves as an example for how these adjustments can be implemented. Additional details on this subject can be found in the port specifics documentation, which is available at: https://www.feaser.com/openblt/doku.php?id=manual:ports:armcm3_stm32.

5.130 Bootloader

Bootloader.

Files

- file [ARMCM3_STM32F2_Olimex_STM32P207_CubeIDE/Boot/App/app.c](#)
Bootloader application source file.
- file [ARMCM3_STM32F2_Olimex_STM32P207_CubeIDE/Boot/App/app.h](#)
Bootloader application header file.
- file [ARMCM3_STM32F2_Olimex_STM32P207_CubeIDE/Boot/App/blt_conf.h](#)
Bootloader configuration header file.
- file [ARMCM3_STM32F2_Olimex_STM32P207_CubeIDE/Boot/App/hooks.c](#)
Bootloader callback source file.
- file [ARMCM3_STM32F2_Olimex_STM32P207_CubeIDE/Boot/App/led.c](#)
LED driver source file.
- file [ARMCM3_STM32F2_Olimex_STM32P207_CubeIDE/Boot/App/led.h](#)
LED driver header file.

5.130.1 Detailed Description

Bootloader.

5.131 Demo for Olimex STM32-P207/STM32CubeIDE

Preconfigured programs for the Olimex STM32-P207 and the STM32CubeIDE toolchain.

Modules

- [Bootloader](#)
Bootloader.
- [User Program](#)
User Program.

5.131.1 Detailed Description

Preconfigured programs for the Olimex STM32-P207 and the STM32CubeIDE toolchain.

For detailed getting started instructions, refer to: https://www.feaser.com/openblt/doku.php?id=manual:demos:olimex_stm32p207_cubeide.

5.132 User Program

User Program.

Files

- file [ARMCM3_STM32F2_Olimex_STM32P207_CubeIDE/Prog/App/app.c](#)
User program application source file.
- file [ARMCM3_STM32F2_Olimex_STM32P207_CubeIDE/Prog/App/app.h](#)
User program application header file.
- file [Demo/ARMCM3_STM32F2_Olimex_STM32P207_CubeIDE/Prog/App/boot.c](#)
Demo program bootloader interface source file.
- file [Demo/ARMCM3_STM32F2_Olimex_STM32P207_CubeIDE/Prog/App/boot.h](#)
Demo program bootloader interface header file.
- file [ARMCM3_STM32F2_Olimex_STM32P207_CubeIDE/Prog/App/header.h](#)
Generic header file.
- file [ARMCM3_STM32F2_Olimex_STM32P207_CubeIDE/Prog/App/led.c](#)
LED driver source file.
- file [ARMCM3_STM32F2_Olimex_STM32P207_CubeIDE/Prog/App/led.h](#)
LED driver header file.
- file [Demo/ARMCM3_STM32F2_Olimex_STM32P207_CubeIDE/Prog/App/timer.c](#)
Timer driver source file.
- file [Demo/ARMCM3_STM32F2_Olimex_STM32P207_CubeIDE/Prog/App/timer.h](#)
Timer driver header file.

5.132.1 Detailed Description

User Program.

The intention of the demo user program is two-fold. (1) To test the bootloader, you need some sort of firmware to see if you can perform a firmware update with the bootloader. This program can be used for this purpose. (2) To make firmware programmable by the bootloader, a few adjustments to the firmware are required. The demo user program serves as an example for how these adjustments can be implemented. Additional details on this subject can be found in the port specifics documentation, which is available at: https://www.feaser.com/openblt/doku.php?id=manual:ports:armcm3_stm32f2.

5.133 Bootloader

Bootloader.

Files

- file [ARMCM3_STM32F2_Olimex_STM32P207_GCC/Boot/blt_conf.h](#)
Bootloader configuration header file.
- file [ARMCM3_STM32F2_Olimex_STM32P207_GCC/Boot/hooks.c](#)
Bootloader callback source file.
- file [ARMCM3_STM32F2_Olimex_STM32P207_GCC/Boot/led.c](#)
LED driver source file.
- file [ARMCM3_STM32F2_Olimex_STM32P207_GCC/Boot/led.h](#)
LED driver header file.
- file [ARMCM3_STM32F2_Olimex_STM32P207_GCC/Boot/main.c](#)
Bootloader application source file.

5.133.1 Detailed Description

Bootloader.

5.134 Demo for Olimex STM32-P207/GCC

Preconfigured programs for the Olimex STM32-P207 and the GCC compiler.

Modules

- [Bootloader](#)
Bootloader.
- [User Program](#)
User Program.

5.134.1 Detailed Description

Preconfigured programs for the Olimex STM32-P207 and the GCC compiler.

For detailed getting started instructions, refer to: https://www.feaser.com/openblt/doku.php?id=manual:demos:olimex_stm32p207_gcc.

5.135 User Program

User Program.

Files

- file [Demo/ARMCM3_STM32F2_Olimex_STM32P207_GCC/Prog/boot.c](#)
Demo program bootloader interface source file.
- file [Demo/ARMCM3_STM32F2_Olimex_STM32P207_GCC/Prog/boot.h](#)
Demo program bootloader interface header file.
- file [ARMCM3_STM32F2_Olimex_STM32P207_GCC/Prog/header.h](#)
Generic header file.
- file [ARMCM3_STM32F2_Olimex_STM32P207_GCC/Prog/led.c](#)
LED driver source file.
- file [ARMCM3_STM32F2_Olimex_STM32P207_GCC/Prog/led.h](#)
LED driver header file.
- file [ARMCM3_STM32F2_Olimex_STM32P207_GCC/Prog/main.c](#)
Demo program application source file.
- file [Demo/ARMCM3_STM32F2_Olimex_STM32P207_GCC/Prog/timer.c](#)
Timer driver source file.
- file [Demo/ARMCM3_STM32F2_Olimex_STM32P207_GCC/Prog/timer.h](#)
Timer driver header file.

5.135.1 Detailed Description

User Program.

The intention of the demo user program is two-fold. (1) To test the bootloader, you need some sort of firmware to see if you can perform a firmware update with the bootloader. This program can be used for this purpose. (2) To make firmware programmable by the bootloader, a few adjustments to the firmware are required. The demo user program serves as an example for how these adjustments can be implemented. Additional details on this subject can be found in the port specifics documentation, which is available at: https://www.feaser.com/openblt/doku.php?id=manual:ports:armcm3_stm32f2.

5.136 Bootloader

Bootloader.

Files

- file [ARMCM3_STM32F2_Olimex_STM32P207_IAR/Boot/blt_conf.h](#)
Bootloader configuration header file.
- file [ARMCM3_STM32F2_Olimex_STM32P207_IAR/Boot/hooks.c](#)
Bootloader callback source file.
- file [ARMCM3_STM32F2_Olimex_STM32P207_IAR/Boot/led.c](#)
LED driver source file.
- file [ARMCM3_STM32F2_Olimex_STM32P207_IAR/Boot/led.h](#)
LED driver header file.
- file [ARMCM3_STM32F2_Olimex_STM32P207_IAR/Boot/main.c](#)
Bootloader application source file.

5.136.1 Detailed Description

Bootloader.

5.137 Demo for Olimex STM32-P207/IAR

Preconfigured programs for the Olimex STM32-P207 and the IAR Embedded Workbench.

Modules

- [Bootloader](#)
Bootloader.
- [User Program](#)
User Program.

5.137.1 Detailed Description

Preconfigured programs for the Olimex STM32-P207 and the IAR Embedded Workbench.

For detailed getting started instructions, refer to: https://www.feaser.com/openblt/doku.php?id=manual:demos:olimex_stm32p207_iar.

5.138 User Program

User Program.

Files

- file [Demo/ARMCM3_STM32F2_Olimex_STM32P207_IAR/Prog/boot.c](#)
Demo program bootloader interface source file.
- file [Demo/ARMCM3_STM32F2_Olimex_STM32P207_IAR/Prog/boot.h](#)
Demo program bootloader interface header file.
- file [ARMCM3_STM32F2_Olimex_STM32P207_IAR/Prog/header.h](#)
Generic header file.
- file [ARMCM3_STM32F2_Olimex_STM32P207_IAR/Prog/led.c](#)
LED driver source file.
- file [ARMCM3_STM32F2_Olimex_STM32P207_IAR/Prog/led.h](#)
LED driver header file.
- file [ARMCM3_STM32F2_Olimex_STM32P207_IAR/Prog/main.c](#)
Demo program application source file.
- file [Demo/ARMCM3_STM32F2_Olimex_STM32P207_IAR/Prog/timer.c](#)
Timer driver source file.
- file [Demo/ARMCM3_STM32F2_Olimex_STM32P207_IAR/Prog/timer.h](#)
Timer driver header file.

5.138.1 Detailed Description

User Program.

The intention of the demo user program is two-fold. (1) To test the bootloader, you need some sort of firmware to see if you can perform a firmware update with the bootloader. This program can be used for this purpose. (2) To make firmware programmable by the bootloader, a few adjustments to the firmware are required. The demo user program serves as an example for how these adjustments can be implemented. Additional details on this subject can be found in the port specifics documentation, which is available at: https://www.feaser.com/openblt/doku.php?id=manual:ports:armcm3_stm32f2.

5.139 Bootloader

Bootloader.

Files

- file [ARMCM3_STM32F2_Olimex_STM32P207_Keil/Boot/blt_conf.h](#)
Bootloader configuration header file.
- file [ARMCM3_STM32F2_Olimex_STM32P207_Keil/Boot/hooks.c](#)
Bootloader callback source file.
- file [ARMCM3_STM32F2_Olimex_STM32P207_Keil/Boot/led.c](#)
LED driver source file.
- file [ARMCM3_STM32F2_Olimex_STM32P207_Keil/Boot/led.h](#)
LED driver header file.
- file [ARMCM3_STM32F2_Olimex_STM32P207_Keil/Boot/main.c](#)
Bootloader application source file.

5.139.1 Detailed Description

Bootloader.

5.140 Demo for Olimex STM32-P207/Keil

Preconfigured programs for the Olimex STM32-P207 and the Keil MDK toolchain.

Modules

- [Bootloader](#)
Bootloader.
- [User Program](#)
User Program.

5.140.1 Detailed Description

Preconfigured programs for the Olimex STM32-P207 and the Keil MDK toolchain.

For detailed getting started instructions, refer to: https://www.feaser.com/openblt/doku.php?id=manual:demos:olimex_stm32p207_keil.

5.141 User Program

User Program.

Files

- file [Demo/ARMCM3_STM32F2_Olimex_STM32P207_Keil/Prog/boot.c](#)
Demo program bootloader interface source file.
- file [Demo/ARMCM3_STM32F2_Olimex_STM32P207_Keil/Prog/boot.h](#)
Demo program bootloader interface header file.
- file [ARMCM3_STM32F2_Olimex_STM32P207_Keil/Prog/header.h](#)
Generic header file.
- file [ARMCM3_STM32F2_Olimex_STM32P207_Keil/Prog/led.c](#)
LED driver source file.
- file [ARMCM3_STM32F2_Olimex_STM32P207_Keil/Prog/led.h](#)
LED driver header file.
- file [ARMCM3_STM32F2_Olimex_STM32P207_Keil/Prog/main.c](#)
Demo program application source file.
- file [Demo/ARMCM3_STM32F2_Olimex_STM32P207_Keil/Prog/timer.c](#)
Timer driver source file.
- file [Demo/ARMCM3_STM32F2_Olimex_STM32P207_Keil/Prog/timer.h](#)
Timer driver header file.

5.141.1 Detailed Description

User Program.

The intention of the demo user program is two-fold. (1) To test the bootloader, you need some sort of firmware to see if you can perform a firmware update with the bootloader. This program can be used for this purpose. (2) To make firmware programmable by the bootloader, a few adjustments to the firmware are required. The demo user program serves as an example for how these adjustments can be implemented. Additional details on this subject can be found in the port specifics documentation, which is available at: https://www.feaser.com/openblt/doku.php?id=manual:ports:armcm3_stm32f2.

5.142 Bootloader

Bootloader.

Files

- file [ARMCM4_S32K14_S32K144EVB_GCC/Boot/blt_conf.h](#)
Bootloader configuration header file.
- file [ARMCM4_S32K14_S32K144EVB_GCC/Boot/hooks.c](#)
Bootloader callback source file.
- file [ARMCM4_S32K14_S32K144EVB_GCC/Boot/led.c](#)
LED driver source file.
- file [ARMCM4_S32K14_S32K144EVB_GCC/Boot/led.h](#)
LED driver header file.
- file [ARMCM4_S32K14_S32K144EVB_GCC/Boot/main.c](#)
Bootloader application source file.

5.142.1 Detailed Description

Bootloader.

5.143 Demo for S32K144EVB/GCC

Preconfigured programs for the NXP S32K144EVB board and the S32 Design Studio development environment, which is based on the ARM GCC toolchain.

Modules

- [Bootloader](#)
Bootloader.
- [User Program](#)
User Program.

5.143.1 Detailed Description

Preconfigured programs for the NXP S32K144EVB board and the S32 Design Studio development environment, which is based on the ARM GCC toolchain.

5.144 User Program

User Program.

Files

- file [Demo/ARMCM4_S32K14_S32K144EVB_GCC/Prog/boot.c](#)
Demo program bootloader interface source file.
- file [Demo/ARMCM4_S32K14_S32K144EVB_GCC/Prog/boot.h](#)
Demo program bootloader interface header file.
- file [ARMCM4_S32K14_S32K144EVB_GCC/Prog/header.h](#)
Generic header file.
- file [ARMCM4_S32K14_S32K144EVB_GCC/Prog/led.c](#)
LED driver source file.
- file [ARMCM4_S32K14_S32K144EVB_GCC/Prog/led.h](#)
LED driver header file.
- file [ARMCM4_S32K14_S32K144EVB_GCC/Prog/main.c](#)
Demo program application source file.
- file [Demo/ARMCM4_S32K14_S32K144EVB_GCC/Prog/timer.c](#)
Timer driver source file.
- file [Demo/ARMCM4_S32K14_S32K144EVB_GCC/Prog/timer.h](#)
Timer driver header file.

5.144.1 Detailed Description

User Program.

The intention of the demo user program is two-fold. (1) To test the bootloader, you need some sort of firmware to see if you can perform a firmware update with the bootloader. This program can be used for this purpose. (2) To make firmware programmable by the bootloader, a few adjustments to the firmware are required. The demo user program serves as an example for how these adjustments can be implemented. This demo user program is a template that can be used as a starting point for creating your own demo user program.

5.145 Bootloader

Bootloader.

Files

- file [ARMCM4_S32K14_S32K144EVB_IAR/Boot/blt_conf.h](#)
Bootloader configuration header file.
- file [ARMCM4_S32K14_S32K144EVB_IAR/Boot/hooks.c](#)
Bootloader callback source file.
- file [ARMCM4_S32K14_S32K144EVB_IAR/Boot/led.c](#)
LED driver source file.
- file [ARMCM4_S32K14_S32K144EVB_IAR/Boot/led.h](#)
LED driver header file.
- file [ARMCM4_S32K14_S32K144EVB_IAR/Boot/main.c](#)
Bootloader application source file.

5.145.1 Detailed Description

Bootloader.

5.146 Demo for S32K144EVB/IAR

Preconfigured programs for the NXP S32K144EVB board and the IAR Embedded Workbench for ARM.

Modules

- [Bootloader](#)
Bootloader.
- [User Program](#)
User Program.

5.146.1 Detailed Description

Preconfigured programs for the NXP S32K144EVB board and the IAR Embedded Workbench for ARM.

5.147 User Program

User Program.

Files

- file [Demo/ARMCM4_S32K14_S32K144EVB_IAR/Prog/boot.c](#)
Demo program bootloader interface source file.
- file [Demo/ARMCM4_S32K14_S32K144EVB_IAR/Prog/boot.h](#)
Demo program bootloader interface header file.
- file [ARMCM4_S32K14_S32K144EVB_IAR/Prog/header.h](#)
Generic header file.
- file [ARMCM4_S32K14_S32K144EVB_IAR/Prog/led.c](#)
LED driver source file.
- file [ARMCM4_S32K14_S32K144EVB_IAR/Prog/led.h](#)
LED driver header file.
- file [ARMCM4_S32K14_S32K144EVB_IAR/Prog/main.c](#)
Demo program application source file.
- file [Demo/ARMCM4_S32K14_S32K144EVB_IAR/Prog/timer.c](#)
Timer driver source file.
- file [Demo/ARMCM4_S32K14_S32K144EVB_IAR/Prog/timer.h](#)
Timer driver header file.

5.147.1 Detailed Description

User Program.

The intention of the demo user program is two-fold. (1) To test the bootloader, you need some sort of firmware to see if you can perform a firmware update with the bootloader. This program can be used for this purpose. (2) To make firmware programmable by the bootloader, a few adjustments to the firmware are required. The demo user program serves as an example for how these adjustments can be implemented. This demo user program is a template that can be used as a starting point for creating your own demo user program.

5.148 Bootloader

Bootloader.

Files

- file [ARMCM4_STM32F3_Discovery_F303VC_CubeIDE/Boot/App/app.c](#)
Bootloader application source file.
- file [ARMCM4_STM32F3_Discovery_F303VC_CubeIDE/Boot/App/app.h](#)
Bootloader application header file.
- file [ARMCM4_STM32F3_Discovery_F303VC_CubeIDE/Boot/App/blt_conf.h](#)
Bootloader configuration header file.
- file [ARMCM4_STM32F3_Discovery_F303VC_CubeIDE/Boot/App/flash_layout.c](#)
Custom flash layout table source file.
- file [ARMCM4_STM32F3_Discovery_F303VC_CubeIDE/Boot/App/hooks.c](#)
Bootloader callback source file.
- file [ARMCM4_STM32F3_Discovery_F303VC_CubeIDE/Boot/App/led.c](#)
LED driver source file.
- file [ARMCM4_STM32F3_Discovery_F303VC_CubeIDE/Boot/App/led.h](#)
LED driver header file.

5.148.1 Detailed Description

Bootloader.

5.149 Demo for STM32F3-Discovery/STM32CubeIDE

Preconfigured programs for the STM32F3-Discovery board and the ST STM32CubeIDE toolchain.

Modules

- [Bootloader](#)
Bootloader.
- [User Program](#)
User Program.

5.149.1 Detailed Description

Preconfigured programs for the STM32F3-Discovery board and the ST STM32CubeIDE toolchain.

For detailed getting started instructions, refer to: https://www.feaser.com/openblt/doku.php?id=manual:demos:stm32f3_discovery_cubeide.

5.150 User Program

User Program.

Files

- file [ARMCM4_STM32F3_Discovery_F303VC_CubeIDE/Prog/App/app.c](#)
User program application source file.
- file [ARMCM4_STM32F3_Discovery_F303VC_CubeIDE/Prog/App/app.h](#)
User program application header file.
- file [ARMCM4_STM32F3_Discovery_F303VC_CubeIDE/Prog/App/header.h](#)
Generic header file.
- file [ARMCM4_STM32F3_Discovery_F303VC_CubeIDE/Prog/App/led.c](#)
LED driver source file.
- file [ARMCM4_STM32F3_Discovery_F303VC_CubeIDE/Prog/App/led.h](#)
LED driver header file.
- file [Demo/ARMCM4_STM32F3_Discovery_F303VC_CubeIDE/Prog/App/timer.c](#)
Timer driver source file.
- file [Demo/ARMCM4_STM32F3_Discovery_F303VC_CubeIDE/Prog/App/timer.h](#)
Timer driver header file.

5.150.1 Detailed Description

User Program.

The intention of the demo user program is two-fold. (1) To test the bootloader, you need some sort of firmware to see if you can perform a firmware update with the bootloader. This program can be used for this purpose. (2) To make firmware programmable by the bootloader, a few adjustments to the firmware are required. The demo user program serves as an example for how these adjustments can be implemented. Additional details on this subject can be found in the port specifics documentation, which is available at: https://www.feaser.com/openblt/doku.php?id=manual:ports:armcm4_stm32f3.

5.151 Bootloader

Bootloader.

Files

- file [ARMCM4_STM32F3_Discovery_F303VC_GCC/Boot/blt_conf.h](#)
Bootloader configuration header file.
- file [ARMCM4_STM32F3_Discovery_F303VC_GCC/Boot/flash_layout.c](#)
Custom flash layout table source file.
- file [ARMCM4_STM32F3_Discovery_F303VC_GCC/Boot/hooks.c](#)
Bootloader callback source file.
- file [ARMCM4_STM32F3_Discovery_F303VC_GCC/Boot/led.c](#)
LED driver source file.
- file [ARMCM4_STM32F3_Discovery_F303VC_GCC/Boot/led.h](#)
LED driver header file.
- file [ARMCM4_STM32F3_Discovery_F303VC_GCC/Boot/main.c](#)
Bootloader application source file.

5.151.1 Detailed Description

Bootloader.

5.152 Demo for STM32F3-Discovery/GCC

Preconfigured programs for the STM32F3-Discovery board and the ARM GNU Embedded toolchain.

Modules

- [Bootloader](#)
Bootloader.
- [User Program](#)
User Program.

5.152.1 Detailed Description

Preconfigured programs for the STM32F3-Discovery board and the ARM GNU Embedded toolchain.

For detailed getting started instructions, refer to: https://www.feaser.com/openblt/doku.php?id=manual:demos:stm32f3_discovery_gcc.

5.153 User Program

User Program.

Files

- file [ARMCM4_STM32F3_Discovery_F303VC_GCC/Prog/header.h](#)
Generic header file.
- file [ARMCM4_STM32F3_Discovery_F303VC_GCC/Prog/led.c](#)
LED driver source file.
- file [ARMCM4_STM32F3_Discovery_F303VC_GCC/Prog/led.h](#)
LED driver header file.
- file [ARMCM4_STM32F3_Discovery_F303VC_GCC/Prog/main.c](#)
Demo program application source file.
- file [Demo/ARMCM4_STM32F3_Discovery_F303VC_GCC/Prog/timer.c](#)
Timer driver source file.
- file [Demo/ARMCM4_STM32F3_Discovery_F303VC_GCC/Prog/timer.h](#)
Timer driver header file.

5.153.1 Detailed Description

User Program.

The intention of the demo user program is two-fold. (1) To test the bootloader, you need some sort of firmware to see if you can perform a firmware update with the bootloader. This program can be used for this purpose. (2) To make firmware programmable by the bootloader, a few adjustments to the firmware are required. The demo user program serves as an example for how these adjustments can be implemented. Additional details on this subject can be found in the port specifics documentation, which is available at: https://www.feaser.com/openblt/doku.php?id=manual:ports:armcm4_stm32f3.

5.154 Bootloader

Bootloader.

Files

- file [ARMCM4_STM32F3_Discovery_F303VC_IAR/Boot/blt_conf.h](#)
Bootloader configuration header file.
- file [ARMCM4_STM32F3_Discovery_F303VC_IAR/Boot/flash_layout.c](#)
Custom flash layout table source file.
- file [ARMCM4_STM32F3_Discovery_F303VC_IAR/Boot/hooks.c](#)
Bootloader callback source file.
- file [ARMCM4_STM32F3_Discovery_F303VC_IAR/Boot/led.c](#)
LED driver source file.
- file [ARMCM4_STM32F3_Discovery_F303VC_IAR/Boot/led.h](#)
LED driver header file.
- file [ARMCM4_STM32F3_Discovery_F303VC_IAR/Boot/main.c](#)
Bootloader application source file.

5.154.1 Detailed Description

Bootloader.

5.155 Demo for STM32F3-Discovery/IAR

Preconfigured programs for the STM32F3-Discovery board and the IAR Embedded Workbench.

Modules

- [Bootloader](#)
Bootloader.
- [User Program](#)
User Program.

5.155.1 Detailed Description

Preconfigured programs for the STM32F3-Discovery board and the IAR Embedded Workbench.

For detailed getting started instructions, refer to: https://www.feaser.com/openblt/doku.php?id=manual:demos:stm32f3_discovery_iar.

5.156 User Program

User Program.

Files

- file [ARMCM4_STM32F3_Discovery_F303VC_IAR/Prog/header.h](#)
Generic header file.
- file [ARMCM4_STM32F3_Discovery_F303VC_IAR/Prog/led.c](#)
LED driver source file.
- file [ARMCM4_STM32F3_Discovery_F303VC_IAR/Prog/led.h](#)
LED driver header file.
- file [ARMCM4_STM32F3_Discovery_F303VC_IAR/Prog/main.c](#)
Demo program application source file.
- file [Demo/ARMCM4_STM32F3_Discovery_F303VC_IAR/Prog/timer.c](#)
Timer driver source file.
- file [Demo/ARMCM4_STM32F3_Discovery_F303VC_IAR/Prog/timer.h](#)
Timer driver header file.

5.156.1 Detailed Description

User Program.

The intention of the demo user program is two-fold. (1) To test the bootloader, you need some sort of firmware to see if you can perform a firmware update with the bootloader. This program can be used for this purpose. (2) To make firmware programmable by the bootloader, a few adjustments to the firmware are required. The demo user program serves as an example for how these adjustments can be implemented. Additional details on this subject can be found in the port specifics documentation, which is available at: https://www.feaser.com/openblt/doku.php?id=manual:ports:armcm4_stm32f3.

5.157 Bootloader

Bootloader.

Files

- file [ARMCM4_STM32F3_Discovery_F303VC_Keil/Boot/blt_conf.h](#)
Bootloader configuration header file.
- file [ARMCM4_STM32F3_Discovery_F303VC_Keil/Boot/flash_layout.c](#)
Custom flash layout table source file.
- file [ARMCM4_STM32F3_Discovery_F303VC_Keil/Boot/hooks.c](#)
Bootloader callback source file.
- file [ARMCM4_STM32F3_Discovery_F303VC_Keil/Boot/led.c](#)
LED driver source file.
- file [ARMCM4_STM32F3_Discovery_F303VC_Keil/Boot/led.h](#)
LED driver header file.
- file [ARMCM4_STM32F3_Discovery_F303VC_Keil/Boot/main.c](#)
Bootloader application source file.

5.157.1 Detailed Description

Bootloader.

5.158 Demo for STM32F3-Discovery/Keil

Preconfigured programs for the STM32F3-Discovery board and the Keil MDK for ARM IDE.

Modules

- [Bootloader](#)
Bootloader.
- [User Program](#)
User Program.

5.158.1 Detailed Description

Preconfigured programs for the STM32F3-Discovery board and the Keil MDK for ARM IDE.

For detailed getting started instructions, refer to: https://www.feaser.com/openblt/doku.php?id=manual:demos:stm32f3_discovery_keil.

5.159 User Program

User Program.

Files

- file [ARMCM4_STM32F3_Discovery_F303VC_Keil/Prog/header.h](#)
Generic header file.
- file [ARMCM4_STM32F3_Discovery_F303VC_Keil/Prog/led.c](#)
LED driver source file.
- file [ARMCM4_STM32F3_Discovery_F303VC_Keil/Prog/led.h](#)
LED driver header file.
- file [ARMCM4_STM32F3_Discovery_F303VC_Keil/Prog/main.c](#)
Demo program application source file.
- file [Demo/ARMCM4_STM32F3_Discovery_F303VC_Keil/Prog/timer.c](#)
Timer driver source file.
- file [Demo/ARMCM4_STM32F3_Discovery_F303VC_Keil/Prog/timer.h](#)
Timer driver header file.

5.159.1 Detailed Description

User Program.

The intention of the demo user program is two-fold. (1) To test the bootloader, you need some sort of firmware to see if you can perform a firmware update with the bootloader. This program can be used for this purpose. (2) To make firmware programmable by the bootloader, a few adjustments to the firmware are required. The demo user program serves as an example for how these adjustments can be implemented. Additional details on this subject can be found in the port specifics documentation, which is available at: https://www.feaser.com/openblt/doku.php?id=manual:ports:armcm4_stm32f3.

5.160 Bootloader

Bootloader.

Files

- file [ARMCM4_STM32F3_Nucleo_F303K8_CubeIDE/Boot/App/app.c](#)
Bootloader application source file.
- file [ARMCM4_STM32F3_Nucleo_F303K8_CubeIDE/Boot/App/app.h](#)
Bootloader application header file.
- file [ARMCM4_STM32F3_Nucleo_F303K8_CubeIDE/Boot/App/blt_conf.h](#)
Bootloader configuration header file.
- file [ARMCM4_STM32F3_Nucleo_F303K8_CubeIDE/Boot/App/hooks.c](#)
Bootloader callback source file.
- file [ARMCM4_STM32F3_Nucleo_F303K8_CubeIDE/Boot/App/led.c](#)
LED driver source file.
- file [ARMCM4_STM32F3_Nucleo_F303K8_CubeIDE/Boot/App/led.h](#)
LED driver header file.

5.160.1 Detailed Description

Bootloader.

5.161 Demo for Nucleo-F303K8/STM32CubeIDE

Preconfigured programs for the Nucleo-F303K8 and the ST STM32CubeIDE toolchain.

Modules

- [Bootloader](#)
Bootloader.
- [User Program](#)
User Program.

5.161.1 Detailed Description

Preconfigured programs for the Nucleo-F303K8 and the ST STM32CubeIDE toolchain.

For detailed getting started instructions, refer to: https://www.feaser.com/openblt/doku.php?id=manual:demos:nucleo_f303k8_cubeide.

5.162 User Program

User Program.

Files

- file [ARMCM4_STM32F3_Nucleo_F303K8_CubeIDE/Prog/App/app.c](#)
User program application source file.
- file [ARMCM4_STM32F3_Nucleo_F303K8_CubeIDE/Prog/App/app.h](#)
User program application header file.
- file [Demo/ARMCM4_STM32F3_Nucleo_F303K8_CubeIDE/Prog/App/boot.c](#)
Demo program bootloader interface source file.
- file [Demo/ARMCM4_STM32F3_Nucleo_F303K8_CubeIDE/Prog/App/boot.h](#)
Demo program bootloader interface header file.
- file [ARMCM4_STM32F3_Nucleo_F303K8_CubeIDE/Prog/App/header.h](#)
Generic header file.
- file [ARMCM4_STM32F3_Nucleo_F303K8_CubeIDE/Prog/App/led.c](#)
LED driver source file.
- file [ARMCM4_STM32F3_Nucleo_F303K8_CubeIDE/Prog/App/led.h](#)
LED driver header file.
- file [Demo/ARMCM4_STM32F3_Nucleo_F303K8_CubeIDE/Prog/App/timer.c](#)
Timer driver source file.
- file [Demo/ARMCM4_STM32F3_Nucleo_F303K8_CubeIDE/Prog/App/timer.h](#)
Timer driver header file.

5.162.1 Detailed Description

User Program.

The intention of the demo user program is two-fold. (1) To test the bootloader, you need some sort of firmware to see if you can perform a firmware update with the bootloader. This program can be used for this purpose. (2) To make firmware programmable by the bootloader, a few adjustments to the firmware are required. The demo user program serves as an example for how these adjustments can be implemented. Additional details on this subject can be found in the port specifics documentation, which is available at: https://www.feaser.com/openblt/doku.php?id=manual:ports:armcm4_stm32f3.

5.163 Bootloader

Bootloader.

Files

- file [ARMCM4_STM32F3_Nucleo_F303K8_GCC/Boot/blt_conf.h](#)
Bootloader configuration header file.
- file [ARMCM4_STM32F3_Nucleo_F303K8_GCC/Boot/hooks.c](#)
Bootloader callback source file.
- file [ARMCM4_STM32F3_Nucleo_F303K8_GCC/Boot/led.c](#)
LED driver source file.
- file [ARMCM4_STM32F3_Nucleo_F303K8_GCC/Boot/led.h](#)
LED driver header file.
- file [ARMCM4_STM32F3_Nucleo_F303K8_GCC/Boot/main.c](#)
Bootloader application source file.

5.163.1 Detailed Description

Bootloader.

5.164 Demo for Nucleo-F303K8/GCC

Preconfigured programs for the Nucleo-F303K8 and the ARM GNU Embedded toolchain.

Modules

- [Bootloader](#)
Bootloader.
- [User Program](#)
User Program.

5.164.1 Detailed Description

Preconfigured programs for the Nucleo-F303K8 and the ARM GNU Embedded toolchain.

For detailed getting started instructions, refer to: https://www.feaser.com/openblt/doku.php?id=manual:demos:nucleo_f303k8_gcc.

5.165 User Program

User Program.

Files

- file [Demo/ARMCM4_STM32F3_Nucleo_F303K8_GCC/Prog/boot.c](#)
Demo program bootloader interface source file.
- file [Demo/ARMCM4_STM32F3_Nucleo_F303K8_GCC/Prog/boot.h](#)
Demo program bootloader interface header file.
- file [ARMCM4_STM32F3_Nucleo_F303K8_GCC/Prog/header.h](#)
Generic header file.
- file [ARMCM4_STM32F3_Nucleo_F303K8_GCC/Prog/led.c](#)
LED driver source file.
- file [ARMCM4_STM32F3_Nucleo_F303K8_GCC/Prog/led.h](#)
LED driver header file.
- file [ARMCM4_STM32F3_Nucleo_F303K8_GCC/Prog/main.c](#)
Demo program application source file.
- file [Demo/ARMCM4_STM32F3_Nucleo_F303K8_GCC/Prog/timer.c](#)
Timer driver source file.
- file [Demo/ARMCM4_STM32F3_Nucleo_F303K8_GCC/Prog/timer.h](#)
Timer driver header file.

5.165.1 Detailed Description

User Program.

The intention of the demo user program is two-fold. (1) To test the bootloader, you need some sort of firmware to see if you can perform a firmware update with the bootloader. This program can be used for this purpose. (2) To make firmware programmable by the bootloader, a few adjustments to the firmware are required. The demo user program serves as an example for how these adjustments can be implemented. Additional details on this subject can be found in the port specifics documentation, which is available at: https://www.feaser.com/openblt/doku.php?id=manual:ports:armcm4_stm32f3.

5.166 Bootloader

Bootloader.

Files

- file [ARMCM4_STM32F3_Nucleo_F303K8_IAR/Boot/blt_conf.h](#)
Bootloader configuration header file.
- file [ARMCM4_STM32F3_Nucleo_F303K8_IAR/Boot/hooks.c](#)
Bootloader callback source file.
- file [ARMCM4_STM32F3_Nucleo_F303K8_IAR/Boot/led.c](#)
LED driver source file.
- file [ARMCM4_STM32F3_Nucleo_F303K8_IAR/Boot/led.h](#)
LED driver header file.
- file [ARMCM4_STM32F3_Nucleo_F303K8_IAR/Boot/main.c](#)
Bootloader application source file.

5.166.1 Detailed Description

Bootloader.

5.167 Demo for Nucleo-F303K8/IAR

Preconfigured programs for the Nucleo-F303K8 and the IAR Embedded Workbench IDE.

Modules

- [Bootloader](#)
Bootloader.
- [User Program](#)
User Program.

5.167.1 Detailed Description

Preconfigured programs for the Nucleo-F303K8 and the IAR Embedded Workbench IDE.

For detailed getting started instructions, refer to: https://www.feaser.com/openblt/doku.php?id=manual:demos:nucleo_f303k8_iar.

5.168 User Program

User Program.

Files

- file [Demo/ARMCM4_STM32F3_Nucleo_F303K8_IAR/Prog/boot.c](#)
Demo program bootloader interface source file.
- file [Demo/ARMCM4_STM32F3_Nucleo_F303K8_IAR/Prog/boot.h](#)
Demo program bootloader interface header file.
- file [ARMCM4_STM32F3_Nucleo_F303K8_IAR/Prog/header.h](#)
Generic header file.
- file [ARMCM4_STM32F3_Nucleo_F303K8_IAR/Prog/led.c](#)
LED driver source file.
- file [ARMCM4_STM32F3_Nucleo_F303K8_IAR/Prog/led.h](#)
LED driver header file.
- file [ARMCM4_STM32F3_Nucleo_F303K8_IAR/Prog/main.c](#)
Demo program application source file.
- file [Demo/ARMCM4_STM32F3_Nucleo_F303K8_IAR/Prog/timer.c](#)
Timer driver source file.
- file [Demo/ARMCM4_STM32F3_Nucleo_F303K8_IAR/Prog/timer.h](#)
Timer driver header file.

5.168.1 Detailed Description

User Program.

The intention of the demo user program is two-fold. (1) To test the bootloader, you need some sort of firmware to see if you can perform a firmware update with the bootloader. This program can be used for this purpose. (2) To make firmware programmable by the bootloader, a few adjustments to the firmware are required. The demo user program serves as an example for how these adjustments can be implemented. Additional details on this subject can be found in the port specifics documentation, which is available at: https://www.feaser.com/openblt/doku.php?id=manual:ports:armcm4_stm32f3.

5.169 Bootloader

Bootloader.

Files

- file [ARMCM4_STM32F3_Nucleo_F303K8_Keil/Boot/blt_conf.h](#)
Bootloader configuration header file.
- file [ARMCM4_STM32F3_Nucleo_F303K8_Keil/Boot/hooks.c](#)
Bootloader callback source file.
- file [ARMCM4_STM32F3_Nucleo_F303K8_Keil/Boot/led.c](#)
LED driver source file.
- file [ARMCM4_STM32F3_Nucleo_F303K8_Keil/Boot/led.h](#)
LED driver header file.
- file [ARMCM4_STM32F3_Nucleo_F303K8_Keil/Boot/main.c](#)
Bootloader application source file.

5.169.1 Detailed Description

Bootloader.

5.170 Demo for Nucleo-F303K8/Keil

Preconfigured programs for the Nucleo-F303K8 and the Keil MDK toolchain.

Modules

- [Bootloader](#)
Bootloader.
- [User Program](#)
User Program.

5.170.1 Detailed Description

Preconfigured programs for the Nucleo-F303K8 and the Keil MDK toolchain.

For detailed getting started instructions, refer to: https://www.feaser.com/openblt/doku.php?id=manual:demos:nucleo_f303k8_keil.

5.171 User Program

User Program.

Files

- file [Demo/ARMCM4_STM32F3_Nucleo_F303K8_Keil/Prog/boot.c](#)
Demo program bootloader interface source file.
- file [Demo/ARMCM4_STM32F3_Nucleo_F303K8_Keil/Prog/boot.h](#)
Demo program bootloader interface header file.
- file [ARMCM4_STM32F3_Nucleo_F303K8_Keil/Prog/header.h](#)
Generic header file.
- file [ARMCM4_STM32F3_Nucleo_F303K8_Keil/Prog/led.c](#)
LED driver source file.
- file [ARMCM4_STM32F3_Nucleo_F303K8_Keil/Prog/led.h](#)
LED driver header file.
- file [ARMCM4_STM32F3_Nucleo_F303K8_Keil/Prog/main.c](#)
Demo program application source file.
- file [Demo/ARMCM4_STM32F3_Nucleo_F303K8_Keil/Prog/timer.c](#)
Timer driver source file.
- file [Demo/ARMCM4_STM32F3_Nucleo_F303K8_Keil/Prog/timer.h](#)
Timer driver header file.

5.171.1 Detailed Description

User Program.

The intention of the demo user program is two-fold. (1) To test the bootloader, you need some sort of firmware to see if you can perform a firmware update with the bootloader. This program can be used for this purpose. (2) To make firmware programmable by the bootloader, a few adjustments to the firmware are required. The demo user program serves as an example for how these adjustments can be implemented. Additional details on this subject can be found in the port specifics documentation, which is available at: https://www.feaser.com/openblt/doku.php?id=manual:ports:armcm4_stm32f3.

5.172 Bootloader

Bootloader.

Files

- file [ARMCM4_STM32F4_Nucleo_F429ZI_CubeIDE/Boot/App/app.c](#)
Bootloader application source file.
- file [ARMCM4_STM32F4_Nucleo_F429ZI_CubeIDE/Boot/App/app.h](#)
Bootloader application header file.
- file [ARMCM4_STM32F4_Nucleo_F429ZI_CubeIDE/Boot/App/blt_conf.h](#)
Bootloader configuration header file.
- file [ARMCM4_STM32F4_Nucleo_F429ZI_CubeIDE/Boot/App/hooks.c](#)
Bootloader callback source file.
- file [ARMCM4_STM32F4_Nucleo_F429ZI_CubeIDE/Boot/App/led.c](#)
LED driver source file.
- file [ARMCM4_STM32F4_Nucleo_F429ZI_CubeIDE/Boot/App/led.h](#)
LED driver header file.
- file [ARMCM4_STM32F4_Nucleo_F429ZI_CubeIDE/Boot/App/shared_params.c](#)
Shared RAM parameters source file.
- file [ARMCM4_STM32F4_Nucleo_F429ZI_CubeIDE/Boot/App/shared_params.h](#)
Shared RAM parameters header file.

5.172.1 Detailed Description

Bootloader.

5.173 Demo for Nucleo-F429ZI/STM32CubeIDE

Preconfigured programs for the Nucleo-F429ZI board and the ST STM32CubeIDE toolchain.

Modules

- [Bootloader](#)
Bootloader.
- [User Program](#)
User Program.

5.173.1 Detailed Description

Preconfigured programs for the Nucleo-F429ZI board and the ST STM32CubeIDE toolchain.

For detailed getting started instructions, refer to: https://www.feaser.com/openblt/doku.php?id=manual:demos:nucleo_f429zi_cubeide.

5.174 User Program

User Program.

Files

- file [ARMCM4_STM32F4_Nucleo_F429ZI_CubeIDE/Prog/App/app.c](#)
User program application source file.
- file [ARMCM4_STM32F4_Nucleo_F429ZI_CubeIDE/Prog/App/app.h](#)
User program application header file.
- file [Demo/ARMCM4_STM32F4_Nucleo_F429ZI_CubeIDE/Prog/App/boot.c](#)
Demo program bootloader interface source file.
- file [Demo/ARMCM4_STM32F4_Nucleo_F429ZI_CubeIDE/Prog/App/boot.h](#)
Demo program bootloader interface header file.
- file [ARMCM4_STM32F4_Nucleo_F429ZI_CubeIDE/Prog/App/header.h](#)
Generic header file.
- file [ARMCM4_STM32F4_Nucleo_F429ZI_CubeIDE/Prog/App/led.c](#)
LED driver source file.
- file [ARMCM4_STM32F4_Nucleo_F429ZI_CubeIDE/Prog/App/led.h](#)
LED driver header file.
- file [Demo/ARMCM4_STM32F4_Nucleo_F429ZI_CubeIDE/Prog/App/net.c](#)
Network application for the uIP TCP/IP stack.
- file [Demo/ARMCM4_STM32F4_Nucleo_F429ZI_CubeIDE/Prog/App/net.h](#)
Network application for the uIP TCP/IP stack.
- file [ARMCM4_STM32F4_Nucleo_F429ZI_CubeIDE/Prog/App/shared_params.c](#)
Shared RAM parameters source file.
- file [ARMCM4_STM32F4_Nucleo_F429ZI_CubeIDE/Prog/App/shared_params.h](#)
Shared RAM parameters header file.
- file [Demo/ARMCM4_STM32F4_Nucleo_F429ZI_CubeIDE/Prog/App/timer.c](#)
Timer driver source file.
- file [Demo/ARMCM4_STM32F4_Nucleo_F429ZI_CubeIDE/Prog/App/timer.h](#)
Timer driver header file.
- file [ARMCM4_STM32F4_Nucleo_F429ZI_GCC/Prog/header.h](#)
Generic header file.

5.174.1 Detailed Description

User Program.

The intention of the demo user program is two-fold. (1) To test the bootloader, you need some sort of firmware to see if you can perform a firmware update with the bootloader. This program can be used for this purpose. (2) To make firmware programmable by the bootloader, a few adjustments to the firmware are required. The demo user program serves as an example for how these adjustments can be implemented. Additional details on this subject can be found in the port specifics documentation, which is available at: https://www.feaser.com/openblt/doku.php?id=manual:ports:armcm4_stm32.

5.175 Bootloader

Bootloader.

Files

- file [ARMCM4_STM32F4_Nucleo_F429ZI_GCC/Boot/blt_conf.h](#)
Bootloader configuration header file.
- file [ARMCM4_STM32F4_Nucleo_F429ZI_GCC/Boot/hooks.c](#)
Bootloader callback source file.
- file [ARMCM4_STM32F4_Nucleo_F429ZI_GCC/Boot/led.h](#)
LED driver header file.
- file [ARMCM4_STM32F4_Nucleo_F429ZI_GCC/Boot/main.c](#)
Bootloader application source file.
- file [ARMCM4_STM32F4_Nucleo_F429ZI_GCC/Boot/shared_params.c](#)
Shared RAM parameters source file.

5.175.1 Detailed Description

Bootloader.

5.176 Demo for Nucleo-F429ZI/GCC

Preconfigured programs for the Nucleo-F429ZI board and the GNU ARM GCC compiler.

Modules

- [Bootloader](#)
Bootloader.
- [User Program](#)
User Program.

5.176.1 Detailed Description

Preconfigured programs for the Nucleo-F429ZI board and the GNU ARM GCC compiler.

For detailed getting started instructions, refer to: https://www.feaser.com/openblt/doku.php?id=manual:demos:nucleo_f429zi_gcc.

5.177 User Program

User Program.

Files

- file [Demo/ARMCM4_STM32F4_Nucleo_F429ZI_GCC/Prog/boot.c](#)
Demo program bootloader interface source file.
- file [Demo/ARMCM4_STM32F4_Nucleo_F429ZI_GCC/Prog/boot.h](#)
Demo program bootloader interface header file.
- file [ARMCM4_STM32F4_Nucleo_F429ZI_GCC/Prog/led.c](#)
LED driver source file.
- file [ARMCM4_STM32F4_Nucleo_F429ZI_GCC/Prog/main.c](#)
Demo program application source file.
- file [Demo/ARMCM4_STM32F4_Nucleo_F429ZI_GCC/Prog/net.c](#)
Network application for the uIP TCP/IP stack.
- file [Demo/ARMCM4_STM32F4_Nucleo_F429ZI_GCC/Prog/net.h](#)
Network application for the uIP TCP/IP stack.
- file [ARMCM4_STM32F4_Nucleo_F429ZI_GCC/Prog/shared_params.c](#)
Shared RAM parameters source file.
- file [ARMCM4_STM32F4_Nucleo_F429ZI_GCC/Prog/shared_params.h](#)
Shared RAM parameters header file.
- file [Demo/ARMCM4_STM32F4_Nucleo_F429ZI_GCC/Prog/timer.c](#)
Timer driver source file.
- file [Demo/ARMCM4_STM32F4_Nucleo_F429ZI_GCC/Prog/timer.h](#)
Timer driver header file.

5.177.1 Detailed Description

User Program.

The intention of the demo user program is two-fold. (1) To test the bootloader, you need some sort of firmware to see if you can perform a firmware update with the bootloader. This program can be used for this purpose. (2) To make firmware programmable by the bootloader, a few adjustments to the firmware are required. The demo user program serves as an example for how these adjustments can be implemented. Additional details on this subject can be found in the port specifics documentation, which is available at: https://www.feaser.com/openblt/doku.php?id=manual:ports:armcm4_stm32.

5.178 Bootloader

Bootloader.

Files

- file [ARMCM4_STM32F4_Nucleo_F429ZI_IAR/Boot/blt_conf.h](#)
Bootloader configuration header file.
- file [ARMCM4_STM32F4_Nucleo_F429ZI_IAR/Boot/hooks.c](#)
Bootloader callback source file.
- file [ARMCM4_STM32F4_Nucleo_F429ZI_IAR/Boot/led.h](#)
LED driver header file.
- file [ARMCM4_STM32F4_Nucleo_F429ZI_IAR/Boot/main.c](#)
Bootloader application source file.
- file [ARMCM4_STM32F4_Nucleo_F429ZI_IAR/Boot/shared_params.c](#)
Shared RAM parameters source file.

5.178.1 Detailed Description

Bootloader.

5.179 Demo for Nucleo-F429ZI/IAR

Preconfigured programs for the Nucleo-F429ZI board and the IAR Embedded Workbench IDE.

Modules

- [Bootloader](#)
Bootloader.
- [User Program](#)
User Program.

5.179.1 Detailed Description

Preconfigured programs for the Nucleo-F429ZI board and the IAR Embedded Workbench IDE.

For detailed getting started instructions, refer to: https://www.feaser.com/openblt/doku.php?id=manual:demos:nucleo_f429zi_iar.

5.180 User Program

User Program.

Files

- file [Demo/ARMCM4_STM32F4_Nucleo_F429ZI_IAR/Prog/boot.c](#)
Demo program bootloader interface source file.
- file [Demo/ARMCM4_STM32F4_Nucleo_F429ZI_IAR/Prog/boot.h](#)
Demo program bootloader interface header file.
- file [ARMCM4_STM32F4_Nucleo_F429ZI_IAR/Prog/header.h](#)
Generic header file.
- file [ARMCM4_STM32F4_Nucleo_F429ZI_IAR/Prog/led.c](#)
LED driver source file.
- file [ARMCM4_STM32F4_Nucleo_F429ZI_IAR/Prog/main.c](#)
Demo program application source file.
- file [Demo/ARMCM4_STM32F4_Nucleo_F429ZI_IAR/Prog/net.c](#)
Network application for the uIP TCP/IP stack.
- file [Demo/ARMCM4_STM32F4_Nucleo_F429ZI_IAR/Prog/net.h](#)
Network application for the uIP TCP/IP stack.
- file [ARMCM4_STM32F4_Nucleo_F429ZI_IAR/Prog/shared_params.c](#)
Shared RAM parameters source file.
- file [ARMCM4_STM32F4_Nucleo_F429ZI_IAR/Prog/shared_params.h](#)
Shared RAM parameters header file.
- file [Demo/ARMCM4_STM32F4_Nucleo_F429ZI_IAR/Prog/timer.c](#)
Timer driver source file.
- file [Demo/ARMCM4_STM32F4_Nucleo_F429ZI_IAR/Prog/timer.h](#)
Timer driver header file.

5.180.1 Detailed Description

User Program.

The intention of the demo user program is two-fold. (1) To test the bootloader, you need some sort of firmware to see if you can perform a firmware update with the bootloader. This program can be used for this purpose. (2) To make firmware programmable by the bootloader, a few adjustments to the firmware are required. The demo user program serves as an example for how these adjustments can be implemented. Additional details on this subject can be found in the port specifics documentation, which is available at: https://www.feaser.com/openblt/doku.php?id=manual:ports:armcm4_stm32.

5.181 Bootloader

Bootloader.

Files

- file [ARMCM4_STM32F4_Nucleo_F429ZI_Keil/Boot/blt_conf.h](#)
Bootloader configuration header file.
- file [ARMCM4_STM32F4_Nucleo_F429ZI_Keil/Boot/hooks.c](#)
Bootloader callback source file.
- file [ARMCM4_STM32F4_Nucleo_F429ZI_Keil/Boot/led.h](#)
LED driver header file.
- file [ARMCM4_STM32F4_Nucleo_F429ZI_Keil/Boot/main.c](#)
Bootloader application source file.
- file [ARMCM4_STM32F4_Nucleo_F429ZI_Keil/Boot/shared_params.c](#)
Shared RAM parameters source file.

5.181.1 Detailed Description

Bootloader.

5.182 Demo for Nucleo-F429ZI/Keil

Preconfigured programs for the Nucleo-F429ZI board and the Keil MDK toolchain.

Modules

- [Bootloader](#)
Bootloader.
- [User Program](#)
User Program.

5.182.1 Detailed Description

Preconfigured programs for the Nucleo-F429ZI board and the Keil MDK toolchain.

For detailed getting started instructions, refer to: https://www.feaser.com/openblt/doku.php?id=manual:demos:nucleo_f429zi_keil.

5.183 User Program

User Program.

Files

- file [Demo/ARMCM4_STM32F4_Nucleo_F429ZI_Keil/Prog/boot.c](#)
Demo program bootloader interface source file.
- file [Demo/ARMCM4_STM32F4_Nucleo_F429ZI_Keil/Prog/boot.h](#)
Demo program bootloader interface header file.
- file [ARMCM4_STM32F4_Nucleo_F429ZI_Keil/Prog/header.h](#)
Generic header file.
- file [ARMCM4_STM32F4_Nucleo_F429ZI_Keil/Prog/led.c](#)
LED driver source file.
- file [ARMCM4_STM32F4_Nucleo_F429ZI_Keil/Prog/main.c](#)
Demo program application source file.
- file [Demo/ARMCM4_STM32F4_Nucleo_F429ZI_Keil/Prog/net.c](#)
Network application for the uIP TCP/IP stack.
- file [Demo/ARMCM4_STM32F4_Nucleo_F429ZI_Keil/Prog/net.h](#)
Network application for the uIP TCP/IP stack.
- file [ARMCM4_STM32F4_Nucleo_F429ZI_Keil/Prog/shared_params.c](#)
Shared RAM parameters source file.
- file [ARMCM4_STM32F4_Nucleo_F429ZI_Keil/Prog/shared_params.h](#)
Shared RAM parameters header file.
- file [Demo/ARMCM4_STM32F4_Nucleo_F429ZI_Keil/Prog/timer.c](#)
Timer driver source file.
- file [Demo/ARMCM4_STM32F4_Nucleo_F429ZI_Keil/Prog/timer.h](#)
Timer driver header file.

5.183.1 Detailed Description

User Program.

The intention of the demo user program is two-fold. (1) To test the bootloader, you need some sort of firmware to see if you can perform a firmware update with the bootloader. This program can be used for this purpose. (2) To make firmware programmable by the bootloader, a few adjustments to the firmware are required. The demo user program serves as an example for how these adjustments can be implemented. Additional details on this subject can be found in the port specifics documentation, which is available at: https://www.feaser.com/openblt/doku.php?id=manual:ports:armcm4_stm32.

5.184 Bootloader

Bootloader.

Files

- file [ARMCM4_STM32F4_Olimex_STM32P405_CubeIDE/Boot/App/app.c](#)
Bootloader application source file.
- file [ARMCM4_STM32F4_Olimex_STM32P405_CubeIDE/Boot/App/app.h](#)
Bootloader application header file.
- file [ARMCM4_STM32F4_Olimex_STM32P405_CubeIDE/Boot/App/blt_conf.h](#)
Bootloader configuration header file.
- file [ARMCM4_STM32F4_Olimex_STM32P405_CubeIDE/Boot/App/hooks.c](#)
Bootloader callback source file.
- file [ARMCM4_STM32F4_Olimex_STM32P405_CubeIDE/Boot/App/led.c](#)
LED driver source file.
- file [ARMCM4_STM32F4_Olimex_STM32P405_CubeIDE/Boot/App/led.h](#)
LED driver header file.

5.184.1 Detailed Description

Bootloader.

5.185 Demo for Olimex STM32-P405/STM32CubeIDE

Preconfigured programs for the Olimex STM32-P405 and the ST STM32CubeIDE toolchain.

Modules

- [Bootloader](#)
Bootloader.
- [User Program](#)
User Program.

5.185.1 Detailed Description

Preconfigured programs for the Olimex STM32-P405 and the ST STM32CubeIDE toolchain.

For detailed getting started instructions, refer to: https://www.feaser.com/openblt/doku.php?id=manual:demos:olimex_stm32p405_cubeide.

5.186 User Program

User Program.

Files

- file [ARMCM4_STM32F4_Olimex_STM32P405_CubeIDE/Prog/App/app.c](#)
User program application source file.
- file [ARMCM4_STM32F4_Olimex_STM32P405_CubeIDE/Prog/App/app.h](#)
User program application header file.
- file [Demo/ARMCM4_STM32F4_Olimex_STM32P405_CubeIDE/Prog/App/boot.c](#)
Demo program bootloader interface source file.
- file [Demo/ARMCM4_STM32F4_Olimex_STM32P405_CubeIDE/Prog/App/boot.h](#)
Demo program bootloader interface header file.
- file [ARMCM4_STM32F4_Olimex_STM32P405_CubeIDE/Prog/App/header.h](#)
Generic header file.
- file [ARMCM4_STM32F4_Olimex_STM32P405_CubeIDE/Prog/App/led.c](#)
LED driver source file.
- file [ARMCM4_STM32F4_Olimex_STM32P405_CubeIDE/Prog/App/led.h](#)
LED driver header file.
- file [Demo/ARMCM4_STM32F4_Olimex_STM32P405_CubeIDE/Prog/App/timer.c](#)
Timer driver source file.
- file [Demo/ARMCM4_STM32F4_Olimex_STM32P405_CubeIDE/Prog/App/timer.h](#)
Timer driver header file.

5.186.1 Detailed Description

User Program.

The intention of the demo user program is two-fold. (1) To test the bootloader, you need some sort of firmware to see if you can perform a firmware update with the bootloader. This program can be used for this purpose. (2) To make firmware programmable by the bootloader, a few adjustments to the firmware are required. The demo user program serves as an example for how these adjustments can be implemented. Additional details on this subject can be found in the port specifics documentation, which is available at: https://www.feaser.com/openblt/doku.php?id=manual:ports:armcm4_stm32.

5.187 Bootloader

Bootloader.

Files

- file [ARMCM4_STM32F4_Olimex_STM32P405_GCC/Boot/blt_conf.h](#)
Bootloader configuration header file.
- file [ARMCM4_STM32F4_Olimex_STM32P405_GCC/Boot/hooks.c](#)
Bootloader callback source file.
- file [ARMCM4_STM32F4_Olimex_STM32P405_GCC/Boot/led.c](#)
LED driver source file.
- file [ARMCM4_STM32F4_Olimex_STM32P405_GCC/Boot/led.h](#)
LED driver header file.
- file [ARMCM4_STM32F4_Olimex_STM32P405_GCC/Boot/main.c](#)
Bootloader application source file.

5.187.1 Detailed Description

Bootloader.

5.188 Demo for Olimex STM32-P405/GCC

Preconfigured programs for the Olimex STM32-P405 and the GCC compiler.

Modules

- [Bootloader](#)
Bootloader.
- [User Program](#)
User Program.

5.188.1 Detailed Description

Preconfigured programs for the Olimex STM32-P405 and the GCC compiler.

For detailed getting started instructions, refer to: https://www.feaser.com/openblt/doku.php?id=manual:demos:olimex_stm32p405_gcc.

5.189 User Program

User Program.

Files

- file [Demo/ARMCM4_STM32F4_Olimex_STM32P405_GCC/Prog/boot.c](#)
Demo program bootloader interface source file.
- file [Demo/ARMCM4_STM32F4_Olimex_STM32P405_GCC/Prog/boot.h](#)
Demo program bootloader interface header file.
- file [ARMCM4_STM32F4_Olimex_STM32P405_GCC/Prog/header.h](#)
Generic header file.
- file [ARMCM4_STM32F4_Olimex_STM32P405_GCC/Prog/led.c](#)
LED driver source file.
- file [ARMCM4_STM32F4_Olimex_STM32P405_GCC/Prog/led.h](#)
LED driver header file.
- file [ARMCM4_STM32F4_Olimex_STM32P405_GCC/Prog/main.c](#)
Demo program application source file.
- file [Demo/ARMCM4_STM32F4_Olimex_STM32P405_GCC/Prog/timer.c](#)
Timer driver source file.
- file [Demo/ARMCM4_STM32F4_Olimex_STM32P405_GCC/Prog/timer.h](#)
Timer driver header file.

5.189.1 Detailed Description

User Program.

The intention of the demo user program is two-fold. (1) To test the bootloader, you need some sort of firmware to see if you can perform a firmware update with the bootloader. This program can be used for this purpose. (2) To make firmware programmable by the bootloader, a few adjustments to the firmware are required. The demo user program serves as an example for how these adjustments can be implemented. Additional details on this subject can be found in the port specifics documentation, which is available at: https://www.feaser.com/openblt/doku.php?id=manual:ports:armcm4_stm32.

5.190 Bootloader

Bootloader.

Files

- file [ARMCM4_STM32F4_Olimex_STM32P405_IAR/Boot/blt_conf.h](#)
Bootloader configuration header file.
- file [ARMCM4_STM32F4_Olimex_STM32P405_IAR/Boot/hooks.c](#)
Bootloader callback source file.
- file [ARMCM4_STM32F4_Olimex_STM32P405_IAR/Boot/led.c](#)
LED driver source file.
- file [ARMCM4_STM32F4_Olimex_STM32P405_IAR/Boot/led.h](#)
LED driver header file.
- file [ARMCM4_STM32F4_Olimex_STM32P405_IAR/Boot/main.c](#)
Bootloader application source file.

5.190.1 Detailed Description

Bootloader.

5.191 Demo for Olimex STM32-P405/IAR

Preconfigured programs for the Olimex STM32-P405 and the IAR Embedded Workbench IDE.

Modules

- [Bootloader](#)
Bootloader.
- [User Program](#)
User Program.

5.191.1 Detailed Description

Preconfigured programs for the Olimex STM32-P405 and the IAR Embedded Workbench IDE.

For detailed getting started instructions, refer to: https://www.feaser.com/openblt/doku.php?id=manual:demos:olimex_stm32p405_iar.

5.192 User Program

User Program.

Files

- file [Demo/ARMCM4_STM32F4_Olimex_STM32P405_IAR/Prog/boot.c](#)
Demo program bootloader interface source file.
- file [Demo/ARMCM4_STM32F4_Olimex_STM32P405_IAR/Prog/boot.h](#)
Demo program bootloader interface header file.
- file [ARMCM4_STM32F4_Olimex_STM32P405_IAR/Prog/header.h](#)
Generic header file.
- file [ARMCM4_STM32F4_Olimex_STM32P405_IAR/Prog/led.c](#)
LED driver source file.
- file [ARMCM4_STM32F4_Olimex_STM32P405_IAR/Prog/led.h](#)
LED driver header file.
- file [ARMCM4_STM32F4_Olimex_STM32P405_IAR/Prog/main.c](#)
Demo program application source file.
- file [Demo/ARMCM4_STM32F4_Olimex_STM32P405_IAR/Prog/timer.c](#)
Timer driver source file.
- file [Demo/ARMCM4_STM32F4_Olimex_STM32P405_IAR/Prog/timer.h](#)
Timer driver header file.

5.192.1 Detailed Description

User Program.

The intention of the demo user program is two-fold. (1) To test the bootloader, you need some sort of firmware to see if you can perform a firmware update with the bootloader. This program can be used for this purpose. (2) To make firmware programmable by the bootloader, a few adjustments to the firmware are required. The demo user program serves as an example for how these adjustments can be implemented. Additional details on this subject can be found in the port specifics documentation, which is available at: https://www.feaser.com/openblt/doku.php?id=manual:ports:armcm4_stm32.

5.193 Bootloader

Bootloader.

Files

- file [ARMCM4_STM32F4_Olimex_STM32P405_Keil/Boot/blt_conf.h](#)
Bootloader configuration header file.
- file [ARMCM4_STM32F4_Olimex_STM32P405_Keil/Boot/hooks.c](#)
Bootloader callback source file.
- file [ARMCM4_STM32F4_Olimex_STM32P405_Keil/Boot/led.c](#)
LED driver source file.
- file [ARMCM4_STM32F4_Olimex_STM32P405_Keil/Boot/led.h](#)
LED driver header file.
- file [ARMCM4_STM32F4_Olimex_STM32P405_Keil/Boot/main.c](#)
Bootloader application source file.

5.193.1 Detailed Description

Bootloader.

5.194 Demo for Olimex STM32-P405/Keil

Preconfigured programs for the Olimex STM32-P405 and the Keil MDK for ARM IDE.

Modules

- [Bootloader](#)
Bootloader.
- [User Program](#)
User Program.

5.194.1 Detailed Description

Preconfigured programs for the Olimex STM32-P405 and the Keil MDK for ARM IDE.

For detailed getting started instructions, refer to: https://www.feaser.com/openblt/doku.php?id=manual:demos:olimex_stm32p405_keil.

5.195 User Program

User Program.

Files

- file [Demo/ARMCM4_STM32F4_Olimex_STM32P405_Keil/Prog/boot.c](#)
Demo program bootloader interface source file.
- file [Demo/ARMCM4_STM32F4_Olimex_STM32P405_Keil/Prog/boot.h](#)
Demo program bootloader interface header file.
- file [ARMCM4_STM32F4_Olimex_STM32P405_Keil/Prog/header.h](#)
Generic header file.
- file [ARMCM4_STM32F4_Olimex_STM32P405_Keil/Prog/led.c](#)
LED driver source file.
- file [ARMCM4_STM32F4_Olimex_STM32P405_Keil/Prog/led.h](#)
LED driver header file.
- file [ARMCM4_STM32F4_Olimex_STM32P405_Keil/Prog/main.c](#)
Demo program application source file.
- file [Demo/ARMCM4_STM32F4_Olimex_STM32P405_Keil/Prog/timer.c](#)
Timer driver source file.
- file [Demo/ARMCM4_STM32F4_Olimex_STM32P405_Keil/Prog/timer.h](#)
Timer driver header file.

5.195.1 Detailed Description

User Program.

The intention of the demo user program is two-fold. (1) To test the bootloader, you need some sort of firmware to see if you can perform a firmware update with the bootloader. This program can be used for this purpose. (2) To make firmware programmable by the bootloader, a few adjustments to the firmware are required. The demo user program serves as an example for how these adjustments can be implemented. Additional details on this subject can be found in the port specifics documentation, which is available at: https://www.feaser.com/openblt/doku.php?id=manual:ports:armcm4_stm32.

5.196 Bootloader

Bootloader.

Files

- file [ARMCM4_STM32L4_Nucleo_L476RG_CubeIDE/Boot/App/app.c](#)
Bootloader application source file.
- file [ARMCM4_STM32L4_Nucleo_L476RG_CubeIDE/Boot/App/app.h](#)
Bootloader application header file.
- file [ARMCM4_STM32L4_Nucleo_L476RG_CubeIDE/Boot/App/blt_conf.h](#)
Bootloader configuration header file.
- file [ARMCM4_STM32L4_Nucleo_L476RG_CubeIDE/Boot/App/hooks.c](#)
Bootloader callback source file.
- file [ARMCM4_STM32L4_Nucleo_L476RG_CubeIDE/Boot/App/led.c](#)
LED driver source file.
- file [ARMCM4_STM32L4_Nucleo_L476RG_CubeIDE/Boot/App/led.h](#)
LED driver header file.

5.196.1 Detailed Description

Bootloader.

5.197 Demo for Nucleo-L476RG/STM32CubeIDE

Preconfigured programs for the Nucleo-L476RG and the STM32CubeIDE toolchain.

Modules

- [Bootloader](#)
Bootloader.
- [User Program](#)
User Program.

5.197.1 Detailed Description

Preconfigured programs for the Nucleo-L476RG and the STM32CubeIDE toolchain.

For detailed getting started instructions, refer to: https://www.feaser.com/openblt/doku.php?id=manual:demos:nucleo_l476rg_cubeide.

5.198 User Program

User Program.

Files

- file [ARMCM4_STM32L4_Nucleo_L476RG_CubeIDE/Prog/App/app.c](#)
User program application source file.
- file [ARMCM4_STM32L4_Nucleo_L476RG_CubeIDE/Prog/App/app.h](#)
User program application header file.
- file [Demo/ARMCM4_STM32L4_Nucleo_L476RG_CubeIDE/Prog/App/boot.c](#)
Demo program bootloader interface source file.
- file [Demo/ARMCM4_STM32L4_Nucleo_L476RG_CubeIDE/Prog/App/boot.h](#)
Demo program bootloader interface header file.
- file [ARMCM4_STM32L4_Nucleo_L476RG_CubeIDE/Prog/App/header.h](#)
Generic header file.
- file [ARMCM4_STM32L4_Nucleo_L476RG_CubeIDE/Prog/App/led.c](#)
LED driver source file.
- file [ARMCM4_STM32L4_Nucleo_L476RG_CubeIDE/Prog/App/led.h](#)
LED driver header file.
- file [Demo/ARMCM4_STM32L4_Nucleo_L476RG_CubeIDE/Prog/App/timer.c](#)
Timer driver source file.
- file [Demo/ARMCM4_STM32L4_Nucleo_L476RG_CubeIDE/Prog/App/timer.h](#)
Timer driver header file.

5.198.1 Detailed Description

User Program.

The intention of the demo user program is two-fold. (1) To test the bootloader, you need some sort of firmware to see if you can perform a firmware update with the bootloader. This program can be used for this purpose. (2) To make firmware programmable by the bootloader, a few adjustments to the firmware are required. The demo user program serves as an example for how these adjustments can be implemented. Additional details on this subject can be found in the port specifics documentation, which is available at: https://www.feaser.com/openblt/doku.php?id=manual:ports:armcm4_stm32l4.

5.199 Bootloader

Bootloader.

Files

- file [ARMCM4_STM32L4_Nucleo_L476RG_GCC/Boot/blt_conf.h](#)
Bootloader configuration header file.
- file [ARMCM4_STM32L4_Nucleo_L476RG_GCC/Boot/hooks.c](#)
Bootloader callback source file.
- file [ARMCM4_STM32L4_Nucleo_L476RG_GCC/Boot/led.c](#)
LED driver source file.
- file [ARMCM4_STM32L4_Nucleo_L476RG_GCC/Boot/led.h](#)
LED driver header file.
- file [ARMCM4_STM32L4_Nucleo_L476RG_GCC/Boot/main.c](#)
Bootloader application source file.

5.199.1 Detailed Description

Bootloader.

5.200 Demo for Nucleo-L476RG/GCC

Preconfigured programs for the Nucleo-L476RG and the GNU ARM GCC compiler.

Modules

- [Bootloader](#)
Bootloader.
- [User Program](#)
User Program.

5.200.1 Detailed Description

Preconfigured programs for the Nucleo-L476RG and the GNU ARM GCC compiler.

For detailed getting started instructions, refer to: https://www.feaser.com/openblt/doku.php?id=manual:demos:nucleo_l476rg_gcc.

5.201 User Program

User Program.

Files

- file [Demo/ARMCM4_STM32L4_Nucleo_L476RG_GCC/Prog/boot.c](#)
Demo program bootloader interface source file.
- file [Demo/ARMCM4_STM32L4_Nucleo_L476RG_GCC/Prog/boot.h](#)
Demo program bootloader interface header file.
- file [ARMCM4_STM32L4_Nucleo_L476RG_GCC/Prog/header.h](#)
Generic header file.
- file [ARMCM4_STM32L4_Nucleo_L476RG_GCC/Prog/led.c](#)
LED driver source file.
- file [ARMCM4_STM32L4_Nucleo_L476RG_GCC/Prog/led.h](#)
LED driver header file.
- file [ARMCM4_STM32L4_Nucleo_L476RG_GCC/Prog/main.c](#)
Demo program application source file.
- file [Demo/ARMCM4_STM32L4_Nucleo_L476RG_GCC/Prog/timer.c](#)
Timer driver source file.
- file [Demo/ARMCM4_STM32L4_Nucleo_L476RG_GCC/Prog/timer.h](#)
Timer driver header file.

5.201.1 Detailed Description

User Program.

The intention of the demo user program is two-fold. (1) To test the bootloader, you need some sort of firmware to see if you can perform a firmware update with the bootloader. This program can be used for this purpose. (2) To make firmware programmable by the bootloader, a few adjustments to the firmware are required. The demo user program serves as an example for how these adjustments can be implemented. Additional details on this subject can be found in the port specifics documentation, which is available at: https://www.feaser.com/openblt/doku.php?id=manual:ports:armcm4_stm32l4.

5.202 Bootloader

Bootloader.

Files

- file [ARMCM4_STM32L4_Nucleo_L476RG_IAR/Boot/blt_conf.h](#)
Bootloader configuration header file.
- file [ARMCM4_STM32L4_Nucleo_L476RG_IAR/Boot/hooks.c](#)
Bootloader callback source file.
- file [ARMCM4_STM32L4_Nucleo_L476RG_IAR/Boot/led.c](#)
LED driver source file.
- file [ARMCM4_STM32L4_Nucleo_L476RG_IAR/Boot/led.h](#)
LED driver header file.
- file [ARMCM4_STM32L4_Nucleo_L476RG_IAR/Boot/main.c](#)
Bootloader application source file.

5.202.1 Detailed Description

Bootloader.

5.203 Demo for Nucleo-L476RG/IAR

Preconfigured programs for the Nucleo-L476RG and the IAR Embedded Workbench IDE.

Modules

- [Bootloader](#)
Bootloader.
- [User Program](#)
User Program.

5.203.1 Detailed Description

Preconfigured programs for the Nucleo-L476RG and the IAR Embedded Workbench IDE.

For detailed getting started instructions, refer to: https://www.feaser.com/openblt/doku.php?id=manual:demos:nucleo_l476rg_iar.

5.204 User Program

User Program.

Files

- file [Demo/ARMCM4_STM32L4_Nucleo_L476RG_IAR/Prog/boot.c](#)
Demo program bootloader interface source file.
- file [Demo/ARMCM4_STM32L4_Nucleo_L476RG_IAR/Prog/boot.h](#)
Demo program bootloader interface header file.
- file [ARMCM4_STM32L4_Nucleo_L476RG_IAR/Prog/header.h](#)
Generic header file.
- file [ARMCM4_STM32L4_Nucleo_L476RG_IAR/Prog/led.c](#)
LED driver source file.
- file [ARMCM4_STM32L4_Nucleo_L476RG_IAR/Prog/led.h](#)
LED driver header file.
- file [ARMCM4_STM32L4_Nucleo_L476RG_IAR/Prog/main.c](#)
Demo program application source file.
- file [Demo/ARMCM4_STM32L4_Nucleo_L476RG_IAR/Prog/timer.c](#)
Timer driver source file.
- file [Demo/ARMCM4_STM32L4_Nucleo_L476RG_IAR/Prog/timer.h](#)
Timer driver header file.

5.204.1 Detailed Description

User Program.

The intention of the demo user program is two-fold. (1) To test the bootloader, you need some sort of firmware to see if you can perform a firmware update with the bootloader. This program can be used for this purpose. (2) To make firmware programmable by the bootloader, a few adjustments to the firmware are required. The demo user program serves as an example for how these adjustments can be implemented. Additional details on this subject can be found in the port specifics documentation, which is available at: https://www.feaser.com/openblt/doku.php?id=manual:ports:armcm4_stm32l4.

5.205 Bootloader

Bootloader.

Files

- file [ARMCM4_STM32L4_Nucleo_L476RG_Keil/Boot/blt_conf.h](#)
Bootloader configuration header file.
- file [ARMCM4_STM32L4_Nucleo_L476RG_Keil/Boot/hooks.c](#)
Bootloader callback source file.
- file [ARMCM4_STM32L4_Nucleo_L476RG_Keil/Boot/led.c](#)
LED driver source file.
- file [ARMCM4_STM32L4_Nucleo_L476RG_Keil/Boot/led.h](#)
LED driver header file.
- file [ARMCM4_STM32L4_Nucleo_L476RG_Keil/Boot/main.c](#)
Bootloader application source file.

5.205.1 Detailed Description

Bootloader.

5.206 Demo for Nucleo-L476RG/Keil

Preconfigured programs for the Nucleo-L476RG and the Keil MDK toolchain.

Modules

- [Bootloader](#)
Bootloader.
- [User Program](#)
User Program.

5.206.1 Detailed Description

Preconfigured programs for the Nucleo-L476RG and the Keil MDK toolchain.

For detailed getting started instructions, refer to: https://www.feaser.com/openblt/doku.php?id=manual:demos:nucleo_l476rg_keil.

5.207 User Program

User Program.

Files

- file [Demo/ARMCM4_STM32L4_Nucleo_L476RG_Keil/Prog/boot.c](#)
Demo program bootloader interface source file.
- file [Demo/ARMCM4_STM32L4_Nucleo_L476RG_Keil/Prog/boot.h](#)
Demo program bootloader interface header file.
- file [ARMCM4_STM32L4_Nucleo_L476RG_Keil/Prog/header.h](#)
Generic header file.
- file [ARMCM4_STM32L4_Nucleo_L476RG_Keil/Prog/led.c](#)
LED driver source file.
- file [ARMCM4_STM32L4_Nucleo_L476RG_Keil/Prog/led.h](#)
LED driver header file.
- file [ARMCM4_STM32L4_Nucleo_L476RG_Keil/Prog/main.c](#)
Demo program application source file.
- file [Demo/ARMCM4_STM32L4_Nucleo_L476RG_Keil/Prog/timer.c](#)
Timer driver source file.
- file [Demo/ARMCM4_STM32L4_Nucleo_L476RG_Keil/Prog/timer.h](#)
Timer driver header file.

5.207.1 Detailed Description

User Program.

The intention of the demo user program is two-fold. (1) To test the bootloader, you need some sort of firmware to see if you can perform a firmware update with the bootloader. This program can be used for this purpose. (2) To make firmware programmable by the bootloader, a few adjustments to the firmware are required. The demo user program serves as an example for how these adjustments can be implemented. Additional details on this subject can be found in the port specifics documentation, which is available at: https://www.feaser.com/openblt/doku.php?id=manual:ports:armcm4_stm32l4.

5.208 Bootloader

Bootloader.

Files

- file [ARMCM4_TM4C_DK_TM4C123G_IAR/Boot/blt_conf.h](#)
Bootloader configuration header file.
- file [ARMCM4_TM4C_DK_TM4C123G_IAR/Boot/hooks.c](#)
Bootloader callback source file.
- file [ARMCM4_TM4C_DK_TM4C123G_IAR/Boot/main.c](#)
Bootloader application source file.
- file [ARMCM4_TM4C_DK_TM4C123G_IAR/Boot/vectors.c](#)
Bootloader interrupt vector table source file.

5.208.1 Detailed Description

Bootloader.

5.209 Demo for Texas Instruments DK-TM4C123G/IAR

Preconfigured programs for the Texas Instruments DK-TM4C123G and the IAR Embedded Workbench IDE.

Modules

- [Bootloader](#)
Bootloader.
- [User Program](#)
User Program.

5.209.1 Detailed Description

Preconfigured programs for the Texas Instruments DK-TM4C123G and the IAR Embedded Workbench IDE.

For detailed getting started instructions, refer to: https://www.feaser.com/openblt/doku.php?id=manual:demos:dk_tm4c123g_iar.

5.210 User Program

User Program.

Files

- file [Demo/ARMCM4_TM4C_DK_TM4C123G_IAR/Prog/boot.c](#)
Demo program bootloader interface source file.
- file [Demo/ARMCM4_TM4C_DK_TM4C123G_IAR/Prog/boot.h](#)
Demo program bootloader interface header file.
- file [ARMCM4_TM4C_DK_TM4C123G_IAR/Prog/header.h](#)
Generic header file.
- file [ARMCM4_TM4C_DK_TM4C123G_IAR/Prog/led.c](#)
LED driver source file.
- file [ARMCM4_TM4C_DK_TM4C123G_IAR/Prog/led.h](#)
LED driver header file.
- file [ARMCM4_TM4C_DK_TM4C123G_IAR/Prog/main.c](#)
Demo program application source file.
- file [ARMCM4_TM4C_DK_TM4C123G_IAR/Prog/time.c](#)
Timer driver source file.
- file [ARMCM4_TM4C_DK_TM4C123G_IAR/Prog/time.h](#)
Timer driver header file.
- file [ARMCM4_TM4C_DK_TM4C123G_IAR/Prog/vectors.c](#)
Demo program interrupt vectors source file.

5.210.1 Detailed Description

User Program.

The intention of the demo user program is two-fold. (1) To test the bootloader, you need some sort of firmware to see if you can perform a firmware update with the bootloader. This program can be used for this purpose. (2) To make firmware programmable by the bootloader, a few adjustments to the firmware are required. The demo user program serves as an example for how these adjustments can be implemented. Additional details on this subject can be found in the port specifics documentation, which is available at: https://www.feaser.com/openblt/doku.php?id=manual:ports:armcm4_tm4c.

5.211 Bootloader

Bootloader.

Files

- file [ARMCM4_XMC4_XMC4700_Relax_Kit_GCC/Boot/blt_conf.h](#)
Bootloader configuration header file.
- file [ARMCM4_XMC4_XMC4700_Relax_Kit_GCC/Boot/hooks.c](#)
Bootloader callback source file.
- file [ARMCM4_XMC4_XMC4700_Relax_Kit_GCC/Boot/led.c](#)
LED driver source file.
- file [ARMCM4_XMC4_XMC4700_Relax_Kit_GCC/Boot/led.h](#)
LED driver header file.
- file [ARMCM4_XMC4_XMC4700_Relax_Kit_GCC/Boot/main.c](#)
Bootloader application source file.
- file [ARMCM4_XMC4_XMC4700_Relax_Kit_GCC/Boot/shared_params.c](#)
Shared RAM parameters source file.
- file [ARMCM4_XMC4_XMC4700_Relax_Kit_GCC/Boot/shared_params.h](#)
Shared RAM parameters header file.

5.211.1 Detailed Description

Bootloader.

5.212 Demo for XMC4700 Relax Kit/GCC

Preconfigured programs for the Infineon XMC4700 Relax Kit board and the Dave 4 development environment, which is based on the ARM GCC toolchain.

Modules

- [Bootloader](#)
Bootloader.
- [User Program](#)
User Program.

5.212.1 Detailed Description

Preconfigured programs for the Infineon XMC4700 Relax Kit board and the Dave 4 development environment, which is based on the ARM GCC toolchain.

For detailed getting started instructions, refer to: https://www.feaser.com/openblt/doku.php?id=manual:demos:infineon_xmc4700_relax_kit_gcc.

5.213 User Program

User Program.

Files

- file [Demo/ARMCM4_XMC4_XMC4700_Relax_Kit_GCC/Prog/boot.c](#)
Demo program bootloader interface source file.
- file [Demo/ARMCM4_XMC4_XMC4700_Relax_Kit_GCC/Prog/boot.h](#)
Demo program bootloader interface header file.
- file [ARMCM4_XMC4_XMC4700_Relax_Kit_GCC/Prog/header.h](#)
Generic header file.
- file [ARMCM4_XMC4_XMC4700_Relax_Kit_GCC/Prog/led.c](#)
LED driver source file.
- file [ARMCM4_XMC4_XMC4700_Relax_Kit_GCC/Prog/led.h](#)
LED driver header file.
- file [ARMCM4_XMC4_XMC4700_Relax_Kit_GCC/Prog/main.c](#)
Demo program application source file.
- file [Demo/ARMCM4_XMC4_XMC4700_Relax_Kit_GCC/Prog/net.c](#)
Network application for the uIP TCP/IP stack.
- file [Demo/ARMCM4_XMC4_XMC4700_Relax_Kit_GCC/Prog/net.h](#)
Network application for the uIP TCP/IP stack.
- file [ARMCM4_XMC4_XMC4700_Relax_Kit_GCC/Prog/shared_params.c](#)
Shared RAM parameters source file.
- file [ARMCM4_XMC4_XMC4700_Relax_Kit_GCC/Prog/shared_params.h](#)
Shared RAM parameters header file.
- file [Demo/ARMCM4_XMC4_XMC4700_Relax_Kit_GCC/Prog/timer.c](#)
Timer driver source file.
- file [Demo/ARMCM4_XMC4_XMC4700_Relax_Kit_GCC/Prog/timer.h](#)
Timer driver header file.

5.213.1 Detailed Description

User Program.

The intention of the demo user program is two-fold. (1) To test the bootloader, you need some sort of firmware to see if you can perform a firmware update with the bootloader. This program can be used for this purpose. (2) To make firmware programmable by the bootloader, a few adjustments to the firmware are required. The demo user program serves as an example for how these adjustments can be implemented. Additional details on this subject can be found in the port specifics documentation, which is available at: https://www.feaser.com/openblt/doku.php?id=manual:ports:armcm4_xmc4.

5.214 Bootloader

Bootloader.

Files

- file [ARMCM4_XMC4_XMC4700_Relax_Kit_IAR/Boot/blt_conf.h](#)
Bootloader configuration header file.
- file [ARMCM4_XMC4_XMC4700_Relax_Kit_IAR/Boot/hooks.c](#)
Bootloader callback source file.
- file [ARMCM4_XMC4_XMC4700_Relax_Kit_IAR/Boot/led.c](#)
LED driver source file.
- file [ARMCM4_XMC4_XMC4700_Relax_Kit_IAR/Boot/led.h](#)
LED driver header file.
- file [ARMCM4_XMC4_XMC4700_Relax_Kit_IAR/Boot/main.c](#)
Bootloader application source file.
- file [ARMCM4_XMC4_XMC4700_Relax_Kit_IAR/Boot/shared_params.c](#)
Shared RAM parameters source file.
- file [ARMCM4_XMC4_XMC4700_Relax_Kit_IAR/Boot/shared_params.h](#)
Shared RAM parameters header file.

5.214.1 Detailed Description

Bootloader.

5.215 Demo for XMC4700 Relax Kit/IAR

Preconfigured programs for the Infineon XMC4700 Relax Kit board and the IAR compiler.

Modules

- [Bootloader](#)
Bootloader.
- [User Program](#)
User Program.

5.215.1 Detailed Description

Preconfigured programs for the Infineon XMC4700 Relax Kit board and the IAR compiler.

For detailed getting started instructions, refer to: https://www.feaser.com/openblt/doku.php?id=manual:demos:infineon_xmc4700_relax_kit_iar.

5.216 User Program

User Program.

Files

- file [Demo/ARMCM4_XMC4_XMC4700_Relax_Kit_IAR/Prog/boot.c](#)
Demo program bootloader interface source file.
- file [Demo/ARMCM4_XMC4_XMC4700_Relax_Kit_IAR/Prog/boot.h](#)
Demo program bootloader interface header file.
- file [ARMCM4_XMC4_XMC4700_Relax_Kit_IAR/Prog/header.h](#)
Generic header file.
- file [ARMCM4_XMC4_XMC4700_Relax_Kit_IAR/Prog/led.c](#)
LED driver source file.
- file [ARMCM4_XMC4_XMC4700_Relax_Kit_IAR/Prog/led.h](#)
LED driver header file.
- file [ARMCM4_XMC4_XMC4700_Relax_Kit_IAR/Prog/main.c](#)
Demo program application source file.
- file [Demo/ARMCM4_XMC4_XMC4700_Relax_Kit_IAR/Prog/net.c](#)
Network application for the uIP TCP/IP stack.
- file [Demo/ARMCM4_XMC4_XMC4700_Relax_Kit_IAR/Prog/net.h](#)
Network application for the uIP TCP/IP stack.
- file [ARMCM4_XMC4_XMC4700_Relax_Kit_IAR/Prog/shared_params.c](#)
Shared RAM parameters source file.
- file [ARMCM4_XMC4_XMC4700_Relax_Kit_IAR/Prog/shared_params.h](#)
Shared RAM parameters header file.
- file [Demo/ARMCM4_XMC4_XMC4700_Relax_Kit_IAR/Prog/timer.c](#)
Timer driver source file.
- file [Demo/ARMCM4_XMC4_XMC4700_Relax_Kit_IAR/Prog/timer.h](#)
Timer driver header file.

5.216.1 Detailed Description

User Program.

The intention of the demo user program is two-fold. (1) To test the bootloader, you need some sort of firmware to see if you can perform a firmware update with the bootloader. This program can be used for this purpose. (2) To make firmware programmable by the bootloader, a few adjustments to the firmware are required. The demo user program serves as an example for how these adjustments can be implemented. Additional details on this subject can be found in the port specifics documentation, which is available at: https://www.feaser.com/openblt/doku.php?id=manual:ports:armcm4_xmc4.

5.217 Bootloader

Bootloader.

Files

- file [ARMCM7_STM32F7_Nucleo_F746ZG_CubeIDE/Boot/App/app.c](#)
Bootloader application source file.
- file [ARMCM7_STM32F7_Nucleo_F746ZG_CubeIDE/Boot/App/app.h](#)
Bootloader application header file.
- file [ARMCM7_STM32F7_Nucleo_F746ZG_CubeIDE/Boot/App/blt_conf.h](#)
Bootloader configuration header file.
- file [ARMCM7_STM32F7_Nucleo_F746ZG_CubeIDE/Boot/App/hooks.c](#)
Bootloader callback source file.
- file [ARMCM7_STM32F7_Nucleo_F746ZG_CubeIDE/Boot/App/led.c](#)
LED driver source file.
- file [ARMCM7_STM32F7_Nucleo_F746ZG_CubeIDE/Boot/App/led.h](#)
LED driver header file.

5.217.1 Detailed Description

Bootloader.

5.218 Demo for Nucleo-F746ZG/STM32CubeIDE

Preconfigured programs for the Nucleo-F746ZG board and the ST STM32CubeIDE toolchain.

Modules

- [Bootloader](#)
Bootloader.
- [User Program](#)
User Program.

5.218.1 Detailed Description

Preconfigured programs for the Nucleo-F746ZG board and the ST STM32CubeIDE toolchain.

For detailed getting started instructions, refer to: https://www.feaser.com/openblt/doku.php?id=manual:demos:nucleo_f746zg_cubeide.

5.219 User Program

User Program.

Files

- file [ARMCM7_STM32F7_Nucleo_F746ZG_CubeIDE/Prog/App/app.c](#)
User program application source file.
- file [ARMCM7_STM32F7_Nucleo_F746ZG_CubeIDE/Prog/App/app.h](#)
User program application header file.
- file [Demo/ARMCM7_STM32F7_Nucleo_F746ZG_CubeIDE/Prog/App/boot.c](#)
Demo program bootloader interface source file.
- file [Demo/ARMCM7_STM32F7_Nucleo_F746ZG_CubeIDE/Prog/App/boot.h](#)
Demo program bootloader interface header file.
- file [ARMCM7_STM32F7_Nucleo_F746ZG_CubeIDE/Prog/App/header.h](#)
Generic header file.
- file [ARMCM7_STM32F7_Nucleo_F746ZG_CubeIDE/Prog/App/led.c](#)
LED driver source file.
- file [ARMCM7_STM32F7_Nucleo_F746ZG_CubeIDE/Prog/App/led.h](#)
LED driver header file.
- file [Demo/ARMCM7_STM32F7_Nucleo_F746ZG_CubeIDE/Prog/App/timer.c](#)
Timer driver source file.
- file [Demo/ARMCM7_STM32F7_Nucleo_F746ZG_CubeIDE/Prog/App/timer.h](#)
Timer driver header file.

5.219.1 Detailed Description

User Program.

The intention of the demo user program is two-fold. (1) To test the bootloader, you need some sort of firmware to see if you can perform a firmware update with the bootloader. This program can be used for this purpose. (2) To make firmware programmable by the bootloader, a few adjustments to the firmware are required. The demo user program serves as an example for how these adjustments can be implemented. Additional details on this subject can be found in the port specifics documentation, which is available at: https://www.feaser.com/openblt/doku.php?id=manual:ports:armcm7_stm32f7.

5.220 Bootloader

Bootloader.

Files

- file [ARMCM7_STM32F7_Nucleo_F746ZG_GCC/Boot/blt_conf.h](#)
Bootloader configuration header file.
- file [ARMCM7_STM32F7_Nucleo_F746ZG_GCC/Boot/hooks.c](#)
Bootloader callback source file.
- file [ARMCM7_STM32F7_Nucleo_F746ZG_GCC/Boot/led.c](#)
LED driver source file.
- file [ARMCM7_STM32F7_Nucleo_F746ZG_GCC/Boot/led.h](#)
LED driver header file.
- file [ARMCM7_STM32F7_Nucleo_F746ZG_GCC/Boot/main.c](#)
Bootloader application source file.

5.220.1 Detailed Description

Bootloader.

5.221 Demo for Nucleo-F746ZG/GCC

Preconfigured programs for the Nucleo-F746ZG board and the GNU ARM GCC compiler.

Modules

- [Bootloader](#)
Bootloader.
- [User Program](#)
User Program.

5.221.1 Detailed Description

Preconfigured programs for the Nucleo-F746ZG board and the GNU ARM GCC compiler.

For detailed getting started instructions, refer to: https://www.feaser.com/openblt/doku.php?id=manual:demos:nucleo_f746zg_gcc.

5.222 User Program

User Program.

Files

- file [Demo/ARMCM7_STM32F7_Nucleo_F746ZG_GCC/Prog/boot.c](#)
Demo program bootloader interface source file.
- file [Demo/ARMCM7_STM32F7_Nucleo_F746ZG_GCC/Prog/boot.h](#)
Demo program bootloader interface header file.
- file [ARMCM7_STM32F7_Nucleo_F746ZG_GCC/Prog/header.h](#)
Generic header file.
- file [ARMCM7_STM32F7_Nucleo_F746ZG_GCC/Prog/led.c](#)
LED driver source file.
- file [ARMCM7_STM32F7_Nucleo_F746ZG_GCC/Prog/led.h](#)
LED driver header file.
- file [ARMCM7_STM32F7_Nucleo_F746ZG_GCC/Prog/main.c](#)
Demo program application source file.
- file [Demo/ARMCM7_STM32F7_Nucleo_F746ZG_GCC/Prog/timer.c](#)
Timer driver source file.
- file [Demo/ARMCM7_STM32F7_Nucleo_F746ZG_GCC/Prog/timer.h](#)
Timer driver header file.

5.222.1 Detailed Description

User Program.

The intention of the demo user program is two-fold. (1) To test the bootloader, you need some sort of firmware to see if you can perform a firmware update with the bootloader. This program can be used for this purpose. (2) To make firmware programmable by the bootloader, a few adjustments to the firmware are required. The demo user program serves as an example for how these adjustments can be implemented. Additional details on this subject can be found in the port specifics documentation, which is available at: https://www.feaser.com/openblt/doku.php?id=manual:ports:armcm7_stm32f7.

5.223 Bootloader

Bootloader.

Files

- file [ARMCM7_STM32F7_Nucleo_F746ZG_IAR/Boot/blt_conf.h](#)
Bootloader configuration header file.
- file [ARMCM7_STM32F7_Nucleo_F746ZG_IAR/Boot/hooks.c](#)
Bootloader callback source file.
- file [ARMCM7_STM32F7_Nucleo_F746ZG_IAR/Boot/led.c](#)
LED driver source file.
- file [ARMCM7_STM32F7_Nucleo_F746ZG_IAR/Boot/led.h](#)
LED driver header file.
- file [ARMCM7_STM32F7_Nucleo_F746ZG_IAR/Boot/main.c](#)
Bootloader application source file.

5.223.1 Detailed Description

Bootloader.

5.224 Demo for Nucleo-F746ZG/IAR

Preconfigured programs for the Nucleo-F746ZG board and the IAR Embedded Workbench IDE.

Modules

- [Bootloader](#)
Bootloader.
- [User Program](#)
User Program.

5.224.1 Detailed Description

Preconfigured programs for the Nucleo-F746ZG board and the IAR Embedded Workbench IDE.

For detailed getting started instructions, refer to: https://www.feaser.com/openblt/doku.php?id=manual:demos:nucleo_f746zg_iar.

5.225 User Program

User Program.

Files

- file [Demo/ARMCM7_STM32F7_Nucleo_F746ZG_IAR/Prog/boot.c](#)
Demo program bootloader interface source file.
- file [Demo/ARMCM7_STM32F7_Nucleo_F746ZG_IAR/Prog/boot.h](#)
Demo program bootloader interface header file.
- file [ARMCM7_STM32F7_Nucleo_F746ZG_IAR/Prog/header.h](#)
Generic header file.
- file [ARMCM7_STM32F7_Nucleo_F746ZG_IAR/Prog/led.c](#)
LED driver source file.
- file [ARMCM7_STM32F7_Nucleo_F746ZG_IAR/Prog/led.h](#)
LED driver header file.
- file [ARMCM7_STM32F7_Nucleo_F746ZG_IAR/Prog/main.c](#)
Demo program application source file.
- file [Demo/ARMCM7_STM32F7_Nucleo_F746ZG_IAR/Prog/timer.c](#)
Timer driver source file.
- file [Demo/ARMCM7_STM32F7_Nucleo_F746ZG_IAR/Prog/timer.h](#)
Timer driver header file.

5.225.1 Detailed Description

User Program.

The intention of the demo user program is two-fold. (1) To test the bootloader, you need some sort of firmware to see if you can perform a firmware update with the bootloader. This program can be used for this purpose. (2) To make firmware programmable by the bootloader, a few adjustments to the firmware are required. The demo user program serves as an example for how these adjustments can be implemented. Additional details on this subject can be found in the port specifics documentation, which is available at: https://www.feaser.com/openblt/doku.php?id=manual:ports:armcm7_stm32f7.

5.226 Bootloader

Bootloader.

Files

- file [ARMCM7_STM32F7_Nucleo_F746ZG_Keil/Boot/blt_conf.h](#)
Bootloader configuration header file.
- file [ARMCM7_STM32F7_Nucleo_F746ZG_Keil/Boot/hooks.c](#)
Bootloader callback source file.
- file [ARMCM7_STM32F7_Nucleo_F746ZG_Keil/Boot/led.c](#)
LED driver source file.
- file [ARMCM7_STM32F7_Nucleo_F746ZG_Keil/Boot/led.h](#)
LED driver header file.
- file [ARMCM7_STM32F7_Nucleo_F746ZG_Keil/Boot/main.c](#)
Bootloader application source file.

5.226.1 Detailed Description

Bootloader.

5.227 Demo for Nucleo-F746ZG/Keil

Preconfigured programs for the Nucleo-F746ZG board and the Keil MDK for ARM IDE.

Modules

- [Bootloader](#)
Bootloader.
- [User Program](#)
User Program.

5.227.1 Detailed Description

Preconfigured programs for the Nucleo-F746ZG board and the Keil MDK for ARM IDE.

For detailed getting started instructions, refer to: https://www.feaser.com/openblt/doku.php?id=manual:demos:nucleo_f746zg_keil.

5.228 User Program

User Program.

Files

- file [Demo/ARMCM7_STM32F7_Nucleo_F746ZG_Keil/Prog/boot.c](#)
Demo program bootloader interface source file.
- file [Demo/ARMCM7_STM32F7_Nucleo_F746ZG_Keil/Prog/boot.h](#)
Demo program bootloader interface header file.
- file [ARMCM7_STM32F7_Nucleo_F746ZG_Keil/Prog/header.h](#)
Generic header file.
- file [ARMCM7_STM32F7_Nucleo_F746ZG_Keil/Prog/led.c](#)
LED driver source file.
- file [ARMCM7_STM32F7_Nucleo_F746ZG_Keil/Prog/led.h](#)
LED driver header file.
- file [ARMCM7_STM32F7_Nucleo_F746ZG_Keil/Prog/main.c](#)
Demo program application source file.
- file [Demo/ARMCM7_STM32F7_Nucleo_F746ZG_Keil/Prog/timer.c](#)
Timer driver source file.
- file [Demo/ARMCM7_STM32F7_Nucleo_F746ZG_Keil/Prog/timer.h](#)
Timer driver header file.

5.228.1 Detailed Description

User Program.

The intention of the demo user program is two-fold. (1) To test the bootloader, you need some sort of firmware to see if you can perform a firmware update with the bootloader. This program can be used for this purpose. (2) To make firmware programmable by the bootloader, a few adjustments to the firmware are required. The demo user program serves as an example for how these adjustments can be implemented. Additional details on this subject can be found in the port specifics documentation, which is available at: https://www.feaser.com/openblt/doku.php?id=manual:ports:armcm7_stm32f7.

5.229 Bootloader

Bootloader.

Files

- file [ARMCM7_STM32F7_Nucleo_F767ZI_CubeIDE/Boot/App/app.c](#)
Bootloader application source file.
- file [ARMCM7_STM32F7_Nucleo_F767ZI_CubeIDE/Boot/App/app.h](#)
Bootloader application header file.
- file [ARMCM7_STM32F7_Nucleo_F767ZI_CubeIDE/Boot/App/blt_conf.h](#)
Bootloader configuration header file.
- file [ARMCM7_STM32F7_Nucleo_F767ZI_CubeIDE/Boot/App/hooks.c](#)
Bootloader callback source file.
- file [ARMCM7_STM32F7_Nucleo_F767ZI_CubeIDE/Boot/App/led.c](#)
LED driver source file.
- file [ARMCM7_STM32F7_Nucleo_F767ZI_CubeIDE/Boot/App/led.h](#)
LED driver header file.
- file [ARMCM7_STM32F7_Nucleo_F767ZI_CubeIDE/Boot/App/shared_params.c](#)
Shared RAM parameters source file.
- file [ARMCM7_STM32F7_Nucleo_F767ZI_CubeIDE/Boot/App/shared_params.h](#)
Shared RAM parameters header file.

5.229.1 Detailed Description

Bootloader.

5.230 Demo for Nucleo-F767ZI/STM32CubeIDE

Preconfigured programs for the Nucleo-F767ZI board and the ST STM32CubeIDE toolchain.

Modules

- [Bootloader](#)
Bootloader.
- [User Program](#)
User Program.

5.230.1 Detailed Description

Preconfigured programs for the Nucleo-F767ZI board and the ST STM32CubeIDE toolchain.

For detailed getting started instructions, refer to: https://www.feaser.com/openblt/doku.php?id=manual:demos:nucleo_f767zi_cubeide.

5.231 User Program

User Program.

Files

- file [ARMCM7_STM32F7_Nucleo_F767ZI_CubeIDE/Prog/App/app.c](#)
User program application source file.
- file [ARMCM7_STM32F7_Nucleo_F767ZI_CubeIDE/Prog/App/app.h](#)
User program application header file.
- file [Demo/ARMCM7_STM32F7_Nucleo_F767ZI_CubeIDE/Prog/App/boot.c](#)
Demo program bootloader interface source file.
- file [Demo/ARMCM7_STM32F7_Nucleo_F767ZI_CubeIDE/Prog/App/boot.h](#)
Demo program bootloader interface header file.
- file [ARMCM7_STM32F7_Nucleo_F767ZI_CubeIDE/Prog/App/header.h](#)
Generic header file.
- file [ARMCM7_STM32F7_Nucleo_F767ZI_CubeIDE/Prog/App/led.c](#)
LED driver source file.
- file [ARMCM7_STM32F7_Nucleo_F767ZI_CubeIDE/Prog/App/led.h](#)
LED driver header file.
- file [Demo/ARMCM7_STM32F7_Nucleo_F767ZI_CubeIDE/Prog/App/net.c](#)
Network application for the uIP TCP/IP stack.
- file [Demo/ARMCM7_STM32F7_Nucleo_F767ZI_CubeIDE/Prog/App/net.h](#)
Network application for the uIP TCP/IP stack.
- file [ARMCM7_STM32F7_Nucleo_F767ZI_CubeIDE/Prog/App/shared_params.c](#)
Shared RAM parameters source file.
- file [ARMCM7_STM32F7_Nucleo_F767ZI_CubeIDE/Prog/App/shared_params.h](#)
Shared RAM parameters header file.
- file [Demo/ARMCM7_STM32F7_Nucleo_F767ZI_CubeIDE/Prog/App/timer.c](#)
Timer driver source file.
- file [Demo/ARMCM7_STM32F7_Nucleo_F767ZI_CubeIDE/Prog/App/timer.h](#)
Timer driver header file.

5.231.1 Detailed Description

User Program.

The intention of the demo user program is two-fold. (1) To test the bootloader, you need some sort of firmware to see if you can perform a firmware update with the bootloader. This program can be used for this purpose. (2) To make firmware programmable by the bootloader, a few adjustments to the firmware are required. The demo user program serves as an example for how these adjustments can be implemented. Additional details on this subject can be found in the port specifics documentation, which is available at: https://www.feaser.com/openblt/doku.php?id=manual:ports:armcm7_stm32f7.

5.232 Bootloader

Bootloader.

Files

- file [ARMCM7_STM32F7_Nucleo_F767ZI_GCC/Boot/blt_conf.h](#)
Bootloader configuration header file.
- file [ARMCM7_STM32F7_Nucleo_F767ZI_GCC/Boot/hooks.c](#)
Bootloader callback source file.
- file [ARMCM7_STM32F7_Nucleo_F767ZI_GCC/Boot/led.c](#)
LED driver source file.
- file [ARMCM7_STM32F7_Nucleo_F767ZI_GCC/Boot/led.h](#)
LED driver header file.
- file [ARMCM7_STM32F7_Nucleo_F767ZI_GCC/Boot/main.c](#)
Bootloader application source file.
- file [ARMCM7_STM32F7_Nucleo_F767ZI_GCC/Boot/shared_params.c](#)
Shared RAM parameters source file.
- file [ARMCM7_STM32F7_Nucleo_F767ZI_GCC/Boot/shared_params.h](#)
Shared RAM parameters header file.

5.232.1 Detailed Description

Bootloader.

5.233 Demo for Nucleo-F767ZI/GCC

Preconfigured programs for the Nucleo-F767ZI board and the GNU ARM GCC compiler.

Modules

- [Bootloader](#)
Bootloader.
- [User Program](#)
User Program.

5.233.1 Detailed Description

Preconfigured programs for the Nucleo-F767ZI board and the GNU ARM GCC compiler.

For detailed getting started instructions, refer to: https://www.feaser.com/openblt/doku.php?id=manual:demos:nucleo_f767zi_gcc.

5.234 User Program

User Program.

Files

- file [Demo/ARMCM7_STM32F7_Nucleo_F767ZI_GCC/Prog/boot.c](#)
Demo program bootloader interface source file.
- file [Demo/ARMCM7_STM32F7_Nucleo_F767ZI_GCC/Prog/boot.h](#)
Demo program bootloader interface header file.
- file [ARMCM7_STM32F7_Nucleo_F767ZI_GCC/Prog/header.h](#)
Generic header file.
- file [ARMCM7_STM32F7_Nucleo_F767ZI_GCC/Prog/led.c](#)
LED driver source file.
- file [ARMCM7_STM32F7_Nucleo_F767ZI_GCC/Prog/led.h](#)
LED driver header file.
- file [ARMCM7_STM32F7_Nucleo_F767ZI_GCC/Prog/main.c](#)
Demo program application source file.
- file [Demo/ARMCM7_STM32F7_Nucleo_F767ZI_GCC/Prog/net.c](#)
Network application for the uIP TCP/IP stack.
- file [Demo/ARMCM7_STM32F7_Nucleo_F767ZI_GCC/Prog/net.h](#)
Network application for the uIP TCP/IP stack.
- file [ARMCM7_STM32F7_Nucleo_F767ZI_GCC/Prog/shared_params.c](#)
Shared RAM parameters source file.
- file [ARMCM7_STM32F7_Nucleo_F767ZI_GCC/Prog/shared_params.h](#)
Shared RAM parameters header file.
- file [Demo/ARMCM7_STM32F7_Nucleo_F767ZI_GCC/Prog/timer.c](#)
Timer driver source file.
- file [Demo/ARMCM7_STM32F7_Nucleo_F767ZI_GCC/Prog/timer.h](#)
Timer driver header file.

5.234.1 Detailed Description

User Program.

The intention of the demo user program is two-fold. (1) To test the bootloader, you need some sort of firmware to see if you can perform a firmware update with the bootloader. This program can be used for this purpose. (2) To make firmware programmable by the bootloader, a few adjustments to the firmware are required. The demo user program serves as an example for how these adjustments can be implemented. Additional details on this subject can be found in the port specifics documentation, which is available at: https://www.feaser.com/openblt/doku.php?id=manual:ports:armcm7_stm32f7.

5.235 Bootloader

Bootloader.

Files

- file [ARMCM7_STM32F7_Nucleo_F767ZI_IAR/Boot/blt_conf.h](#)
Bootloader configuration header file.
- file [ARMCM7_STM32F7_Nucleo_F767ZI_IAR/Boot/hooks.c](#)
Bootloader callback source file.
- file [ARMCM7_STM32F7_Nucleo_F767ZI_IAR/Boot/led.c](#)
LED driver source file.
- file [ARMCM7_STM32F7_Nucleo_F767ZI_IAR/Boot/led.h](#)
LED driver header file.
- file [ARMCM7_STM32F7_Nucleo_F767ZI_IAR/Boot/main.c](#)
Bootloader application source file.
- file [ARMCM7_STM32F7_Nucleo_F767ZI_IAR/Boot/shared_params.c](#)
Shared RAM parameters source file.
- file [ARMCM7_STM32F7_Nucleo_F767ZI_IAR/Boot/shared_params.h](#)
Shared RAM parameters header file.

5.235.1 Detailed Description

Bootloader.

5.236 Demo for Nucleo-F767ZI/IAR

Preconfigured programs for the Nucleo-F767ZI board and the IAR Embedded Workbench IDE.

Modules

- [Bootloader](#)
Bootloader.
- [User Program](#)
User Program.

5.236.1 Detailed Description

Preconfigured programs for the Nucleo-F767ZI board and the IAR Embedded Workbench IDE.

For detailed getting started instructions, refer to: https://www.feaser.com/openblt/doku.php?id=manual:demos:nucleo_f767zi_iar.

5.237 User Program

User Program.

Files

- file [Demo/ARMCM7_STM32F7_Nucleo_F767ZI_IAR/Prog/boot.c](#)
Demo program bootloader interface source file.
- file [Demo/ARMCM7_STM32F7_Nucleo_F767ZI_IAR/Prog/boot.h](#)
Demo program bootloader interface header file.
- file [ARMCM7_STM32F7_Nucleo_F767ZI_IAR/Prog/header.h](#)
Generic header file.
- file [ARMCM7_STM32F7_Nucleo_F767ZI_IAR/Prog/led.c](#)
LED driver source file.
- file [ARMCM7_STM32F7_Nucleo_F767ZI_IAR/Prog/led.h](#)
LED driver header file.
- file [ARMCM7_STM32F7_Nucleo_F767ZI_IAR/Prog/main.c](#)
Demo program application source file.
- file [Demo/ARMCM7_STM32F7_Nucleo_F767ZI_IAR/Prog/net.c](#)
Network application for the uIP TCP/IP stack.
- file [Demo/ARMCM7_STM32F7_Nucleo_F767ZI_IAR/Prog/net.h](#)
Network application for the uIP TCP/IP stack.
- file [ARMCM7_STM32F7_Nucleo_F767ZI_IAR/Prog/shared_params.c](#)
Shared RAM parameters source file.
- file [ARMCM7_STM32F7_Nucleo_F767ZI_IAR/Prog/shared_params.h](#)
Shared RAM parameters header file.
- file [Demo/ARMCM7_STM32F7_Nucleo_F767ZI_IAR/Prog/timer.c](#)
Timer driver source file.
- file [Demo/ARMCM7_STM32F7_Nucleo_F767ZI_IAR/Prog/timer.h](#)
Timer driver header file.

5.237.1 Detailed Description

User Program.

The intention of the demo user program is two-fold. (1) To test the bootloader, you need some sort of firmware to see if you can perform a firmware update with the bootloader. This program can be used for this purpose. (2) To make firmware programmable by the bootloader, a few adjustments to the firmware are required. The demo user program serves as an example for how these adjustments can be implemented. Additional details on this subject can be found in the port specifics documentation, which is available at: <https://www.feaser.com/openblt/doku.php?id=manual:ports:armcm7-stm32f7>.

5.238 Bootloader

Bootloader.

Files

- file [ARMCM7_STM32F7_Nucleo_F767ZI_Keil/Boot/blt_conf.h](#)
Bootloader configuration header file.
- file [ARMCM7_STM32F7_Nucleo_F767ZI_Keil/Boot/hooks.c](#)
Bootloader callback source file.
- file [ARMCM7_STM32F7_Nucleo_F767ZI_Keil/Boot/led.c](#)
LED driver source file.
- file [ARMCM7_STM32F7_Nucleo_F767ZI_Keil/Boot/led.h](#)
LED driver header file.
- file [ARMCM7_STM32F7_Nucleo_F767ZI_Keil/Boot/main.c](#)
Bootloader application source file.
- file [ARMCM7_STM32F7_Nucleo_F767ZI_Keil/Boot/shared_params.c](#)
Shared RAM parameters source file.
- file [ARMCM7_STM32F7_Nucleo_F767ZI_Keil/Boot/shared_params.h](#)
Shared RAM parameters header file.

5.238.1 Detailed Description

Bootloader.

5.239 Demo for Nucleo-F767ZI/Keil

Preconfigured programs for the Nucleo-F767ZI board and the Keil MDK toolchain.

Modules

- [Bootloader](#)
Bootloader.
- [User Program](#)
User Program.

5.239.1 Detailed Description

Preconfigured programs for the Nucleo-F767ZI board and the Keil MDK toolchain.

For detailed getting started instructions, refer to: https://www.feaser.com/openblt/doku.php?id=manual:demos:nucleo_f767zi_keil.

5.240 User Program

User Program.

Files

- file [Demo/ARMCM7_STM32F7_Nucleo_F767ZI_Keil/Prog/boot.c](#)
Demo program bootloader interface source file.
- file [Demo/ARMCM7_STM32F7_Nucleo_F767ZI_Keil/Prog/boot.h](#)
Demo program bootloader interface header file.
- file [ARMCM7_STM32F7_Nucleo_F767ZI_Keil/Prog/header.h](#)
Generic header file.
- file [ARMCM7_STM32F7_Nucleo_F767ZI_Keil/Prog/led.c](#)
LED driver source file.
- file [ARMCM7_STM32F7_Nucleo_F767ZI_Keil/Prog/led.h](#)
LED driver header file.
- file [ARMCM7_STM32F7_Nucleo_F767ZI_Keil/Prog/main.c](#)
Demo program application source file.
- file [Demo/ARMCM7_STM32F7_Nucleo_F767ZI_Keil/Prog/net.c](#)
Network application for the uIP TCP/IP stack.
- file [Demo/ARMCM7_STM32F7_Nucleo_F767ZI_Keil/Prog/net.h](#)
Network application for the uIP TCP/IP stack.
- file [ARMCM7_STM32F7_Nucleo_F767ZI_Keil/Prog/shared_params.c](#)
Shared RAM parameters source file.
- file [ARMCM7_STM32F7_Nucleo_F767ZI_Keil/Prog/shared_params.h](#)
Shared RAM parameters header file.
- file [Demo/ARMCM7_STM32F7_Nucleo_F767ZI_Keil/Prog/timer.c](#)
Timer driver source file.
- file [Demo/ARMCM7_STM32F7_Nucleo_F767ZI_Keil/Prog/timer.h](#)
Timer driver header file.

5.240.1 Detailed Description

User Program.

The intention of the demo user program is two-fold. (1) To test the bootloader, you need some sort of firmware to see if you can perform a firmware update with the bootloader. This program can be used for this purpose. (2) To make firmware programmable by the bootloader, a few adjustments to the firmware are required. The demo user program serves as an example for how these adjustments can be implemented. Additional details on this subject can be found in the port specifics documentation, which is available at: https://www.feaser.com/openblt/doku.php?id=manual:ports:armcm7_stm32f7.

5.241 Bootloader

Bootloader.

Files

- file [ARMCM7_STM32H7_Nucleo_H743ZI_CubeIDE/Boot/App/app.c](#)
Bootloader application source file.
- file [ARMCM7_STM32H7_Nucleo_H743ZI_CubeIDE/Boot/App/app.h](#)
Bootloader application header file.
- file [ARMCM7_STM32H7_Nucleo_H743ZI_CubeIDE/Boot/App/blt_conf.h](#)
Bootloader configuration header file.
- file [ARMCM7_STM32H7_Nucleo_H743ZI_CubeIDE/Boot/App/hooks.c](#)
Bootloader callback source file.
- file [ARMCM7_STM32H7_Nucleo_H743ZI_CubeIDE/Boot/App/led.c](#)
LED driver source file.
- file [ARMCM7_STM32H7_Nucleo_H743ZI_CubeIDE/Boot/App/led.h](#)
LED driver header file.

5.241.1 Detailed Description

Bootloader.

5.242 Demo for Nucleo-H743ZI/STM32CubeIDE

Preconfigured programs for the Nucleo-H743ZI board and the ST STM32CubeIDE toolchain.

Modules

- [Bootloader](#)
Bootloader.
- [User Program](#)
User Program.

5.242.1 Detailed Description

Preconfigured programs for the Nucleo-H743ZI board and the ST STM32CubeIDE toolchain.

For detailed getting started instructions, refer to: https://www.feaser.com/openblt/doku.php?id=manual:demos:nucleo_h743zi_cubeide.

5.243 User Program

User Program.

Files

- file [ARMCM7_STM32H7_Nucleo_H743ZI_CubeIDE/Prog/App/app.c](#)
User program application source file.
- file [ARMCM7_STM32H7_Nucleo_H743ZI_CubeIDE/Prog/App/app.h](#)
User program application header file.
- file [Demo/ARMCM7_STM32H7_Nucleo_H743ZI_CubeIDE/Prog/App/boot.c](#)
Demo program bootloader interface source file.
- file [Demo/ARMCM7_STM32H7_Nucleo_H743ZI_CubeIDE/Prog/App/boot.h](#)
Demo program bootloader interface header file.
- file [ARMCM7_STM32H7_Nucleo_H743ZI_CubeIDE/Prog/App/header.h](#)
Generic header file.
- file [ARMCM7_STM32H7_Nucleo_H743ZI_CubeIDE/Prog/App/led.c](#)
LED driver source file.
- file [ARMCM7_STM32H7_Nucleo_H743ZI_CubeIDE/Prog/App/led.h](#)
LED driver header file.
- file [Demo/ARMCM7_STM32H7_Nucleo_H743ZI_CubeIDE/Prog/App/timer.c](#)
Timer driver source file.
- file [Demo/ARMCM7_STM32H7_Nucleo_H743ZI_CubeIDE/Prog/App/timer.h](#)
Timer driver header file.

5.243.1 Detailed Description

User Program.

The intention of the demo user program is two-fold. (1) To test the bootloader, you need some sort of firmware to see if you can perform a firmware update with the bootloader. This program can be used for this purpose. (2) To make firmware programmable by the bootloader, a few adjustments to the firmware are required. The demo user program serves as an example for how these adjustments can be implemented. Additional details on this subject can be found in the port specifics documentation, which is available at: https://www.feaser.com/openblt/doku.php?id=manual:ports:armcm7_stm32h7.

5.244 Bootloader

Bootloader.

Files

- file [ARMCM7_STM32H7_Nucleo_H743ZI_GCC/Boot/blt_conf.h](#)
Bootloader configuration header file.
- file [ARMCM7_STM32H7_Nucleo_H743ZI_GCC/Boot/hooks.c](#)
Bootloader callback source file.
- file [ARMCM7_STM32H7_Nucleo_H743ZI_GCC/Boot/led.c](#)
LED driver source file.
- file [ARMCM7_STM32H7_Nucleo_H743ZI_GCC/Boot/led.h](#)
LED driver header file.
- file [ARMCM7_STM32H7_Nucleo_H743ZI_GCC/Boot/main.c](#)
Bootloader application source file.

5.244.1 Detailed Description

Bootloader.

5.245 Demo for Nucleo-H743ZI/GCC

Preconfigured programs for the Nucleo-H743ZI board and the GNU ARM GCC compiler.

Modules

- [Bootloader](#)
Bootloader.
- [User Program](#)
User Program.

5.245.1 Detailed Description

Preconfigured programs for the Nucleo-H743ZI board and the GNU ARM GCC compiler.

For detailed getting started instructions, refer to: https://www.feaser.com/openblt/doku.php?id=manual:demos:nucleo_h743zi_gcc.

5.246 User Program

User Program.

Files

- file [Demo/ARMCM7_STM32H7_Nucleo_H743ZI_GCC/Prog/boot.c](#)
Demo program bootloader interface source file.
- file [Demo/ARMCM7_STM32H7_Nucleo_H743ZI_GCC/Prog/boot.h](#)
Demo program bootloader interface header file.
- file [ARMCM7_STM32H7_Nucleo_H743ZI_GCC/Prog/header.h](#)
Generic header file.
- file [ARMCM7_STM32H7_Nucleo_H743ZI_GCC/Prog/led.c](#)
LED driver source file.
- file [ARMCM7_STM32H7_Nucleo_H743ZI_GCC/Prog/led.h](#)
LED driver header file.
- file [ARMCM7_STM32H7_Nucleo_H743ZI_GCC/Prog/main.c](#)
Demo program application source file.
- file [Demo/ARMCM7_STM32H7_Nucleo_H743ZI_GCC/Prog/timer.c](#)
Timer driver source file.
- file [Demo/ARMCM7_STM32H7_Nucleo_H743ZI_GCC/Prog/timer.h](#)
Timer driver header file.

5.246.1 Detailed Description

User Program.

The intention of the demo user program is two-fold. (1) To test the bootloader, you need some sort of firmware to see if you can perform a firmware update with the bootloader. This program can be used for this purpose. (2) To make firmware programmable by the bootloader, a few adjustments to the firmware are required. The demo user program serves as an example for how these adjustments can be implemented. Additional details on this subject can be found in the port specifics documentation, which is available at: https://www.feaser.com/openblt/doku.php?id=manual:ports:armcm7_stm32h7.

5.247 Bootloader

Bootloader.

Files

- file [ARMCM7_STM32H7_Nucleo_H743ZI_IAR/Boot/blt_conf.h](#)
Bootloader configuration header file.
- file [ARMCM7_STM32H7_Nucleo_H743ZI_IAR/Boot/hooks.c](#)
Bootloader callback source file.
- file [ARMCM7_STM32H7_Nucleo_H743ZI_IAR/Boot/led.c](#)
LED driver source file.
- file [ARMCM7_STM32H7_Nucleo_H743ZI_IAR/Boot/led.h](#)
LED driver header file.
- file [ARMCM7_STM32H7_Nucleo_H743ZI_IAR/Boot/main.c](#)
Bootloader application source file.

5.247.1 Detailed Description

Bootloader.

5.248 Demo for Nucleo-H743ZI/IAR

Preconfigured programs for the Nucleo-H743ZI board and the IAR Embedded Workbench IDE.

Modules

- [Bootloader](#)
Bootloader.
- [User Program](#)
User Program.

5.248.1 Detailed Description

Preconfigured programs for the Nucleo-H743ZI board and the IAR Embedded Workbench IDE.

For detailed getting started instructions, refer to: https://www.feaser.com/openblt/doku.php?id=manual:demos:nucleo_h743zi_iar.

5.249 User Program

User Program.

Files

- file [Demo/ARMCM7_STM32H7_Nucleo_H743ZI_IAR/Prog/boot.c](#)
Demo program bootloader interface source file.
- file [Demo/ARMCM7_STM32H7_Nucleo_H743ZI_IAR/Prog/boot.h](#)
Demo program bootloader interface header file.
- file [ARMCM7_STM32H7_Nucleo_H743ZI_IAR/Prog/header.h](#)
Generic header file.
- file [ARMCM7_STM32H7_Nucleo_H743ZI_IAR/Prog/led.c](#)
LED driver source file.
- file [ARMCM7_STM32H7_Nucleo_H743ZI_IAR/Prog/led.h](#)
LED driver header file.
- file [ARMCM7_STM32H7_Nucleo_H743ZI_IAR/Prog/main.c](#)
Demo program application source file.
- file [Demo/ARMCM7_STM32H7_Nucleo_H743ZI_IAR/Prog/timer.c](#)
Timer driver source file.
- file [Demo/ARMCM7_STM32H7_Nucleo_H743ZI_IAR/Prog/timer.h](#)
Timer driver header file.

5.249.1 Detailed Description

User Program.

The intention of the demo user program is two-fold. (1) To test the bootloader, you need some sort of firmware to see if you can perform a firmware update with the bootloader. This program can be used for this purpose. (2) To make firmware programmable by the bootloader, a few adjustments to the firmware are required. The demo user program serves as an example for how these adjustments can be implemented. Additional details on this subject can be found in the port specifics documentation, which is available at: https://www.feaser.com/openblt/doku.php?id=manual:ports:armcm7_stm32h7.

5.250 Bootloader

Bootloader.

Files

- file [ARMCM7_STM32H7_Nucleo_H743ZI_Keil/Boot/blt_conf.h](#)
Bootloader configuration header file.
- file [ARMCM7_STM32H7_Nucleo_H743ZI_Keil/Boot/hooks.c](#)
Bootloader callback source file.
- file [ARMCM7_STM32H7_Nucleo_H743ZI_Keil/Boot/led.c](#)
LED driver source file.
- file [ARMCM7_STM32H7_Nucleo_H743ZI_Keil/Boot/led.h](#)
LED driver header file.
- file [ARMCM7_STM32H7_Nucleo_H743ZI_Keil/Boot/main.c](#)
Bootloader application source file.

5.250.1 Detailed Description

Bootloader.

5.251 Demo for Nucleo-H743ZI/Keil

Preconfigured programs for the Nucleo-H743ZI board and the Keil MDK for ARM IDE.

Modules

- [Bootloader](#)
Bootloader.
- [User Program](#)
User Program.

5.251.1 Detailed Description

Preconfigured programs for the Nucleo-H743ZI board and the Keil MDK for ARM IDE.

For detailed getting started instructions, refer to: https://www.feaser.com/openblt/doku.php?id=manual:demos:nucleo_h743zi_keil.

5.252 User Program

User Program.

Files

- file [Demo/ARMCM7_STM32H7_Nucleo_H743ZI_Keil/Prog/boot.c](#)
Demo program bootloader interface source file.
- file [Demo/ARMCM7_STM32H7_Nucleo_H743ZI_Keil/Prog/boot.h](#)
Demo program bootloader interface header file.
- file [ARMCM7_STM32H7_Nucleo_H743ZI_Keil/Prog/header.h](#)
Generic header file.
- file [ARMCM7_STM32H7_Nucleo_H743ZI_Keil/Prog/led.c](#)
LED driver source file.
- file [ARMCM7_STM32H7_Nucleo_H743ZI_Keil/Prog/led.h](#)
LED driver header file.
- file [ARMCM7_STM32H7_Nucleo_H743ZI_Keil/Prog/main.c](#)
Demo program application source file.
- file [Demo/ARMCM7_STM32H7_Nucleo_H743ZI_Keil/Prog/timer.c](#)
Timer driver source file.
- file [Demo/ARMCM7_STM32H7_Nucleo_H743ZI_Keil/Prog/timer.h](#)
Timer driver header file.

5.252.1 Detailed Description

User Program.

The intention of the demo user program is two-fold. (1) To test the bootloader, you need some sort of firmware to see if you can perform a firmware update with the bootloader. This program can be used for this purpose. (2) To make firmware programmable by the bootloader, a few adjustments to the firmware are required. The demo user program serves as an example for how these adjustments can be implemented. Additional details on this subject can be found in the port specifics documentation, which is available at: https://www.feaser.com/openblt/doku.php?id=manual:ports:armcm7_stm32h7.

5.253 Bootloader Demos

Preconfigured demo programs.

Modules

- [Template for demo programs](#)
Template for creating OpenBLT demo programs.
- [Demo for S32K118EVB/GCC](#)
Preconfigured programs for the NXP S32K118EVB board and the S32 Design Studio development environment, which is based on the ARM GCC toolchain.
- [Demo for S32K118EVB/IAR](#)
Preconfigured programs for the NXP S32K118EVB board and the IAR Embedded Workbench for ARM.
- [Demo for STM32F0-Discovery/STM32CubeIDE](#)
Preconfigured programs for the STM32F0-Discovery board and the ST STM32CubeIDE toolchain.
- [Demo for STM32F0-Discovery/GCC](#)
Preconfigured programs for the STM32F0-Discovery board and the GCC compiler.
- [Demo for STM32F0-Discovery/IAR](#)
Preconfigured programs for the STM32F0-Discovery board and the IAR Embedded Workbench IDE.
- [Demo for STM32F0-Discovery/Keil](#)
Preconfigured programs for the STM32F0-Discovery board and the Keil MDK toolchain.
- [Demo for Nucleo-F091RC/STM32CubeIDE](#)
Preconfigured programs for the Nucleo-F091RC and the ST STM32CubeIDE toolchain.
- [Demo for Nucleo-F091RC/GCC](#)
Preconfigured programs for the Nucleo-F091RC and the GNU ARM GCC compiler.
- [Demo for Nucleo-F091RC/IAR](#)
Preconfigured programs for the Nucleo-F091RC board and the IAR Embedded Workbench IDE.
- [Demo for Nucleo-F091RC/Keil](#)
Preconfigured programs for the Nucleo-F091RC and the Keil MDK toolchain.
- [Demo for Nucleo-G071RB/STM32CubeIDE](#)
Preconfigured programs for the Nucleo-G071RB and the ST STM32CubeIDE toolchain.
- [Demo for Nucleo-G071RB/GCC](#)
Preconfigured programs for the Nucleo-G071RB and the GNU ARM GCC compiler.
- [Demo for Nucleo-G071RB/IAR](#)
Preconfigured programs for the Nucleo-G071RB and the IAR Embedded Workbench IDE.
- [Demo for Nucleo-G071RB/Keil](#)
Preconfigured programs for the Nucleo-G071RB and the Keil MDK toolchain.
- [Demo for XMC1400 Boot Kit/GCC](#)
Preconfigured programs for the Infineon XMC1400 Boot Kit board and the Dave 4 development environment, which is based on the ARM GCC toolchain.
- [Demo for XMC1400 Boot Kit/IAR](#)
Preconfigured programs for the Infineon XMC1400 Boot Kit board and the IAR compiler.
- [Demo for Nucleo-L552ZE/STM32CubeIDE](#)
Preconfigured programs for the Nucleo-L552ZE board and the ST STM32CubeIDE toolchain.
- [Demo for Nucleo-L552ZE/GCC](#)
Preconfigured programs for the Nucleo-L552ZE board and the GNU ARM GCC compiler.
- [Demo for Nucleo-L552ZE/IAR](#)
Preconfigured programs for the Nucleo-L552ZE board and the IAR Embedded Workbench IDE.
- [Demo for Nucleo-L552ZE/Keil](#)
Preconfigured programs for the Nucleo-L552ZE board and the Keil MDK for ARM IDE.

- [Demo for Olimex EM-32G880F128-STK/GCC](#)
Preconfigured programs for the Olimex EM-32G880F128-STK and the GCC compiler.
- [Demo for Olimex EM-32G880F128-STK/IAR](#)
Preconfigured programs for the Olimex EM-32G880F128-STK and the IAR Embedded Workbench IDE.
- [Demo for Texas Instruments EK-LM3S6965/GCC](#)
Preconfigured programs for the Texas Instruments EK-LM3S6965 and the GCC compiler.
- [Demo for Texas Instruments EK-LM3S6965/IAR](#)
Preconfigured programs for the Texas Instruments EK-LM3S6965 and the IAR Embedded Workbench IDE.
- [Demo for Texas Instruments EK-LM3S8962/GCC](#)
Preconfigured programs for the Texas Instruments EK-LM3S8962 and the GCC compiler.
- [Demo for Texas Instruments EK-LM3S8962/IAR](#)
Preconfigured programs for the Texas Instruments EK-LM3S8962 and the IAR Embedded Workbench IDE.
- [Demo for Nucleo-F103RB/STM32CubeIDE](#)
Preconfigured programs for the Nucleo-F103RB and the ST STM32CubeIDE toolchain.
- [Demo for Nucleo-F103RB/GCC](#)
Preconfigured programs for the Nucleo-F103RB and the GNU ARM GCC compiler.
- [Demo for Nucleo-F103RB](#)
Preconfigured programs for the Nucleo-F103RB and the IAR Embedded Workbench IDE.
- [Demo for Nucleo-F103RB/Keil](#)
Preconfigured programs for the Nucleo-F103RB and the Keil MDK toolchain.
- [Demo for Olimex STM32-H103/STM32CubeIDE](#)
Preconfigured programs for the Olimex STM32-H103 and the ST STM32CubeIDE toolchain.
- [Demo for Olimex STM32-H103/GCC](#)
Preconfigured programs for the Olimex STM32-H103 and the GCC compiler.
- [Demo for Olimex STM32-H103/IAR](#)
Preconfigured programs for the Olimex STM32-H103 and the IAR Embedded Workbench IDE.
- [Demo for Olimex STM32-H103/Keil](#)
Preconfigured programs for the Olimex STM32-H103 and the Keil MDK for ARM IDE.
- [Demo for Olimex STM32-P103/STM32CubeIDE](#)
Preconfigured programs for the Olimex STM32-P103 and the ST STM32CubeIDE toolchain.
- [Demo for Olimex STM32-P103/GCC](#)
Preconfigured programs for the Olimex STM32-P103 and the GCC compiler.
- [Demo for Olimex STM32-P103/IAR](#)
Preconfigured programs for the Olimex STM32-P103 and the IAR Embedded Workbench IDE.
- [Demo for Olimex STM32-P103/Keil](#)
Preconfigured programs for the Olimex STM32-P103 and the Keil MDK toolchain.
- [Demo for Olimexino-STM32/STM32CubeIDE](#)
Preconfigured programs for the Olimexino-STM32 and the ST STM32CubeIDE toolchain.
- [Demo for Olimexino-STM32/GCC](#)
Preconfigured programs for the Olimexino-STM32 and the GCC compiler.
- [Demo for Olimexino-STM32/IAR](#)
Preconfigured programs for the Olimexino-STM32 and the IAR Embedded Workbench.
- [Demo for Olimexino-STM32/Keil](#)
Preconfigured programs for the Olimexino-STM32 and the Keil MDK toolchain.
- [Demo for Olimex STM32-P207/STM32CubeIDE](#)
Preconfigured programs for the Olimex STM32-P207 and the STM32CubeIDE toolchain.
- [Demo for Olimex STM32-P207/GCC](#)
Preconfigured programs for the Olimex STM32-P207 and the GCC compiler.
- [Demo for Olimex STM32-P207/IAR](#)
Preconfigured programs for the Olimex STM32-P207 and the IAR Embedded Workbench.
- [Demo for Olimex STM32-P207/Keil](#)

- Preconfigured programs for the Olimex STM32-P207 and the Keil MDK toolchain.*

 - [Demo for S32K144EVB/GCC](#)

Preconfigured programs for the NXP S32K144EVB board and the S32 Design Studio development environment, which is based on the ARM GCC toolchain.
- [Demo for S32K144EVB/IAR](#)

Preconfigured programs for the NXP S32K144EVB board and the IAR Embedded Workbench for ARM.
- [Demo for STM32F3-Discovery/STM32CubeIDE](#)

Preconfigured programs for the STM32F3-Discovery board and the ST STM32CubeIDE toolchain.
- [Demo for STM32F3-Discovery/GCC](#)

Preconfigured programs for the STM32F3-Discovery board and the ARM GNU Embedded toolchain.
- [Demo for STM32F3-Discovery/IAR](#)

Preconfigured programs for the STM32F3-Discovery board and the IAR Embedded Workbench.
- [Demo for STM32F3-Discovery/Keil](#)

Preconfigured programs for the STM32F3-Discovery board and the Keil MDK for ARM IDE.
- [Demo for Nucleo-F303K8/STM32CubeIDE](#)

Preconfigured programs for the Nucleo-F303K8 and the ST STM32CubeIDE toolchain.
- [Demo for Nucleo-F303K8/GCC](#)

Preconfigured programs for the Nucleo-F303K8 and the ARM GNU Embedded toolchain.
- [Demo for Nucleo-F303K8/IAR](#)

Preconfigured programs for the Nucleo-F303K8 and the IAR Embedded Workbench IDE.
- [Demo for Nucleo-F303K8/Keil](#)

Preconfigured programs for the Nucleo-F303K8 and the Keil MDK toolchain.
- [Demo for Nucleo-F429ZI/STM32CubeIDE](#)

Preconfigured programs for the Nucleo-F429ZI board and the ST STM32CubeIDE toolchain.
- [Demo for Nucleo-F429ZI/GCC](#)

Preconfigured programs for the Nucleo-F429ZI board and the GNU ARM GCC compiler.
- [Demo for Nucleo-F429ZI/IAR](#)

Preconfigured programs for the Nucleo-F429ZI board and the IAR Embedded Workbench IDE.
- [Demo for Nucleo-F429ZI/Keil](#)

Preconfigured programs for the Nucleo-F429ZI board and the Keil MDK toolchain.
- [Demo for Olimex STM32-P405/STM32CubeIDE](#)

Preconfigured programs for the Olimex STM32-P405 and the ST STM32CubeIDE toolchain.
- [Demo for Olimex STM32-P405/GCC](#)

Preconfigured programs for the Olimex STM32-P405 and the GCC compiler.
- [Demo for Olimex STM32-P405/IAR](#)

Preconfigured programs for the Olimex STM32-P405 and the IAR Embedded Workbench IDE.
- [Demo for Olimex STM32-P405/Keil](#)

Preconfigured programs for the Olimex STM32-P405 and the Keil MDK for ARM IDE.
- [Demo for Nucleo-L476RG/STM32CubeIDE](#)

Preconfigured programs for the Nucleo-L476RG and the STM32CubeIDE toolchain.
- [Demo for Nucleo-L476RG/GCC](#)

Preconfigured programs for the Nucleo-L476RG and the GNU ARM GCC compiler.
- [Demo for Nucleo-L476RG/IAR](#)

Preconfigured programs for the Nucleo-L476RG and the IAR Embedded Workbench IDE.
- [Demo for Nucleo-L476RG/Keil](#)

Preconfigured programs for the Nucleo-L476RG and the Keil MDK toolchain.
- [Demo for Texas Instruments DK-TM4C123G/IAR](#)

Preconfigured programs for the Texas Instruments DK-TM4C123G and the IAR Embedded Workbench IDE.
- [Demo for XMC4700 Relax Kit/GCC](#)

Preconfigured programs for the Infineon XMC4700 Relax Kit board and the Dave 4 development environment, which is based on the ARM GCC toolchain.

- [Demo for XMC4700 Relax Kit/IAR](#)

Preconfigured programs for the Infineon XMC4700 Relax Kit board and the IAR compiler.

- [Demo for Nucleo-F746ZG/STM32CubeIDE](#)

Preconfigured programs for the Nucleo-F746ZG board and the ST STM32CubeIDE toolchain.

- [Demo for Nucleo-F746ZG/GCC](#)

Preconfigured programs for the Nucleo-F746ZG board and the GNU ARM GCC compiler.

- [Demo for Nucleo-F746ZG/IAR](#)

Preconfigured programs for the Nucleo-F746ZG board and the IAR Embedded Workbench IDE.

- [Demo for Nucleo-F746ZG/Keil](#)

Preconfigured programs for the Nucleo-F746ZG board and the Keil MDK for ARM IDE.

- [Demo for Nucleo-F767ZI/STM32CubeIDE](#)

Preconfigured programs for the Nucleo-F767ZI board and the ST STM32CubeIDE toolchain.

- [Demo for Nucleo-F767ZI/GCC](#)

Preconfigured programs for the Nucleo-F767ZI board and the GNU ARM GCC compiler.

- [Demo for Nucleo-F767ZI/IAR](#)

Preconfigured programs for the Nucleo-F767ZI board and the IAR Embedded Workbench IDE.

- [Demo for Nucleo-F767ZI/Keil](#)

Preconfigured programs for the Nucleo-F767ZI board and the Keil MDK toolchain.

- [Demo for Nucleo-H743ZI/STM32CubeIDE](#)

Preconfigured programs for the Nucleo-H743ZI board and the ST STM32CubeIDE toolchain.

- [Demo for Nucleo-H743ZI/GCC](#)

Preconfigured programs for the Nucleo-H743ZI board and the GNU ARM GCC compiler.

- [Demo for Nucleo-H743ZI/IAR](#)

Preconfigured programs for the Nucleo-H743ZI board and the IAR Embedded Workbench IDE.

- [Demo for Nucleo-H743ZI/Keil](#)

Preconfigured programs for the Nucleo-H743ZI board and the Keil MDK for ARM IDE.

- [Demo for NXP DevKit-S12G128/CodeWarrior](#)

Preconfigured programs for the NXP DevKit-S12G128 and the CodeWarrior IDE.

- [Demo for Dragon12-plus/CodeWarrior](#)

Preconfigured programs for the EVBplus Dragon12-plus and the CodeWarrior IDE.

5.253.1 Detailed Description

Preconfigured demo programs.

Each demo consists of a bootloader and user program. The bootloader can be tested by performing a firmware update using the supplied user program. Refer to <https://www.feaser.com/openblt/doku.php?id=manual:demos> for an overview of all available demo programs.

The easiest way to get started with the bootloader is by obtaining a low cost development board with on-chip debugger, for which a demo is available. For example one of the ST STM32 Nucleo boards. Getting to know that bootloader and how it works is much easier once you have it running. Then porting it to your own microcontroller system will go smoother. If you'd like assistance or outsource the integration of the bootloader on your system, feel free to contact Feaser (<http://www.feaser.com>).

5.254 Bootloader

Bootloader.

Files

- file [HCS12_DevKit_S12G128_CodeWarrior/Boot/blt_conf.h](#)
Bootloader configuration header file.
- file [HCS12_DevKit_S12G128_CodeWarrior/Boot/hooks.c](#)
Bootloader callback source file.
- file [HCS12_DevKit_S12G128_CodeWarrior/Boot/led.c](#)
LED driver source file.
- file [HCS12_DevKit_S12G128_CodeWarrior/Boot/led.h](#)
LED driver header file.
- file [HCS12_DevKit_S12G128_CodeWarrior/Boot/main.c](#)
Demo program application source file.
- file [HCS12_DevKit_S12G128_CodeWarrior/Boot/vectors.c](#)
Demo program interrupt vectors source file.

5.254.1 Detailed Description

Bootloader.

5.255 Demo for NXP DevKit-S12G128/CodeWarrior

Preconfigured programs for the NXP DevKit-S12G128 and the CodeWarrior IDE.

Modules

- [Bootloader](#)
Bootloader.
- [User Program](#)
User Program.

5.255.1 Detailed Description

Preconfigured programs for the NXP DevKit-S12G128 and the CodeWarrior IDE.

For detailed getting started instructions, refer to: https://www.feaser.com/openblt/doku.php?id=manual:demos:devkit_s12g128_codewarrior.

5.256 User Program

User Program.

Files

- file [HCS12_DevKit_S12G128_CodeWarrior/Prog/header.h](#)
Generic header file.
- file [HCS12_DevKit_S12G128_CodeWarrior/Prog/led.c](#)
LED driver source file.
- file [HCS12_DevKit_S12G128_CodeWarrior/Prog/led.h](#)
LED driver header file.
- file [HCS12_DevKit_S12G128_CodeWarrior/Prog/main.c](#)
Demo program application source file.
- file [Demo/HCS12_DevKit_S12G128_CodeWarrior/Prog/timer.c](#)
Timer driver source file.
- file [Demo/HCS12_DevKit_S12G128_CodeWarrior/Prog/timer.h](#)
Timer driver header file.
- file [HCS12_DevKit_S12G128_CodeWarrior/Prog/vectors.c](#)
Demo program interrupt vectors source file.

5.256.1 Detailed Description

User Program.

The intention of the demo user program is two-fold. (1) To test the bootloader, you need some sort of firmware to see if you can perform a firmware update with the bootloader. This program can be used for this purpose. (2) To make firmware programmable by the bootloader, a few adjustments to the firmware are required. The demo user program serves as an example for how these adjustments can be implemented. Additional details on this subject can be found in the port specifics documentation, which is available at: <https://www.feaser.com/openblt/doku.php?id=manual:ports:hcs12>.

5.257 Bootloader

Bootloader.

Files

- file [HCS12_Evbplus_Dragon12p_CodeWarrior/Boot/blt_conf.h](#)
Bootloader configuration header file.
- file [HCS12_Evbplus_Dragon12p_CodeWarrior/Boot/hooks.c](#)
Bootloader callback source file.
- file [HCS12_Evbplus_Dragon12p_CodeWarrior/Boot/main.c](#)
Bootloader application source file.
- file [HCS12_Evbplus_Dragon12p_CodeWarrior/Boot/vectors.c](#)
Bootloader interrupt vector table source file.

5.257.1 Detailed Description

Bootloader.

5.258 Demo for Dragon12-plus/CodeWarrior

Preconfigured programs for the EVBplus Dragon12-plus and the CodeWarrior IDE.

Modules

- [Bootloader](#)
Bootloader.
- [User Program](#)
User Program.

5.258.1 Detailed Description

Preconfigured programs for the EVBplus Dragon12-plus and the CodeWarrior IDE.

For detailed getting started instructions, refer to: https://www.feaser.com/openblt/doku.php?id=manual:demos:evbplus_dragon12p_codewarrior.

5.259 User Program

User Program.

Files

- file [Demo/HCS12_Evbplus_Dragon12p_CodeWarrior/Prog/boot.c](#)
Demo program bootloader interface source file.
- file [Demo/HCS12_Evbplus_Dragon12p_CodeWarrior/Prog/boot.h](#)
Demo program bootloader interface header file.
- file [Demo/HCS12_Evbplus_Dragon12p_CodeWarrior/Prog/boot.c](#)
Demo program bootloader interface source file.
- file [Demo/HCS12_Evbplus_Dragon12p_CodeWarrior/Prog/boot.h](#)
Demo program bootloader interface header file.
- file [HCS12_Evbplus_Dragon12p_CodeWarrior/Prog/header.h](#)
Generic header file.
- file [irq.c](#)
IRQ driver source file.
- file [irq.h](#)
IRQ driver header file.
- file [HCS12_Evbplus_Dragon12p_CodeWarrior/Prog/led.c](#)
LED driver source file.
- file [HCS12_Evbplus_Dragon12p_CodeWarrior/Prog/led.h](#)
LED driver header file.
- file [HCS12_Evbplus_Dragon12p_CodeWarrior/Prog/main.c](#)
Demo program application source file.
- file [Demo/HCS12_Evbplus_Dragon12p_CodeWarrior/Prog/timer.c](#)
Timer driver source file.
- file [Demo/HCS12_Evbplus_Dragon12p_CodeWarrior/Prog/timer.h](#)
Timer driver header file.
- file [HCS12_Evbplus_Dragon12p_CodeWarrior/Prog/vectors.c](#)
Demo program interrupt vectors source file.

5.259.1 Detailed Description

User Program.

The intention of the demo user program is two-fold. (1) To test the bootloader, you need some sort of firmware to see if you can perform a firmware update with the bootloader. This program can be used for this purpose. (2) To make firmware programmable by the bootloader, a few adjustments to the firmware are required. The demo user program serves as an example for how these adjustments can be implemented. Additional details on this subject can be found in the port specifics documentation, which is available at: <https://www.feaser.com/openblt/doku.php?id=manual:ports:hcs12>.

5.260 CAN driver of a port

This module implements the CAN driver of a microcontroller port.

Files

- file [_template/can.c](#)
Bootloader CAN communication interface source file.

5.260.1 Detailed Description

This module implements the CAN driver of a microcontroller port.

For the most parts, this driver is already implemented. The only parts that need porting are the CAN initialization, CAN message reception and CAN message transmission.

5.261 CPU driver of a port

This module implements the CPU driver of a microcontroller port.

Files

- file [_template/cpu.c](#)
Bootloader cpu module source file.

5.261.1 Detailed Description

This module implements the CPU driver of a microcontroller port.

5.262 Flash driver of a port

This module implements the flash EEPROM memory driver of a microcontroller port.

Files

- file [_template/flash.c](#)
Bootloader flash driver source file.
- file [_template/flash.h](#)
Bootloader flash driver header file.

5.262.1 Detailed Description

This module implements the flash EEPROM memory driver of a microcontroller port.

The flash driver manages the actual erase and program operations on the flash EEPROM and the signature checksum. The signature checksum is a 32-bit value in the user program. It is used as a flag to determine if a user program is present or not. Newly programmed data is always first buffered in RAM buffers with a size of FLASH_WRITE_BLOCK_SIZE. This driver manages a second RAM buffer of the same size, called the bootBlock. The bootBlock buffers program data that includes the interrupt vector table and the 32-bit signature checksum. The signature checksum value is written as the last step during a firmware update, hence the need for the bootBlock. Note that the majority of this flash driver can be used as is. The only parts that need to be updated / implemented are:

- Macros FLASH_WRITE_BLOCK_SIZE and BOOT_FLASH_VECTOR_TABLE_CS_OFFSET.
- The flashLayout[]-array contents.
- The functions [FlashEraseSectors\(\)](#) and [FlashWriteBlock\(\)](#).
- The functions [FlashWriteChecksum\(\)](#) and [FlashVerifyChecksum\(\)](#).

5.263 Compiler specifics of a port

This module implements the compiler specific parts of a microcontroller port.

Files

- file [_template/GCC/cpu_comp.c](#)
Bootloader cpu module source file.

5.263.1 Detailed Description

This module implements the compiler specific parts of a microcontroller port.

5.264 Non-volatile memory driver of a port

This module implements the non-volatile memory driver of a microcontroller port. Note that the default implementation is for a microcontroller that has internal flash memory. At the time of this writing pretty much all microcontrollers use flash EEPROM as non-volatile memory to store the program code. Assuming that this is also the case for the microcontroller for which the port is developed, nothing needs to be modified in this source file.

Files

- file [_template/nvm.c](#)

Bootloader non-volatile memory driver source file.

5.264.1 Detailed Description

This module implements the non-volatile memory driver of a microcontroller port. Note that the default implementation is for a microcontroller that has internal flash memory. At the time of this writing pretty much all microcontrollers use flash EEPROM as non-volatile memory to store the program code. Assuming that this is also the case for the microcontroller for which the port is developed, nothing needs to be modified in this source file.

5.265 RS232 UART driver of a port

This module implements the RS232 UART driver of a microcontroller port.

Files

- file [_template/rs232.c](#)
Bootloader RS232 communication interface source file.

5.265.1 Detailed Description

This module implements the RS232 UART driver of a microcontroller port.

For the most parts, this driver is already implemented. The only parts that need porting are the UART initialization, byte reception and byte transmission.

5.266 Target Port Template

Target dependent code as a template for new microcontroller ports.

Modules

- [CAN driver of a port](#)
This module implements the CAN driver of a microcontroller port.
- [CPU driver of a port](#)
This module implements the CPU driver of a microcontroller port.
- [Flash driver of a port](#)
This module implements the flash EEPROM memory driver of a microcontroller port.
- [Compiler specifics of a port](#)
This module implements the compiler specific parts of a microcontroller port.
- [Non-volatile memory driver of a port](#)
This module implements the non-volatile memory driver of a microcontroller port. Note that the default implementation is for a microcontroller that has internal flash memory. At the time of this writing pretty much all microcontrollers use flash EEPROM as non-volatile memory to store the program code. Assuming that this is also the case for the microcontroller for which the port is developed, nothing needs to be modified in this source file.
- [RS232 UART driver of a port](#)
This module implements the RS232 UART driver of a microcontroller port.
- [Timer driver of a port](#)
This module implements the timer memory driver of a microcontroller port.
- [Type definitions of a port](#)
This module implements the variable type definitions of a microcontroller port.
- [USB driver of a port](#)
This module implements the USB driver of a microcontroller port.

5.266.1 Detailed Description

Target dependent code as a template for new microcontroller ports.

This module serves as a template to implement the bootloader's target dependent part for a specific microcontroller family. It can be copied and used as a foundation when developing new microcontroller ports. Note that the parts of a port that need to be implemented are described as a source code comment that starts with "TODO ##Port".

5.267 Timer driver of a port

This module implements the timer memory driver of a microcontroller port.

Files

- file [Source/_template/timer.c](#)
Bootloader timer driver source file.

5.267.1 Detailed Description

This module implements the timer memory driver of a microcontroller port.

The timer driver implements a polling based 1 millisecond timer. It provides the time base for all timing related parts of the bootloader. The bootloader calls the function [TimerUpdate\(\)](#) continuously to check if the next millisecond period passed.

5.268 Type definitions of a port

This module implements the variable type definitions of a microcontroller port.

Files

- file [_template/types.h](#)
Bootloader types header file.

5.268.1 Detailed Description

This module implements the variable type definitions of a microcontroller port.

5.269 USB driver of a port

This module implements the USB driver of a microcontroller port.

Files

- file [_template/usb.c](#)
Bootloader USB communication interface source file.

5.269.1 Detailed Description

This module implements the USB driver of a microcontroller port.

The USB driver makes user of bulk communications only, by means of two endpoints. This driver already implements FIFO buffers for the endpoint data including utility functions for managing these FIFOs. Note that the USB descriptor configuration and other enumeration related configuration is typically stored with the bootloader demo application and is not implemented in this driver directly. Basically, whenever the USB communication stack is available to transmit new data on the IN-endpoint, function [UsbTransmitPipeBulkIN\(\)](#) can be called, which checks if there is data to transmit in the FIFO buffer and then needs to copy it to the IN-endpoint buffer. Whenever the USB communication stack signals that new data was received on the OUT-endpoint, [UsbReceivePipeBulkOUT\(\)](#) can be called which then copies the data from the OUT-endpoint buffer into the FIFO buffer.

5.270 Target ARMCM0 S32K11

Target dependent code for the NXP ARMCM0 S32K11x microcontroller family.

Files

- file [ARMCM0_S32K11/can.c](#)
Bootloader CAN communication interface source file.
- file [ARMCM0_S32K11/cpu.c](#)
Bootloader cpu module source file.
- file [ARMCM0_S32K11/flash.c](#)
Bootloader flash driver source file.
- file [ARMCM0_S32K11/flash.h](#)
Bootloader flash driver header file.
- file [ARMCM0_S32K11/GCC/cpu_comp.c](#)
Bootloader cpu module source file.
- file [ARMCM0_S32K11/IAR/cpu_comp.c](#)
Bootloader cpu module source file.
- file [ARMCM0_S32K11/nvm.c](#)
Bootloader non-volatile memory driver source file.
- file [Source/ARMCM0_S32K11/timer.c](#)
Bootloader timer driver source file.
- file [ARMCM0_S32K11/types.h](#)
Bootloader types header file.

5.270.1 Detailed Description

Target dependent code for the NXP ARMCM0 S32K11x microcontroller family.

This module implements the bootloader's target dependent part for the NXP ARMCM0 S32K11x microcontroller family.

5.271 Target ARMCM0 STM32F0

Target dependent code for the ARMCM0 STM32F0 microcontroller family.

Files

- file [ARMCM0_STM32F0/can.c](#)
Bootloader CAN communication interface source file.
- file [ARMCM0_STM32F0/cpu.c](#)
Bootloader cpu module source file.
- file [ARMCM0_STM32F0/flash.c](#)
Bootloader flash driver source file.
- file [ARMCM0_STM32F0/flash.h](#)
Bootloader flash driver header file.
- file [ARMCM0_STM32F0/GCC/cpu_comp.c](#)
Bootloader cpu module source file.
- file [ARMCM0_STM32F0/IAR/cpu_comp.c](#)
Bootloader cpu module source file.
- file [ARMCM0_STM32F0/Keil/cpu_comp.c](#)
Bootloader cpu module source file.
- file [ARMCM0_STM32F0/nvm.c](#)
Bootloader non-volatile memory driver source file.
- file [ARMCM0_STM32F0/rs232.c](#)
Bootloader RS232 communication interface source file.
- file [Source/ARMCM0_STM32F0/timer.c](#)
Bootloader timer driver source file.
- file [ARMCM0_STM32F0/types.h](#)
Bootloader types header file.

5.271.1 Detailed Description

Target dependent code for the ARMCM0 STM32F0 microcontroller family.

This module implements the bootloader's target dependent part for the ARMCM0 STM32F0 microcontroller family.

5.272 Target ARMCM0 STM32G0

Target dependent code for the ARMCM0 STM32G0 microcontroller family.

Files

- file [ARMCM0_STM32G0/cpu.c](#)
Bootloader cpu module source file.
- file [ARMCM0_STM32G0/flash.c](#)
Bootloader flash driver source file.
- file [ARMCM0_STM32G0/flash.h](#)
Bootloader flash driver header file.
- file [ARMCM0_STM32G0/GCC/cpu_comp.c](#)
Bootloader cpu module source file.
- file [ARMCM0_STM32G0/IAR/cpu_comp.c](#)
Bootloader cpu module source file.
- file [ARMCM0_STM32G0/Keil/cpu_comp.c](#)
Bootloader cpu module source file.
- file [ARMCM0_STM32G0/nvm.c](#)
Bootloader non-volatile memory driver source file.
- file [ARMCM0_STM32G0/rs232.c](#)
Bootloader RS232 communication interface source file.
- file [Source/ARMCM0_STM32G0/timer.c](#)
Bootloader timer driver source file.
- file [ARMCM0_STM32G0/types.h](#)
Bootloader types header file.

5.272.1 Detailed Description

Target dependent code for the ARMCM0 STM32G0 microcontroller family.

This module implements the bootloader's target dependent part for the ARMCM0 STM32G0 microcontroller family.

5.273 Target ARMCM0 XMC1

Target dependent code for the ARMCM0 XMC1xxx microcontroller family.

Files

- file [ARMCM0_XMC1/can.c](#)
Bootloader CAN communication interface source file.
- file [ARMCM0_XMC1/cpu.c](#)
Bootloader cpu module source file.
- file [ARMCM0_XMC1/flash.c](#)
Bootloader flash driver source file.
- file [ARMCM0_XMC1/flash.h](#)
Bootloader flash driver header file.
- file [ARMCM0_XMC1/GCC/cpu_comp.c](#)
Bootloader cpu module source file.
- file [ARMCM0_XMC1/IAR/cpu_comp.c](#)
Bootloader cpu module source file.
- file [ARMCM0_XMC1/nvm.c](#)
Bootloader non-volatile memory driver source file.
- file [ARMCM0_XMC1/rs232.c](#)
Bootloader RS232 communication interface source file.
- file [Source/ARMCM0_XMC1/timer.c](#)
Bootloader timer driver source file.
- file [ARMCM0_XMC1/types.h](#)
Bootloader types header file.

5.273.1 Detailed Description

Target dependent code for the ARMCM0 XMC1xxx microcontroller family.

This module implements the bootloader's target dependent part for the ARMCM0 XMC1xxx microcontroller family.

5.274 Target ARMCM33 STM32L5

Target dependent code for the ARMCM33 STM32L5 microcontroller family.

Files

- file [ARMCM33_STM32L5/can.c](#)
Bootloader CAN communication interface source file.
- file [ARMCM33_STM32L5/cpu.c](#)
Bootloader cpu module source file.
- file [ARMCM33_STM32L5/flash.c](#)
Bootloader flash driver source file.
- file [ARMCM33_STM32L5/flash.h](#)
Bootloader flash driver header file.
- file [ARMCM33_STM32L5/GCC/cpu_comp.c](#)
Bootloader cpu module source file.
- file [ARMCM33_STM32L5/IAR/cpu_comp.c](#)
Bootloader cpu module source file.
- file [ARMCM33_STM32L5/Keil/cpu_comp.c](#)
Bootloader cpu module source file.
- file [ARMCM33_STM32L5/nvm.c](#)
Bootloader non-volatile memory driver source file.
- file [ARMCM33_STM32L5/rs232.c](#)
Bootloader RS232 communication interface source file.
- file [Source/ARMCM33_STM32L5/timer.c](#)
Bootloader timer driver source file.
- file [ARMCM33_STM32L5/types.h](#)
Bootloader types header file.
- file [ARMCM33_STM32L5/usb.c](#)
Bootloader USB communication interface source file.

5.274.1 Detailed Description

Target dependent code for the ARMCM33 STM32L5 microcontroller family.

This module implements the bootloader's target dependent part for the ARMCM33 STM32L5 microcontroller family.

5.275 Target ARMCM3 EFM32

Target dependent code for the ARMCM3 EFM32 microcontroller family.

Files

- file [ARMCM3_EFM32/cpu.c](#)
Bootloader cpu module source file.
- file [ARMCM3_EFM32/flash.c](#)
Bootloader flash driver source file.
- file [ARMCM3_EFM32/flash.h](#)
Bootloader flash driver header file.
- file [ARMCM3_EFM32/GCC/cpu_comp.c](#)
Bootloader cpu module source file.
- file [ARMCM3_EFM32/IAR/cpu_comp.c](#)
Bootloader cpu module source file.
- file [ARMCM3_EFM32/nvm.c](#)
Bootloader non-volatile memory driver source file.
- file [ARMCM3_EFM32/rs232.c](#)
Bootloader RS232 communication interface source file.
- file [Source/ARMCM3_EFM32/timer.c](#)
Bootloader timer driver source file.
- file [ARMCM3_EFM32/types.h](#)
Bootloader types header file.

5.275.1 Detailed Description

Target dependent code for the ARMCM3 EFM32 microcontroller family.

This module implements the bootloader's target dependent part for the ARMCM3 EFM32 microcontroller family.

5.276 Target ARMCM3 LM3S

Target dependent code for the ARMCM3 LM3S microcontroller family.

Files

- file [ARMCM3_LM3S/can.c](#)
Bootloader CAN communication interface source file.
- file [ARMCM3_LM3S/cpu.c](#)
Bootloader cpu module source file.
- file [ARMCM3_LM3S/flash.c](#)
Bootloader flash driver source file.
- file [ARMCM3_LM3S/flash.h](#)
Bootloader flash driver header file.
- file [ARMCM3_LM3S/GCC/cpu_comp.c](#)
Bootloader cpu module source file.
- file [ARMCM3_LM3S/IAR/cpu_comp.c](#)
Bootloader cpu module source file.
- file [ARMCM3_LM3S/nvm.c](#)
Bootloader non-volatile memory driver source file.
- file [ARMCM3_LM3S/rs232.c](#)
Bootloader RS232 communication interface source file.
- file [Source/ARMCM3_LM3S/timer.c](#)
Bootloader timer driver source file.
- file [ARMCM3_LM3S/types.h](#)
Bootloader types header file.

5.276.1 Detailed Description

Target dependent code for the ARMCM3 LM3S microcontroller family.

This module implements the bootloader's target dependent part for the ARMCM3 LM3S microcontroller family.

5.277 Target ARMCM3 STM32F1

Target dependent code for the ARMCM3 STM32F1 microcontroller family.

Files

- file [ARMCM3_STM32F1/can.c](#)
Bootloader CAN communication interface source file.
- file [ARMCM3_STM32F1/cpu.c](#)
Bootloader cpu module source file.
- file [ARMCM3_STM32F1/flash.c](#)
Bootloader flash driver source file.
- file [ARMCM3_STM32F1/flash.h](#)
Bootloader flash driver header file.
- file [ARMCM3_STM32F1/GCC/cpu_comp.c](#)
Bootloader cpu module source file.
- file [ARMCM3_STM32F1/IAR/cpu_comp.c](#)
Bootloader cpu module source file.
- file [ARMCM3_STM32F1/Keil/cpu_comp.c](#)
Bootloader cpu module source file.
- file [ARMCM3_STM32F1/nvm.c](#)
Bootloader non-volatile memory driver source file.
- file [ARMCM3_STM32F1/rs232.c](#)
Bootloader RS232 communication interface source file.
- file [Source/ARMCM3_STM32F1/timer.c](#)
Bootloader timer driver source file.
- file [ARMCM3_STM32F1/types.h](#)
Bootloader types header file.
- file [ARMCM3_STM32F1/usb.c](#)
Bootloader USB communication interface source file.

5.277.1 Detailed Description

Target dependent code for the ARMCM3 STM32F1 microcontroller family.

This module implements the bootloader's target dependent part for the ARMCM3 STM32F1 microcontroller family.

5.278 Target ARMCM3 STM32F2

Target dependent code for the ARMCM3 STM32F2xx microcontroller family.

Files

- file [ARMCM3_STM32F2/can.c](#)
Bootloader CAN communication interface source file.
- file [ARMCM3_STM32F2/cpu.c](#)
Bootloader cpu module source file.
- file [ARMCM3_STM32F2/flash.c](#)
Bootloader flash driver source file.
- file [ARMCM3_STM32F2/flash.h](#)
Bootloader flash driver header file.
- file [ARMCM3_STM32F2/GCC/cpu_comp.c](#)
Bootloader cpu module source file.
- file [ARMCM3_STM32F2/IAR/cpu_comp.c](#)
Bootloader cpu module source file.
- file [ARMCM3_STM32F2/Keil/cpu_comp.c](#)
Bootloader cpu module source file.
- file [ARMCM3_STM32F2/nvm.c](#)
Bootloader non-volatile memory driver source file.
- file [ARMCM3_STM32F2/rs232.c](#)
Bootloader RS232 communication interface source file.
- file [Source/ARMCM3_STM32F2/timer.c](#)
Bootloader timer driver source file.
- file [ARMCM3_STM32F2/types.h](#)
Bootloader types header file.

5.278.1 Detailed Description

Target dependent code for the ARMCM3 STM32F2xx microcontroller family.

This module implements the bootloader's target dependent part for the ARMCM3 STM32F2xx microcontroller family.

5.279 Target ARMCM4 S32K14

Target dependent code for the NXP ARMCM4 S32K14x microcontroller family.

Files

- file [ARMCM4_S32K14/can.c](#)
Bootloader CAN communication interface source file.
- file [ARMCM4_S32K14/cpu.c](#)
Bootloader cpu module source file.
- file [ARMCM4_S32K14/flash.c](#)
Bootloader flash driver source file.
- file [ARMCM4_S32K14/flash.h](#)
Bootloader flash driver header file.
- file [ARMCM4_S32K14/GCC/cpu_comp.c](#)
Bootloader cpu module source file.
- file [ARMCM4_S32K14/IAR/cpu_comp.c](#)
Bootloader cpu module source file.
- file [ARMCM4_S32K14/nvm.c](#)
Bootloader non-volatile memory driver source file.
- file [Source/ARMCM4_S32K14/timer.c](#)
Bootloader timer driver source file.
- file [ARMCM4_S32K14/types.h](#)
Bootloader types header file.

5.279.1 Detailed Description

Target dependent code for the NXP ARMCM4 S32K14x microcontroller family.

This module implements the bootloader's target dependent part for the NXP ARMCM4 S32K14x microcontroller family.

5.280 Target ARMCM4 STM32F3

Target dependent code for the ARMCM4 STM32F3 microcontroller family.

Files

- file [ARMCM4_STM32F3/can.c](#)
Bootloader CAN communication interface source file.
- file [ARMCM4_STM32F3/cpu.c](#)
Bootloader cpu module source file.
- file [ARMCM4_STM32F3/flash.c](#)
Bootloader flash driver source file.
- file [ARMCM4_STM32F3/flash.h](#)
Bootloader flash driver header file.
- file [ARMCM4_STM32F3/GCC/cpu_comp.c](#)
Bootloader cpu module source file.
- file [ARMCM4_STM32F3/IAR/cpu_comp.c](#)
Bootloader cpu module source file.
- file [ARMCM4_STM32F3/Keil/cpu_comp.c](#)
Bootloader cpu module source file.
- file [ARMCM4_STM32F3/nvm.c](#)
Bootloader non-volatile memory driver source file.
- file [ARMCM4_STM32F3/rs232.c](#)
Bootloader RS232 communication interface source file.
- file [Source/ARMCM4_STM32F3/timer.c](#)
Bootloader timer driver source file.
- file [ARMCM4_STM32F3/types.h](#)
Bootloader types header file.
- file [ARMCM4_STM32F3/usb.c](#)
Bootloader USB communication interface source file.

5.280.1 Detailed Description

Target dependent code for the ARMCM4 STM32F3 microcontroller family.

This module implements the bootloader's target dependent part for the ARMCM4 STM32F3 microcontroller family.

5.281 Target ARMCM4 STM32F4

Target dependent code for the ARMCM4 STM32F4 microcontroller family.

Files

- file [ARMCM4_STM32F4/can.c](#)
Bootloader CAN communication interface source file.
- file [ARMCM4_STM32F4/cpu.c](#)
Bootloader cpu module source file.
- file [ARMCM4_STM32F4/flash.c](#)
Bootloader flash driver source file.
- file [ARMCM4_STM32F4/flash.h](#)
Bootloader flash driver header file.
- file [ARMCM4_STM32F4/GCC/cpu_comp.c](#)
Bootloader cpu module source file.
- file [ARMCM4_STM32F4/IAR/cpu_comp.c](#)
Bootloader cpu module source file.
- file [ARMCM4_STM32F4/Keil/cpu_comp.c](#)
Bootloader cpu module source file.
- file [ARMCM4_STM32F4/nvm.c](#)
Bootloader non-volatile memory driver source file.
- file [ARMCM4_STM32F4/rs232.c](#)
Bootloader RS232 communication interface source file.
- file [Source/ARMCM4_STM32F4/timer.c](#)
Bootloader timer driver source file.
- file [ARMCM4_STM32F4/types.h](#)
Bootloader types header file.
- file [ARMCM4_STM32F4/usb.c](#)
Bootloader USB communication interface source file.

5.281.1 Detailed Description

Target dependent code for the ARMCM4 STM32F4 microcontroller family.

This module implements the bootloader's target dependent part for the ARMCM4 STM32F4 microcontroller family.

5.282 Target ARMCM4 STM32L4

Target dependent code for the ARMCM4 STM32L4 microcontroller family.

Files

- file [ARMCM4_STM32L4/can.c](#)
Bootloader CAN communication interface source file.
- file [ARMCM4_STM32L4/cpu.c](#)
Bootloader cpu module source file.
- file [ARMCM4_STM32L4/flash.c](#)
Bootloader flash driver source file.
- file [ARMCM4_STM32L4/flash.h](#)
Bootloader flash driver header file.
- file [ARMCM4_STM32L4/GCC/cpu_comp.c](#)
Bootloader cpu module source file.
- file [ARMCM4_STM32L4/IAR/cpu_comp.c](#)
Bootloader cpu module source file.
- file [ARMCM4_STM32L4/Keil/cpu_comp.c](#)
Bootloader cpu module source file.
- file [ARMCM4_STM32L4/nvm.c](#)
Bootloader non-volatile memory driver source file.
- file [ARMCM4_STM32L4/rs232.c](#)
Bootloader RS232 communication interface source file.
- file [Source/ARMCM4_STM32L4/timer.c](#)
Bootloader timer driver source file.
- file [ARMCM4_STM32L4/types.h](#)
Bootloader types header file.

5.282.1 Detailed Description

Target dependent code for the ARMCM4 STM32L4 microcontroller family.

This module implements the bootloader's target dependent part for the ARMCM4 STM32L4 microcontroller family.

5.283 Target ARMCM4 TM4C

Target dependent code for the ARMCM4 TM4C microcontroller family.

Files

- file [ARMCM4_TM4C/cpu.c](#)
Bootloader cpu module source file.
- file [ARMCM4_TM4C/flash.c](#)
Bootloader flash driver source file.
- file [ARMCM4_TM4C/flash.h](#)
Bootloader flash driver header file.
- file [ARMCM4_TM4C/IAR/cpu_comp.c](#)
Bootloader cpu module source file.
- file [ARMCM4_TM4C/nvm.c](#)
Bootloader non-volatile memory driver source file.
- file [ARMCM4_TM4C/rs232.c](#)
Bootloader RS232 communication interface source file.
- file [Source/ARMCM4_TM4C/timer.c](#)
Bootloader timer driver source file.
- file [ARMCM4_TM4C/types.h](#)
Bootloader types header file.
- file [ARMCM4_TM4C/usb.c](#)
Bootloader USB communication interface source file.

5.283.1 Detailed Description

Target dependent code for the ARMCM4 TM4C microcontroller family.

This module implements the bootloader's target dependent part for the ARMCM4 TM4C microcontroller family.

5.284 Target ARMCM4 XMC4

Target dependent code for the ARMCM4 XMC4xxx microcontroller family.

Files

- file [ARMCM4_XMC4/can.c](#)
Bootloader CAN communication interface source file.
- file [ARMCM4_XMC4/cpu.c](#)
Bootloader cpu module source file.
- file [ARMCM4_XMC4/flash.c](#)
Bootloader flash driver source file.
- file [ARMCM4_XMC4/flash.h](#)
Bootloader flash driver header file.
- file [ARMCM4_XMC4/GCC/cpu_comp.c](#)
Bootloader cpu module source file.
- file [ARMCM4_XMC4/IAR/cpu_comp.c](#)
Bootloader cpu module source file.
- file [ARMCM4_XMC4/nvm.c](#)
Bootloader non-volatile memory driver source file.
- file [ARMCM4_XMC4/rs232.c](#)
Bootloader RS232 communication interface source file.
- file [Source/ARMCM4_XMC4/timer.c](#)
Bootloader timer driver source file.
- file [ARMCM4_XMC4/types.h](#)
Bootloader types header file.

5.284.1 Detailed Description

Target dependent code for the ARMCM4 XMC4xxx microcontroller family.

This module implements the bootloader's target dependent part for the ARMCM4 XMC4xxx microcontroller family.

5.285 Target ARMCM7 STM32F7

Target dependent code for the ARMCM7 STM32F7 microcontroller family.

Files

- file [ARMCM7_STM32F7/can.c](#)
Bootloader CAN communication interface source file.
- file [ARMCM7_STM32F7/cpu.c](#)
Bootloader cpu module source file.
- file [ARMCM7_STM32F7/flash.c](#)
Bootloader flash driver source file.
- file [ARMCM7_STM32F7/flash.h](#)
Bootloader flash driver header file.
- file [ARMCM7_STM32F7/GCC/cpu_comp.c](#)
Bootloader cpu module source file.
- file [ARMCM7_STM32F7/IAR/cpu_comp.c](#)
Bootloader cpu module source file.
- file [ARMCM7_STM32F7/Keil/cpu_comp.c](#)
Bootloader cpu module source file.
- file [ARMCM7_STM32F7/nvm.c](#)
Bootloader non-volatile memory driver source file.
- file [ARMCM7_STM32F7/rs232.c](#)
Bootloader RS232 communication interface source file.
- file [Source/ARMCM7_STM32F7/timer.c](#)
Bootloader timer driver source file.
- file [ARMCM7_STM32F7/types.h](#)
Bootloader types header file.
- file [ARMCM7_STM32F7/usb.c](#)
Bootloader USB communication interface source file.

5.285.1 Detailed Description

Target dependent code for the ARMCM7 STM32F7 microcontroller family.

This module implements the bootloader's target dependent part for the ARMCM7 STM32F7 microcontroller family.

5.286 Target ARMCM7 STM32H7

Target dependent code for the ARMCM7 STM32H7 microcontroller family.

Files

- file [ARMCM7_STM32H7/can.c](#)
Bootloader CAN communication interface source file.
- file [ARMCM7_STM32H7/cpu.c](#)
Bootloader cpu module source file.
- file [ARMCM7_STM32H7/flash.c](#)
Bootloader flash driver source file.
- file [ARMCM7_STM32H7/flash.h](#)
Bootloader flash driver header file.
- file [ARMCM7_STM32H7/GCC/cpu_comp.c](#)
Bootloader cpu module source file.
- file [ARMCM7_STM32H7/IAR/cpu_comp.c](#)
Bootloader cpu module source file.
- file [ARMCM7_STM32H7/Keil/cpu_comp.c](#)
Bootloader cpu module source file.
- file [ARMCM7_STM32H7/nvm.c](#)
Bootloader non-volatile memory driver source file.
- file [ARMCM7_STM32H7/rs232.c](#)
Bootloader RS232 communication interface source file.
- file [Source/ARMCM7_STM32H7/timer.c](#)
Bootloader timer driver source file.
- file [ARMCM7_STM32H7/types.h](#)
Bootloader types header file.
- file [ARMCM7_STM32H7/usb.c](#)
Bootloader USB communication interface source file.

5.286.1 Detailed Description

Target dependent code for the ARMCM7 STM32H7 microcontroller family.

This module implements the bootloader's target dependent part for the ARMCM7 STM32H7 microcontroller family.

5.287 Bootloader Core

Target independent code.

Files

- file [asserts.c](#)
Bootloader assertion module source file.
- file [asserts.h](#)
Bootloader assertion module header file.
- file [backdoor.c](#)
Bootloader backdoor entry source file.
- file [backdoor.h](#)
Bootloader backdoor entry header file.
- file [Source/boot.c](#)
Bootloader core module source file.
- file [Source/boot.h](#)
Bootloader core module header file.
- file [can.h](#)
Bootloader CAN communication interface header file.
- file [com.c](#)
Bootloader communication interface source file.
- file [com.h](#)
Bootloader communication interface header file.
- file [cop.c](#)
Bootloader watchdog module source file.
- file [cop.h](#)
Bootloader watchdog module header file.
- file [cpu.h](#)
Bootloader cpu module header file.
- file [file.c](#)
Bootloader file system interface source file.
- file [file.h](#)
Bootloader file system interface header file.
- file [Source/net.c](#)
Bootloader TCP/IP network communication interface source file.
- file [Source/net.h](#)
Bootloader TCP/IP network communication interface header file.
- file [nvm.h](#)
Bootloader non-volatile memory driver header file.
- file [plausibility.h](#)
Bootloader plausibility check header file, for checking the configuration at compile time.
- file [rs232.h](#)
Bootloader RS232 communication interface header file.
- file [Source/timer.h](#)
Bootloader timer driver header file.
- file [usb.h](#)
Bootloader USB communication interface header file.
- file [xcp.c](#)
XCP 1.0 protocol core source file.
- file [xcp.h](#)
XCP 1.0 protocol core header file.

5.287.1 Detailed Description

Target independent code.

The bootloader core contains the main functionality of the bootloader, independent of the microcontroller/compiler and independent of your specific application. There is generally no need for you to make changes here, unless you are adding new functionality such as the support for a new communication interface or a different communication protocol.

By default the XCP version 1.0 is used as the communication transport layer for remote firmware updates, for example via UART or CAN. Its official name is ASAM MCD-1 XCP V1.0.0 and it is a universal measurement and calibration protocol that defines a bus-independent, master-slave communication protocol to connect ECU's with calibration systems. More information can be found at <http://www.asam.net/>.

For local firmware updates, for example from a locally attached SD-card, the FATFS file system is used as a target independent interface to access files. More information can be found at http://elm-chan.org/fsw/ff/00index_e.html

5.288 Target HCS12

Target dependent code for the Freescale HCS12 microcontroller family.

Files

- file [HCS12/can.c](#)
Bootloader CAN communication interface source file.
- file [HCS12/CodeWarrior/cpu_comp.c](#)
Bootloader cpu module source file.
- file [HCS12/cpu.c](#)
Bootloader cpu module source file.
- file [HCS12/flash.c](#)
Bootloader flash driver source file.
- file [HCS12/flash.h](#)
Bootloader flash driver header file.
- file [flash_ecc.c](#)
Bootloader flash driver source file for HCS12 derivatives with error correction code in flash memory, such as the HCS12Pxx. This flash memory uses a different addressing scheme than other HCS12 derivatives.
- file [HCS12/nvm.c](#)
Bootloader non-volatile memory driver source file.
- file [HCS12/rs232.c](#)
Bootloader RS232 communication interface source file.
- file [Source/HCS12/timer.c](#)
Bootloader timer driver source file.
- file [HCS12/types.h](#)
Bootloader types header file.

5.288.1 Detailed Description

Target dependent code for the Freescale HCS12 microcontroller family.

This module implements the bootloader's target dependent part for the Freescale HCS12 microcontroller family.

5.289 Bootloader Ports

Target dependent code.

Modules

- [Target Port Template](#)
Target dependent code as a template for new microcontroller ports.
- [Target ARMCM0 S32K11](#)
Target dependent code for the NXP ARMCM0 S32K11x microcontroller family.
- [Target ARMCM0 STM32F0](#)
Target dependent code for the ARMCM0 STM32F0 microcontroller family.
- [Target ARMCM0 STM32G0](#)
Target dependent code for the ARMCM0 STM32G0 microcontroller family.
- [Target ARMCM0 XMC1](#)
Target dependent code for the ARMCM0 XMC1xxx microcontroller family.
- [Target ARMCM33 STM32L5](#)
Target dependent code for the ARMCM33 STM32L5 microcontroller family.
- [Target ARMCM3 EFM32](#)
Target dependent code for the ARMCM3 EFM32 microcontroller family.
- [Target ARMCM3 LM3S](#)
Target dependent code for the ARMCM3 LM3S microcontroller family.
- [Target ARMCM3 STM32F1](#)
Target dependent code for the ARMCM3 STM32F1 microcontroller family.
- [Target ARMCM3 STM32F2](#)
Target dependent code for the ARMCM3 STM32F2xx microcontroller family.
- [Target ARMCM4 S32K14](#)
Target dependent code for the NXP ARMCM4 S32K14x microcontroller family.
- [Target ARMCM4 STM32F3](#)
Target dependent code for the ARMCM4 STM32F3 microcontroller family.
- [Target ARMCM4 STM32F4](#)
Target dependent code for the ARMCM4 STM32F4 microcontroller family.
- [Target ARMCM4 STM32L4](#)
Target dependent code for the ARMCM4 STM32L4 microcontroller family.
- [Target ARMCM4 TM4C](#)
Target dependent code for the ARMCM4 TM4C microcontroller family.
- [Target ARMCM4 XMC4](#)
Target dependent code for the ARMCM4 XMC4xxx microcontroller family.
- [Target ARMCM7 STM32F7](#)
Target dependent code for the ARMCM7 STM32F7 microcontroller family.
- [Target ARMCM7 STM32H7](#)
Target dependent code for the ARMCM7 STM32H7 microcontroller family.
- [Target HCS12](#)
Target dependent code for the Freescale HCS12 microcontroller family.

5.289.1 Detailed Description

Target dependent code.

The bootloader targets contain the microcontroller and compiler dependent parts of the bootloader. They are grouped per microcontroller family. This is the part that needs to be newly developed when porting the bootloader to a new microcontroller family.

Chapter 6

Data Structure Documentation

6.1 tCanBusTiming Struct Reference

Structure type for grouping CAN bus timing related information.

Data Fields

- unsigned char [timeQuanta](#)
- unsigned char [propSeg](#)
- unsigned char [phaseSeg1](#)
- unsigned char [phaseSeg2](#)
- unsigned char [tseg1](#)
- unsigned char [tseg2](#)
- [blt_int8u](#) [tseg1](#)
- [blt_int8u](#) [tseg2](#)
- [blt_int8u](#) [timeQuanta](#)
- [blt_int8u](#) [propSeg](#)
- [blt_int8u](#) [phaseSeg1](#)
- [blt_int8u](#) [phaseSeg2](#)

6.1.1 Detailed Description

Structure type for grouping CAN bus timing related information.

Structure type with the layout of the CAN bus timing registers.

6.1.2 Field Documentation

6.1.2.1 [phaseSeg1](#) [1/2]

[blt_int8u](#) [phaseSeg1](#)

CAN phase segment 1

6.1.2.2 phaseSeg1 [2/2]

`blt_int8u` phaseSeg1

CAN phase segment 1

Referenced by BootComCanInit(), and CanInit().

6.1.2.3 phaseSeg2 [1/2]

`blt_int8u` phaseSeg2

CAN phase segment 2

6.1.2.4 phaseSeg2 [2/2]

`blt_int8u` phaseSeg2

CAN phase segment 2

Referenced by BootComCanInit(), and CanInit().

6.1.2.5 propSeg [1/2]

`blt_int8u` propSeg

CAN propagation segment

6.1.2.6 propSeg [2/2]

`blt_int8u` propSeg

CAN propagation segment

Referenced by BootComCanInit(), and CanInit().

6.1.2.7 timeQuanta [1/2]

`blt_int8u` timeQuanta

Total number of time quanta

6.1.2.8 timeQuanta [2/2]

`blt_int8u` timeQuanta

Total number of time quanta

Referenced by `CanGetSpeedConfig()`.

6.1.2.9 tseg1 [1/2]

`blt_int8u` tseg1

CAN time segment 1

6.1.2.10 tseg1 [2/2]

`blt_int8u` tseg1

CAN time segment 1

Referenced by `CanGetSpeedConfig()`.

6.1.2.11 tseg2 [1/2]

`blt_int8u` tseg2

CAN time segment 2

6.1.2.12 tseg2 [2/2]

`blt_int8u` tseg2

CAN time segment 2

Referenced by `CanGetSpeedConfig()`.

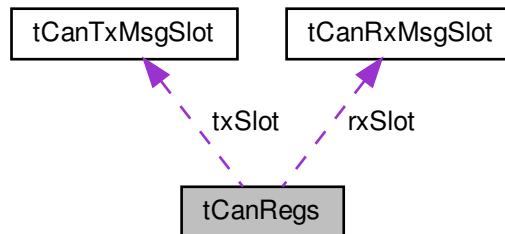
The documentation for this struct was generated from the following files:

- `Demo/ARMCM0_S32K11_S32K118EVB_GCC/Prog/boot.c`
- [_template/can.c](#)

6.2 tCanRegs Struct Reference

Structure type with the layout of the CAN related control registers.

Collaboration diagram for tCanRegs:



Data Fields

- volatile `blt_int8u cctl0`
- volatile `blt_int8u cctl1`
- volatile `blt_int8u cbtr0`
- volatile `blt_int8u cbtr1`
- volatile `blt_int8u crflg`
- volatile `blt_int8u crier`
- volatile `blt_int8u ctflg`
- volatile `blt_int8u ctier`
- volatile `blt_int8u ctarq`
- volatile `blt_int8u ctaak`
- volatile `blt_int8u ctbsel`
- volatile `blt_int8u cidac`
- volatile `blt_int8u dummy1` [2]
- volatile `blt_int8u cxerr`
- volatile `blt_int8u ctxerr`
- volatile `blt_int8u cidar0`
- volatile `blt_int8u cidar1`
- volatile `blt_int8u cidar2`
- volatile `blt_int8u cidar3`
- volatile `blt_int8u cidmr0`
- volatile `blt_int8u cidmr1`
- volatile `blt_int8u cidmr2`
- volatile `blt_int8u cidmr3`
- volatile `blt_int8u cidar4`
- volatile `blt_int8u cidar5`
- volatile `blt_int8u cidar6`
- volatile `blt_int8u cidar7`
- volatile `blt_int8u cidmr4`
- volatile `blt_int8u cidmr5`
- volatile `blt_int8u cidmr6`
- volatile `blt_int8u cidmr7`
- volatile `tCanRxMsgSlot rxSlot`
- volatile `tCanTxMsgSlot txSlot`

6.2.1 Detailed Description

Structure type with the layout of the CAN related control registers.

6.2.2 Field Documentation

6.2.2.1 cbtr0

```
volatile blt_int8u cbtr0
```

bus timing register 0

6.2.2.2 cbtr1

```
volatile blt_int8u cbtr1
```

bus timing register 1

6.2.2.3 cctl0

```
volatile blt_int8u cctl0
```

control register 0

6.2.2.4 cctl1

```
volatile blt_int8u cctl1
```

control register 1

6.2.2.5 cidac

```
volatile blt_int8u cidac
```

identifier acceptance control register

6.2.2.6 cidar0

```
volatile blt_int8u cidar0
```

identifier acceptance register 0

6.2.2.7 cidar1

volatile `blt_int8u` cidar1

identifier acceptance register 1

6.2.2.8 cidar2

volatile `blt_int8u` cidar2

identifier acceptance register 2

6.2.2.9 cidar3

volatile `blt_int8u` cidar3

identifier acceptance register 3

6.2.2.10 cidar4

volatile `blt_int8u` cidar4

identifier acceptance register 4

6.2.2.11 cidar5

volatile `blt_int8u` cidar5

identifier acceptance register 5

6.2.2.12 cidar6

volatile `blt_int8u` cidar6

identifier acceptance register 6

6.2.2.13 cidar7

volatile `blt_int8u` cidar7

identifier acceptance register 7

6.2.2.14 cidmr0

volatile `blt_int8u` cidmr0

identifier mask register 0

6.2.2.15 cidmr1

volatile `blt_int8u` cidmr1

identifier mask register 1

6.2.2.16 cidmr2

volatile `blt_int8u` cidmr2

identifier mask register 2

6.2.2.17 cidmr3

volatile `blt_int8u` cidmr3

identifier mask register 3

6.2.2.18 cidmr4

volatile `blt_int8u` cidmr4

identifier mask register 4

6.2.2.19 cidmr5

volatile `blt_int8u` cidmr5

identifier mask register 5

6.2.2.20 cidmr6

volatile `blt_int8u` cidmr6

identifier mask register 6

6.2.2.21 cidmr7

volatile `blt_int8u` cidmr7

identifier mask register 7

6.2.2.22 crflg

volatile `blt_int8u` crflg

receiver flag register

6.2.2.23 crier

volatile `blt_int8u` crier

receiver interrupt enable register

6.2.2.24 crxerr

volatile `blt_int8u` crxerr

receive error counter

6.2.2.25 ctaak

volatile `blt_int8u` ctaak

transmitter message abort control

6.2.2.26 ctarq

volatile `blt_int8u` ctarq

transmitter message abort control

6.2.2.27 ctbsel

volatile `blt_int8u` ctbsel

transmit buffer selection

6.2.2.28 ctflg

volatile `blt_int8u` ctflg

transmitter flag register

6.2.2.29 ctier

volatile `blt_int8u` ctier

transmitter interrupt enable register

6.2.2.30 ctxerr

volatile `blt_int8u` ctxerr

transmit error counter

6.2.2.31 dummy1

```
volatile blt_int8u dummy1[2]
```

reserved (2)

6.2.2.32 rxSlot

```
volatile tCanRxMsgSlot rxSlot
```

foreground receive message slot

6.2.2.33 txSlot

```
volatile tCanTxMsgSlot txSlot
```

foreground transmit message slot

The documentation for this struct was generated from the following file:

- [HCS12/can.c](#)

6.3 tCanRxMsgSlot Struct Reference

Structure type with the layout of a CAN reception message slot.

Data Fields

- volatile [blt_int8u](#) [idr](#) [4]
- volatile [blt_int8u](#) [dsr](#) [8]
- volatile [blt_int8u](#) [dlr](#)
- volatile [blt_int8u](#) [dummy](#)
- volatile [blt_int16u](#) [tstamp](#)

6.3.1 Detailed Description

Structure type with the layout of a CAN reception message slot.

6.3.2 Field Documentation

6.3.2.1 dlr

```
volatile blt_int8u dlr
```

data length register

6.3.2.2 dsr

```
volatile blt_int8u dsr[8]
```

data segment register 0..7

6.3.2.3 dummy

```
volatile blt_int8u dummy
```

unused

6.3.2.4 idr

```
volatile blt_int8u idr[4]
```

identifier register 0..3

6.3.2.5 tstamp

```
volatile blt_int16u tstamp
```

timestamp register

The documentation for this struct was generated from the following file:

- [HCS12/can.c](#)

6.4 tCanTxMsgSlot Struct Reference

Structure type with the layout of a CAN transmit message slot.

Data Fields

- volatile [blt_int8u idr](#) [4]
- volatile [blt_int8u dsr](#) [8]
- volatile [blt_int8u dlr](#)
- volatile [blt_int8u tbpr](#)
- volatile [blt_int16u tstamp](#)

6.4.1 Detailed Description

Structure type with the layout of a CAN transmit message slot.

6.4.2 Field Documentation

6.4.2.1 dlr

```
volatile blt_int8u dlr
```

data length register

6.4.2.2 dsr

```
volatile blt_int8u dsr[8]
```

data segment register 0..7

6.4.2.3 idr

```
volatile blt_int8u idr[4]
```

identifier register 0..3

6.4.2.4 tbpr

```
volatile blt_int8u tbpr
```

transmit buffer priority register

6.4.2.5 tstamp

```
volatile blt_int16u tstamp
```

timestamp register

The documentation for this struct was generated from the following file:

- [HCS12/can.c](#)

6.5 tFatFsObjects Struct Reference

Structure type for grouping FATFS related objects used by this module.

Data Fields

- FATFS [fs](#)
- FIL [file](#)

6.5.1 Detailed Description

Structure type for grouping FATFS related objects used by this module.

6.5.2 Field Documentation

6.5.2.1 file

FIL [file](#)

file object for firmware file

Referenced by FileTask().

6.5.2.2 fs

FATFS [fs](#)

file system object for mouting

Referenced by FileInit().

The documentation for this struct was generated from the following file:

- [file.c](#)

6.6 tFifoCtrl Struct Reference

Structure type for fifo control.

Data Fields

- [blt_int8u](#) * [startptr](#)
- [blt_int8u](#) * [endptr](#)
- [blt_int8u](#) * [readptr](#)
- [blt_int8u](#) * [writeptr](#)
- [blt_int8u](#) [length](#)
- [blt_int8u](#) [entries](#)
- [blt_int8u](#) [handle](#)
- struct t_fifo_ctrl * [fifoctrlptr](#)

6.6.1 Detailed Description

Structure type for fifo control.

6.6.2 Field Documentation

6.6.2.1 endptr

```
blt_int8u * endptr
```

pointer to end of buffer

Referenced by UsbFifoMgrCreate().

6.6.2.2 entries

```
blt_int8u entries
```

of full buffer elements

Referenced by UsbFifoMgrCreate(), UsbFifoMgrRead(), UsbFifoMgrScan(), and UsbFifoMgrWrite().

6.6.2.3 fifoctrlptr

```
struct t_fifo_ctrl * fifoctrlptr
```

pointer to free buffer control

Referenced by UsbFifoMgrCreate(), and UsbFifoMgrInit().

6.6.2.4 handle

```
blt_int8u handle
```

handle of the buffer

Referenced by UsbFifoMgrCreate(), and UsbFifoMgrInit().

6.6.2.5 length

`blt_int8u` length

number of buffer elements

Referenced by `UsbFifoMgrCreate()`.

6.6.2.6 readptr

`blt_int8u *` readptr

pointer to next read location

Referenced by `UsbFifoMgrCreate()`, and `UsbFifoMgrRead()`.

6.6.2.7 startptr

`blt_int8u *` startptr

pointer to start of buffer

Referenced by `UsbFifoMgrCreate()`, `UsbFifoMgrRead()`, and `UsbFifoMgrWrite()`.

6.6.2.8 writeptr

`blt_int8u *` writeptr

pointer to next free location

Referenced by `UsbFifoMgrCreate()`, and `UsbFifoMgrWrite()`.

The documentation for this struct was generated from the following file:

- [_template/usb.c](#)

6.7 tFifoPipe Struct Reference

Structure type for a fifo pipe.

Data Fields

- [blt_int8u handle](#)
- [blt_int8u data](#) [[FIFO_PIPE_SIZE](#)]

6.7.1 Detailed Description

Structure type for a fifo pipe.

6.7.2 Field Documentation

6.7.2.1 data

[blt_int8u](#) data

fifo data buffer

Referenced by [UsbInit\(\)](#).

6.7.2.2 handle

[blt_int8u](#) handle

fifo handle

Referenced by [UsbInit\(\)](#), [UsbReceiveByte\(\)](#), [UsbReceivePipeBulkOUT\(\)](#), [UsbTransmitByte\(\)](#), and [UsbTransmitPipeBulkIN\(\)](#).

The documentation for this struct was generated from the following file:

- [_template/usb.c](#)

6.8 tFileEraseInfo Struct Reference

Structure type with information for the memory erase opeartion.

Data Fields

- [blt_addr](#) start_address
- [blt_int32u](#) total_size

6.8.1 Detailed Description

Structure type with information for the memory erase operation.

6.8.2 Field Documentation

6.8.2.1 start_address

`blt_addr start_address`

erase start address

Referenced by FileTask().

6.8.2.2 total_size

`blt_int32u total_size`

total number of bytes to erase

Referenced by FileTask().

The documentation for this struct was generated from the following file:

- [file.c](#)

6.9 tFlashBlockInfo Struct Reference

Structure type for grouping flash block information.

6.9.1 Detailed Description

Structure type for grouping flash block information.

Programming is done per block of max FLASH_WRITE_BLOCK_SIZE. for this a flash block manager is implemented in this driver. this flash block manager depends on this flash block info structure. It holds the base address of the flash block and the data that should be programmed into the flash block. The .base_addr must be a multiple of FLASH_WRITE_BLOCK_SIZE.

The documentation for this struct was generated from the following files:

- [_template/flash.c](#)
- [flash_ecc.c](#)

6.10 tFlashPrescalerSysclockMapping Struct Reference

Mapping table for finding the corect flash clock divider prescaler.

Data Fields

- [blt_int16u sysclock_min](#)
- [blt_int16u sysclock_max](#)
- [blt_int8u prescaler](#)

6.10.1 Detailed Description

Mapping table for finding the corect flash clock divider prescaler.

6.10.2 Field Documentation

6.10.2.1 prescaler

[blt_int8u](#) prescaler

prescaler for this busclock range

Referenced by `FlashInit()`.

6.10.2.2 sysclock_max

[blt_int16u](#) sysclock_max

max busclock for this prescaler

Referenced by `FlashInit()`.

6.10.2.3 sysclock_min

[blt_int16u](#) sysclock_min

min busclock for this prescaler

The documentation for this struct was generated from the following file:

- [flash_ecc.c](#)

6.11 tFlashRegs Struct Reference

Structure type for the flash control registers.

Data Fields

- volatile `blt_int8u` `fcldiv`
- volatile `blt_int8u` `fsec`
- volatile `blt_int8u` `ftstmod`
- volatile `blt_int8u` `fcnfg`
- volatile `blt_int8u` `fprot`
- volatile `blt_int8u` `fstat`
- volatile `blt_int8u` `fcmd`
- volatile `blt_int8u` `fccobix`
- volatile `blt_int8u` `frsv0`
- volatile `blt_int8u` `fercnfg`
- volatile `blt_int8u` `ferstat`
- volatile `blt_int8u` `dfprot`
- volatile `blt_int16u` `fccob`
- volatile `blt_int8u` `frsv1`
- volatile `blt_int8u` `frsv2`
- volatile `blt_int8u` `frsv3`
- volatile `blt_int8u` `frsv4`
- volatile `blt_int8u` `fopt`
- volatile `blt_int8u` `frsv5`
- volatile `blt_int8u` `frsv6`
- volatile `blt_int8u` `frsv7`

6.11.1 Detailed Description

Structure type for the flash control registers.

6.11.2 Field Documentation

6.11.2.1 dfprot

```
volatile blt_int8u dfprot
```

data flash protection register

6.11.2.2 fccob

```
volatile blt_int16u fccob
```

flash command common object reg.

6.11.2.3 fccobix

volatile `blt_int8u` fccobix

flash CCOB index register

6.11.2.4 fclkdiv

volatile `blt_int8u` fclkdiv

flash clock divider register

6.11.2.5 fcmd

volatile `blt_int8u` fcmd

flash command register

6.11.2.6 fcncfg

volatile `blt_int8u` fcncfg

flash configuration register

6.11.2.7 fercnfg

volatile `blt_int8u` fercnfg

flash error configuration reg.

6.11.2.8 ferstat

volatile `blt_int8u` ferstat

flash error status register

6.11.2.9 fopt

volatile `blt_int8u` fopt

flash option register

6.11.2.10 fprot

```
volatile blt_int8u fprot
```

flash protection register

program flash protection register

6.11.2.11 frsv0

```
volatile blt_int8u frsv0
```

flash reserver register

6.11.2.12 frsv1

```
volatile blt_int8u frsv1
```

flash reserver register

6.11.2.13 frsv2

```
volatile blt_int8u frsv2
```

flash reserver register

6.11.2.14 frsv3

```
volatile blt_int8u frsv3
```

flash reserver register

6.11.2.15 frsv4

```
volatile blt_int8u frsv4
```

flash reserver register

6.11.2.16 frsv5

```
volatile blt_int8u frsv5
```

flash reserver register

6.11.2.17 frsv6

```
volatile blt_int8u frsv6
```

flash reserver register

6.11.2.18 frsv7

```
volatile blt_int8u frsv7
```

flash reserver register

6.11.2.19 fsec

```
volatile blt_int8u fsec
```

flash security register

6.11.2.20 fstat

```
volatile blt_int8u fstat
```

flash status register

6.11.2.21 ftstmod

```
volatile blt_int8u ftstmod
```

flash test mode register

The documentation for this struct was generated from the following files:

- [HCS12/flash.c](#)
- [flash_ecc.c](#)

6.12 tFlashSector Struct Reference

Flash sector descriptor type.

Data Fields

- [blt_addr sector_start](#)
- [blt_int32u sector_size](#)
- [blt_int8u sector_num](#)
- [blt_int8u bank_num](#)

6.12.1 Detailed Description

Flash sector descriptor type.

Structure type for the flash sectors in the flash layout table.

Flash sector descriptor type. Note that in this driver the word sector is synonym to the word page, which is used in the HAL driver.

6.12.2 Field Documentation

6.12.2.1 bank_num

```
blt_int8u bank_num
```

bank number

Referenced by FlashEraseSectors().

6.12.2.2 sector_num

```
blt_int8u sector_num
```

sector number

Referenced by FlashEraseSectors(), and FlashGetSector().

6.12.2.3 sector_size

```
blt_int32u sector_size
```

sector size in bytes

Referenced by FlashEraseSectors(), and FlashGetSectorSize().

6.12.2.4 sector_start

`blt_addr` sector_start

sector start address

Referenced by FlashEraseSectors(), FlashGetSectorBaseAddr(), FlashGetUserProgBaseAddress(), and FlashSwitchBlock().

The documentation for this struct was generated from the following files:

- `_template/flash.c`
- `flash_ecc.c`

6.13 tIsrFunc Union Reference

Structure type for vector table entries.

Data Fields

- `void(* func)(void)`
- `blt_int32u ptr`
- unsigned long `ptr`
- `void * ptr`

6.13.1 Detailed Description

Structure type for vector table entries.

6.13.2 Field Documentation

6.13.2.1 func

`void(* func)(void)`

for ISR function pointers

6.13.2.2 ptr [1/3]

`void* ptr`

for stack pointer entry

6.13.2.3 ptr [2/3]

```
void * ptr
```

for stack pointer entry

Referenced by `__attribute__()`.

6.13.2.4 ptr [3/3]

```
unsigned long ptr
```

for stack pointer entry

The documentation for this union was generated from the following file:

- [ARMCM3_EFM32_Olimex_EM32G880F128STK_GCC/Boot/vectors.c](#)

6.14 tSharedParamsBuffer Struct Reference

Layout of the shared parameters RAM buffer.

Data Fields

- `uint32_t identifier`
Fixed buffer identifier to validate that the RAM contains valid shared parameters.
- `uint8_t data [SHARED_PARAMS_CFG_BUFFER_DATA_LEN]`
Array for the actual parameter data.
- `uint16_t checksum`
Checksum value of all the bytes in the buffer, excluding this checksum value of obvious reasons. The checksum is calculated as the Two's complement of the sum of the bytes.

6.14.1 Detailed Description

Layout of the shared parameters RAM buffer.

The documentation for this struct was generated from the following file:

- [ARMCM3_LM3S_EK_LM3S6965_GCC/Boot/shared_params.c](#)

6.15 tSrecLineParseObject Struct Reference

Structure type for grouping the parsing results of an S-record line.

```
#include <file.h>
```

Data Fields

- [blt_char](#) line [MAX_CHARS_PER_LINE]
- [blt_int8u](#) data [MAX_DATA_BYTES_PER_LINE]
- [blt_addr](#) address

6.15.1 Detailed Description

Structure type for grouping the parsing results of an S-record line.

6.15.2 Field Documentation

6.15.2.1 address

[blt_addr](#) address

address on S1, S2 or S3 line

Referenced by `FileTask()`.

6.15.2.2 data

[blt_int8u](#) data [MAX_DATA_BYTES_PER_LINE]

array for S1, S2 or S3 data bytes

Referenced by `FileTask()`.

6.15.2.3 line

[blt_char](#) line [MAX_CHARS_PER_LINE]

string buffer for the line chars

Referenced by `FileTask()`.

The documentation for this struct was generated from the following file:

- [file.h](#)

6.16 tSysTickRegs Struct Reference

Systick registers.

Data Fields

- volatile [blt_int32u CTRL](#)
- volatile [blt_int32u LOAD](#)
- volatile [blt_int32u VAL](#)

6.16.1 Detailed Description

Systick registers.

6.16.2 Field Documentation

6.16.2.1 CTRL

volatile [blt_int32u CTRL](#)

SysTick Control and Status Register

6.16.2.2 LOAD

volatile [blt_int32u LOAD](#)

SysTick Reload Value Register

6.16.2.3 VAL

volatile [blt_int32u VAL](#)

SysTick Current Value Register

The documentation for this struct was generated from the following file:

- [Source/ARMCM0_XMC1/timer.c](#)

6.17 tTimerRegs Struct Reference

Structure type with the layout of the timer related control registers.

Data Fields

- volatile `blt_int8u` `tios`
- volatile `blt_int8u` `cforc`
- volatile `blt_int8u` `oc7m`
- volatile `blt_int8u` `oc7d`
- volatile `blt_int16u` `tcnt`
- volatile `blt_int8u` `tscr1`
- volatile `blt_int8u` `ttov`
- volatile `blt_int8u` `tctl1`
- volatile `blt_int8u` `tctl2`
- volatile `blt_int8u` `tctl3`
- volatile `blt_int8u` `tctl4`
- volatile `blt_int8u` `tie`
- volatile `blt_int8u` `tscr2`
- volatile `blt_int8u` `tflg1`
- volatile `blt_int8u` `tflg2`
- volatile `blt_int16u` `tc` [8]

6.17.1 Detailed Description

Structure type with the layout of the timer related control registers.

6.17.2 Field Documentation

6.17.2.1 `cforc`

```
volatile blt_int8u cforc
```

compare force register

6.17.2.2 `oc7d`

```
volatile blt_int8u oc7d
```

output compare 7 data register

6.17.2.3 `oc7m`

```
volatile blt_int8u oc7m
```

output compare 7 mask register

6.17.2.4 tc

```
volatile blt_int16u tc[8]
```

input capture/output compare register n

6.17.2.5 tcnt

```
volatile blt_int16u tcnt
```

timer counter register

6.17.2.6 tctl1

```
volatile blt_int8u tctl1
```

timer control register 1

6.17.2.7 tctl2

```
volatile blt_int8u tctl2
```

timer control register 2

6.17.2.8 tctl3

```
volatile blt_int8u tctl3
```

timer control register 3

6.17.2.9 tctl4

```
volatile blt_int8u tctl4
```

timer control register 4

6.17.2.10 tflg1

```
volatile blt_int8u tflg1
```

timer interrupt flag 1

6.17.2.11 tflg2

```
volatile blt_int8u tflg2
```

timer interrupt flag 2

6.17.2.12 tie

volatile [blt_int8u](#) tie

interrupt enable register

6.17.2.13 tios

volatile [blt_int8u](#) tios

input capture/output compare select

6.17.2.14 tscr1

volatile [blt_int8u](#) tscr1

system control register 1

6.17.2.15 tscr2

volatile [blt_int8u](#) tscr2

system control register 2

6.17.2.16 ttov

volatile [blt_int8u](#) ttov

toggle overflow register

The documentation for this struct was generated from the following file:

- [Source/HCS12/timer.c](#)

6.18 tXcpInfo Struct Reference

Structure type for grouping XCP internal module information.

Data Fields

- [blt_int8u](#) connected
- [blt_int8u](#) protection
- [blt_int8u](#) s_n_k_resource
- [blt_int8u](#) ctoData [BOOT_COM_RX_MAX_DATA]
- [blt_int8u](#) ctoPending
- [blt_int16s](#) ctoLen
- [blt_int32u](#) mta

6.18.1 Detailed Description

Structure type for grouping XCP internal module information.

6.18.2 Field Documentation

6.18.2.1 connected

`blt_int8u` connected

connection established

Referenced by `XcpCmdConnect()`, `XcpCmdDisconnect()`, `XcpCmdUnlock()`, `XcpInit()`, `XcplsConnected()`, and `XcpPacketReceived()`.

6.18.2.2 ctoData

`blt_int8u` ctoData[`BOOT_COM_RX_MAX_DATA`]

cto packet data buffer

Referenced by `XcpCmdBuildCheckSum()`, `XcpCmdConnect()`, `XcpCmdDisconnect()`, `XcpCmdGetId()`, `XcpCmd↵`
`GetSeed()`, `XcpCmdGetStatus()`, `XcpCmdProgram()`, `XcpCmdProgramClear()`, `XcpCmdProgramMax()`, `XcpCmd↵`
`ProgramReset()`, `XcpCmdProgramStart()`, `XcpCmdSetMta()`, `XcpCmdShortUpload()`, `XcpCmdUnlock()`, `XcpCmd↵`
`Upload()`, `XcpPacketReceived()`, and `XcpSetCtoError()`.

6.18.2.3 ctoLen

`blt_int16s` ctoLen

cto current packet length

Referenced by `XcpCmdBuildCheckSum()`, `XcpCmdConnect()`, `XcpCmdDisconnect()`, `XcpCmdGetId()`, `XcpCmd↵`
`GetSeed()`, `XcpCmdGetStatus()`, `XcpCmdProgram()`, `XcpCmdProgramClear()`, `XcpCmdProgramMax()`, `XcpCmd↵`
`ProgramReset()`, `XcpCmdProgramStart()`, `XcpCmdSetMta()`, `XcpCmdShortUpload()`, `XcpCmdUnlock()`, `XcpCmd↵`
`Upload()`, `XcpInit()`, `XcpPacketReceived()`, and `XcpSetCtoError()`.

6.18.2.4 ctoPending

`blt_int8u` ctoPending

cto transmission pending flag

Referenced by `XcpInit()`, `XcpPacketReceived()`, and `XcpPacketTransmitted()`.

6.18.2.5 mta

`blt_int32u` mta

memory transfer address

Referenced by `XcpCmdBuildChecksum()`, `XcpCmdGetId()`, `XcpCmdProgram()`, `XcpCmdProgramClear()`, `XcpCmdProgramMax()`, `XcpCmdSetMta()`, `XcpCmdShortUpload()`, `XcpCmdUpload()`, and `XcpInit()`.

6.18.2.6 protection

`blt_int8u` protection

protection state

Referenced by `XcpCmdGetSeed()`, `XcpCmdGetStatus()`, `XcpCmdProgram()`, `XcpCmdProgramClear()`, `XcpCmdProgramMax()`, `XcpCmdProgramPrepare()`, `XcpCmdProgramReset()`, `XcpCmdProgramStart()`, `XcpCmdShortUpload()`, `XcpCmdUnlock()`, `XcpCmdUpload()`, `XcpInit()`, and `XcpProtectResources()`.

6.18.2.7 s_n_k_resource

`blt_int8u` s_n_k_resource

for seed/key sequence

Referenced by `XcpCmdGetSeed()`, `XcpCmdUnlock()`, and `XcpInit()`.

The documentation for this struct was generated from the following file:

- [xcp.c](#)

6.19 uip_tcp_appstate_t Struct Reference

Define the `uip_tcp_appstate_t` datatype. This is the state of our tcp/ip application, and the memory required for this state is allocated together with each TCP connection. One application state for each TCP connection.

```
#include <net.h>
```

6.19.1 Detailed Description

Define the `uip_tcp_appstate_t` datatype. This is the state of our tcp/ip application, and the memory required for this state is allocated together with each TCP connection. One application state for each TCP connection.

The documentation for this struct was generated from the following file:

- [Demo/ARMCM3_LM3S_EK_LM3S6965_GCC/Prog/net.h](#)

Chapter 7

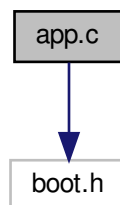
File Documentation

7.1 app.c File Reference

Bootloader application source file.

```
#include "boot.h"
```

Include dependency graph for ARMCM0_STM32F0_Discovery_STM32F051_CubeIDE/Boot/App/app.c:



Functions

- void [AppInit](#) (void)
Initializes the bootloader application. Should be called once during software program initialization.
- void [AppTask](#) (void)
Task function of the bootloader application. Should be called continuously in the program loop.

7.1.1 Detailed Description

Bootloader application source file.

7.1.2 Function Documentation

7.1.2.1 AppInit()

```
void AppInit (
    void )
```

Initializes the bootloader application. Should be called once during software program initialization.

Returns

none.

7.1.2.2 AppTask()

```
void AppTask (
    void )
```

Task function of the bootloader application. Should be called continuously in the program loop.

Returns

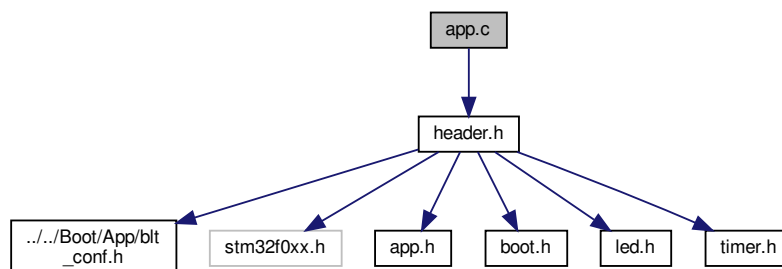
none.

7.2 app.c File Reference

User program application source file.

```
#include "header.h"
```

Include dependency graph for ARMCM0_STM32F0_Discovery_STM32F051_CubeIDE/Prog/App/app.c:



Functions

- void [AppInit](#) (void)
Initializes the user program application. Should be called once during software program initialization.
- void [AppTask](#) (void)
Task function of the user program application. Should be called continuously in the program loop.

7.2.1 Detailed Description

User program application source file.

7.2.2 Function Documentation

7.2.2.1 AppInit()

```
void AppInit (  
    void )
```

Initializes the user program application. Should be called once during software program initialization.

Initializes the bootloader application. Should be called once during software program initialization.

Returns

none.

7.2.2.2 AppTask()

```
void AppTask (  
    void )
```

Task function of the user program application. Should be called continuously in the program loop.

Task function of the bootloader application. Should be called continuously in the program loop.

Returns

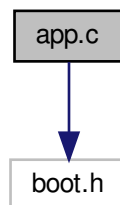
none.

7.3 app.c File Reference

Bootloader application source file.

```
#include "boot.h"
```

Include dependency graph for ARMC00_STM32F0_Nucleo_F091RC_CubeIDE/Boot/App/app.c:



Functions

- void [AppInit](#) (void)
Initializes the bootloader application. Should be called once during software program initialization.
- void [AppTask](#) (void)
Task function of the bootloader application. Should be called continuously in the program loop.

7.3.1 Detailed Description

Bootloader application source file.

7.3.2 Function Documentation

7.3.2.1 AppInit()

```
void AppInit (  
    void )
```

Initializes the bootloader application. Should be called once during software program initialization.

Returns

none.

7.3.2.2 AppTask()

```
void AppTask (  
    void )
```

Task function of the bootloader application. Should be called continuously in the program loop.

Returns

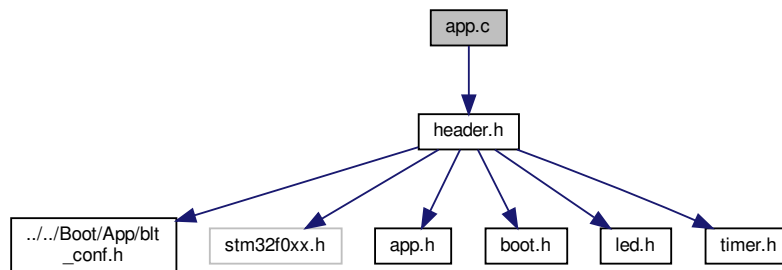
none.

7.4 app.c File Reference

User program application source file.

```
#include "header.h"
```

Include dependency graph for ARMCM0_STM32F0_Nucleo_F091RC_CubeIDE/Prog/App/app.c:



Functions

- void [AppInit](#) (void)
Initializes the user program application. Should be called once during software program initialization.
- void [AppTask](#) (void)
Task function of the user program application. Should be called continuously in the program loop.

7.4.1 Detailed Description

User program application source file.

7.4.2 Function Documentation

7.4.2.1 AppInit()

```
void AppInit (  
    void )
```

Initializes the user program application. Should be called once during software program initialization.

Initializes the bootloader application. Should be called once during software program initialization.

Returns

none.

7.4.2.2 AppTask()

```
void AppTask (  
    void )
```

Task function of the user program application. Should be called continuously in the program loop.

Task function of the bootloader application. Should be called continuously in the program loop.

Returns

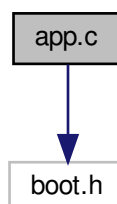
none.

7.5 app.c File Reference

Bootloader application source file.

```
#include "boot.h"
```

Include dependency graph for ARMCM0_STM32G0_Nucleo_G071RB_CubeIDE/Boot/App/app.c:



Functions

- void [Applnit](#) (void)
Initializes the bootloader application. Should be called once during software program initialization.
- void [AppTask](#) (void)
Task function of the bootloader application. Should be called continuously in the program loop.

7.5.1 Detailed Description

Bootloader application source file.

7.5.2 Function Documentation

7.5.2.1 Applnit()

```
void Applnit (
    void )
```

Initializes the bootloader application. Should be called once during software program initialization.

Returns

none.

7.5.2.2 AppTask()

```
void AppTask (
    void )
```

Task function of the bootloader application. Should be called continuously in the program loop.

Returns

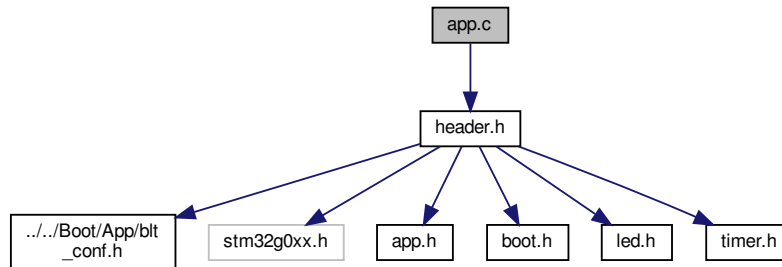
none.

7.6 app.c File Reference

User program application source file.

```
#include "header.h"
```

Include dependency graph for ARMCM0_STM32G0_Nucleo_G071RB_CubeIDE/Prog/App/app.c:



Functions

- void [AppInit](#) (void)
Initializes the user program application. Should be called once during software program initialization.
- void [AppTask](#) (void)
Task function of the user program application. Should be called continuously in the program loop.

7.6.1 Detailed Description

User program application source file.

7.6.2 Function Documentation

7.6.2.1 AppInit()

```
void AppInit (
    void )
```

Initializes the user program application. Should be called once during software program initialization.

Initializes the bootloader application. Should be called once during software program initialization.

Returns

none.

7.6.2.2 AppTask()

```
void AppTask (
    void )
```

Task function of the user program application. Should be called continuously in the program loop.

Task function of the bootloader application. Should be called continuously in the program loop.

Returns

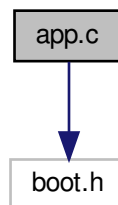
none.

7.7 app.c File Reference

Bootloader application source file.

```
#include "boot.h"
```

Include dependency graph for ARMCM33_STM32L5_Nucleo_L552ZE_CubeIDE/Boot/App/app.c:



Functions

- void [AppInit](#) (void)
Initializes the bootloader application. Should be called once during software program initialization.
- void [AppTask](#) (void)
Task function of the bootloader application. Should be called continuously in the program loop.

7.7.1 Detailed Description

Bootloader application source file.

7.7.2 Function Documentation

7.7.2.1 AppInit()

```
void AppInit (
    void )
```

Initializes the bootloader application. Should be called once during software program initialization.

Returns

none.

7.7.2.2 AppTask()

```
void AppTask (
    void )
```

Task function of the bootloader application. Should be called continuously in the program loop.

Returns

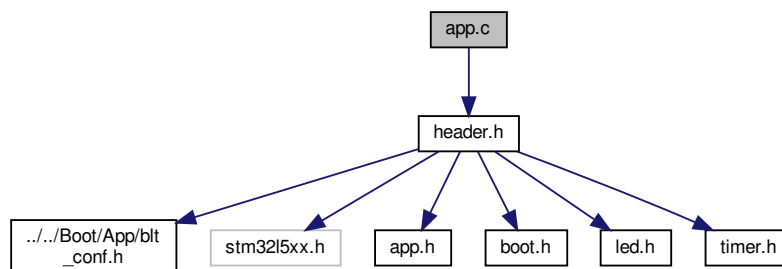
none.

7.8 app.c File Reference

User program application source file.

```
#include "header.h"
```

Include dependency graph for ARMCM33_STM32L5_Nucleo_L552ZE_CubeIDE/Prog/App/app.c:



Functions

- void [AppInit](#) (void)
Initializes the user program application. Should be called once during software program initialization.
- void [AppTask](#) (void)
Task function of the user program application. Should be called continuously in the program loop.

7.8.1 Detailed Description

User program application source file.

7.8.2 Function Documentation

7.8.2.1 AppInit()

```
void AppInit (  
    void )
```

Initializes the user program application. Should be called once during software program initialization.

Initializes the bootloader application. Should be called once during software program initialization.

Returns

none.

7.8.2.2 AppTask()

```
void AppTask (  
    void )
```

Task function of the user program application. Should be called continuously in the program loop.

Task function of the bootloader application. Should be called continuously in the program loop.

Returns

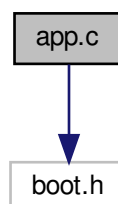
none.

7.9 app.c File Reference

Bootloader application source file.

```
#include "boot.h"
```

Include dependency graph for ARMC3M3_STM32F1_Nucleo_F103RB_CubeIDE/Boot/App/app.c:



Functions

- void [Applnit](#) (void)
Initializes the bootloader application. Should be called once during software program initialization.
- void [AppTask](#) (void)
Task function of the bootloader application. Should be called continuously in the program loop.

7.9.1 Detailed Description

Bootloader application source file.

7.9.2 Function Documentation

7.9.2.1 Applnit()

```
void Applnit (
    void )
```

Initializes the bootloader application. Should be called once during software program initialization.

Returns

none.

7.9.2.2 AppTask()

```
void AppTask (
    void )
```

Task function of the bootloader application. Should be called continuously in the program loop.

Returns

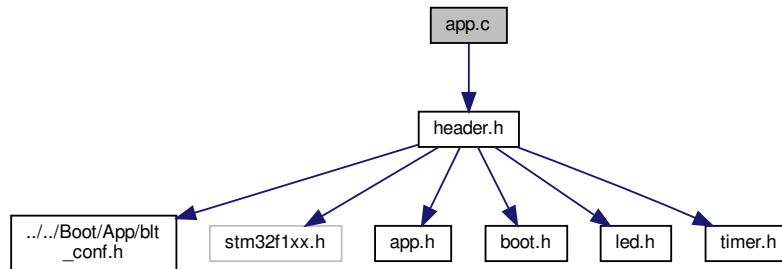
none.

7.10 app.c File Reference

User program application source file.

```
#include "header.h"
```

Include dependency graph for ARMCM3_STM32F1_Nucleo_F103RB_CubeIDE/Prog/App/app.c:



Functions

- void [AppInit](#) (void)
Initializes the user program application. Should be called once during software program initialization.
- void [AppTask](#) (void)
Task function of the user program application. Should be called continuously in the program loop.

7.10.1 Detailed Description

User program application source file.

7.10.2 Function Documentation

7.10.2.1 AppInit()

```
void AppInit (
    void )
```

Initializes the user program application. Should be called once during software program initialization.

Initializes the bootloader application. Should be called once during software program initialization.

Returns

none.

7.10.2.2 AppTask()

```
void AppTask (  
    void )
```

Task function of the user program application. Should be called continuously in the program loop.

Task function of the bootloader application. Should be called continuously in the program loop.

Returns

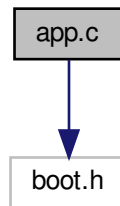
none.

7.11 app.c File Reference

Bootloader application source file.

```
#include "boot.h"
```

Include dependency graph for ARMCM3_STM32F1_Olimex_STM32H103_CubeIDE/Boot/App/app.c:



Functions

- void [AppInit](#) (void)
Initializes the bootloader application. Should be called once during software program initialization.
- void [AppTask](#) (void)
Task function of the bootloader application. Should be called continuously in the program loop.

7.11.1 Detailed Description

Bootloader application source file.

7.11.2 Function Documentation

7.11.2.1 AppInit()

```
void AppInit (
    void )
```

Initializes the bootloader application. Should be called once during software program initialization.

Returns

none.

7.11.2.2 AppTask()

```
void AppTask (
    void )
```

Task function of the bootloader application. Should be called continuously in the program loop.

Returns

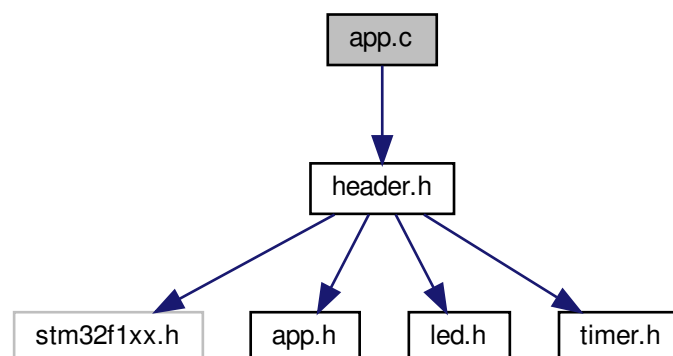
none.

7.12 app.c File Reference

User program application source file.

```
#include "header.h"
```

Include dependency graph for ARMCM3_STM32F1_Olimex_STM32H103_CubeIDE/Prog/App/app.c:



Functions

- void [AppInit](#) (void)
Initializes the user program application. Should be called once during software program initialization.
- void [AppTask](#) (void)
Task function of the user program application. Should be called continuously in the program loop.

7.12.1 Detailed Description

User program application source file.

7.12.2 Function Documentation

7.12.2.1 AppInit()

```
void AppInit (  
    void )
```

Initializes the user program application. Should be called once during software program initialization.

Initializes the bootloader application. Should be called once during software program initialization.

Returns

none.

7.12.2.2 AppTask()

```
void AppTask (  
    void )
```

Task function of the user program application. Should be called continuously in the program loop.

Task function of the bootloader application. Should be called continuously in the program loop.

Returns

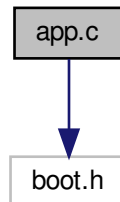
none.

7.13 app.c File Reference

Bootloader application source file.

```
#include "boot.h"
```

Include dependency graph for ARMC32F1_Olimex_STM32P103_CubeIDE/Boot/App/app.c:



Functions

- void `AppInit` (void)
Initializes the bootloader application. Should be called once during software program initialization.
- void `AppTask` (void)
Task function of the bootloader application. Should be called continuously in the program loop.

7.13.1 Detailed Description

Bootloader application source file.

7.13.2 Function Documentation

7.13.2.1 AppInit()

```
void AppInit (  
    void )
```

Initializes the bootloader application. Should be called once during software program initialization.

Returns

none.

7.13.2.2 AppTask()

```
void AppTask (
    void )
```

Task function of the bootloader application. Should be called continuously in the program loop.

Returns

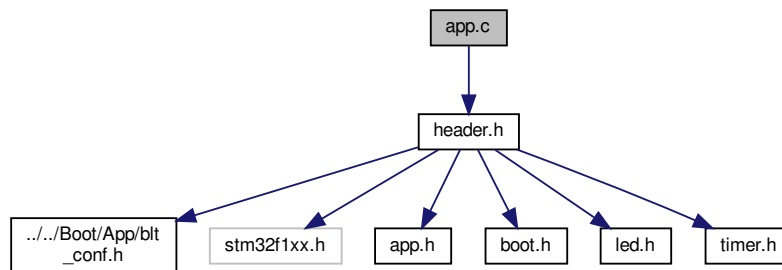
none.

7.14 app.c File Reference

User program application source file.

```
#include "header.h"
```

Include dependency graph for ARMCM3_STM32F1_Olimex_STM32P103_CubeIDE/Prog/App/app.c:



Functions

- void [AppInit](#) (void)
Initializes the user program application. Should be called once during software program initialization.
- void [AppTask](#) (void)
Task function of the user program application. Should be called continuously in the program loop.

7.14.1 Detailed Description

User program application source file.

7.14.2 Function Documentation

7.14.2.1 AppInit()

```
void AppInit (  
    void )
```

Initializes the user program application. Should be called once during software program initialization.

Initializes the bootloader application. Should be called once during software program initialization.

Returns

none.

7.14.2.2 AppTask()

```
void AppTask (  
    void )
```

Task function of the user program application. Should be called continuously in the program loop.

Task function of the bootloader application. Should be called continuously in the program loop.

Returns

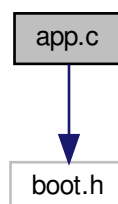
none.

7.15 app.c File Reference

Bootloader application source file.

```
#include "boot.h"
```

Include dependency graph for ARMCM3_STM32F1_Olimexino_STM32_CubeIDE/Boot/App/app.c:



Functions

- void [AppInit](#) (void)
Initializes the bootloader application. Should be called once during software program initialization.
- void [AppTask](#) (void)
Task function of the bootloader application. Should be called continuously in the program loop.

7.15.1 Detailed Description

Bootloader application source file.

7.15.2 Function Documentation

7.15.2.1 AppInit()

```
void AppInit (  
    void )
```

Initializes the bootloader application. Should be called once during software program initialization.

Returns

none.

7.15.2.2 AppTask()

```
void AppTask (  
    void )
```

Task function of the bootloader application. Should be called continuously in the program loop.

Returns

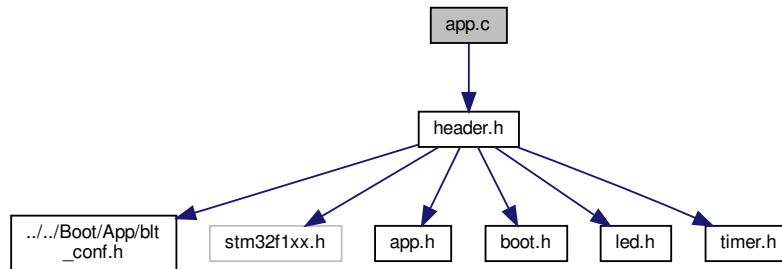
none.

7.16 app.c File Reference

User program application source file.

```
#include "header.h"
```

Include dependency graph for ARMCM3_STM32F1_Olimexino_STM32_CubeIDE/Prog/App/app.c:



Functions

- void [AppInit](#) (void)
Initializes the user program application. Should be called once during software program initialization.
- void [AppTask](#) (void)
Task function of the user program application. Should be called continuously in the program loop.

7.16.1 Detailed Description

User program application source file.

7.16.2 Function Documentation

7.16.2.1 AppInit()

```
void AppInit (
    void )
```

Initializes the user program application. Should be called once during software program initialization.

Initializes the bootloader application. Should be called once during software program initialization.

Returns

none.

7.16.2.2 AppTask()

```
void AppTask (  
    void )
```

Task function of the user program application. Should be called continuously in the program loop.

Task function of the bootloader application. Should be called continuously in the program loop.

Returns

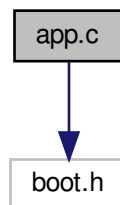
none.

7.17 app.c File Reference

Bootloader application source file.

```
#include "boot.h"
```

Include dependency graph for ARMCM3_STM32F2_Olimex_STM32P207_CubeIDE/Boot/App/app.c:



Functions

- void [AppInit](#) (void)
Initializes the bootloader application. Should be called once during software program initialization.
- void [AppTask](#) (void)
Task function of the bootloader application. Should be called continuously in the program loop.

7.17.1 Detailed Description

Bootloader application source file.

7.17.2 Function Documentation

7.17.2.1 AppInit()

```
void AppInit (  
    void )
```

Initializes the bootloader application. Should be called once during software program initialization.

Returns

none.

7.17.2.2 AppTask()

```
void AppTask (  
    void )
```

Task function of the bootloader application. Should be called continuously in the program loop.

Returns

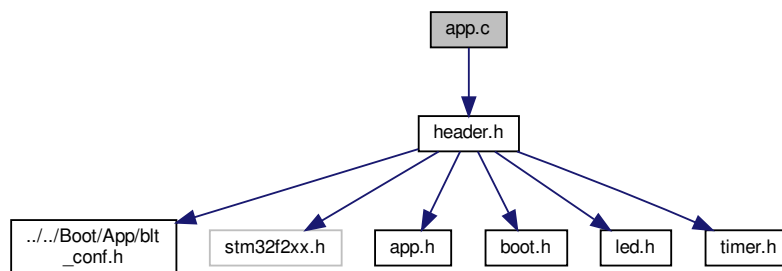
none.

7.18 app.c File Reference

User program application source file.

```
#include "header.h"
```

Include dependency graph for ARMCM3_STM32F2_Olimex_STM32P207_CubeIDE/Prog/App/app.c:



Functions

- void [AppInit](#) (void)
Initializes the user program application. Should be called once during software program initialization.
- void [AppTask](#) (void)
Task function of the user program application. Should be called continuously in the program loop.

7.18.1 Detailed Description

User program application source file.

7.18.2 Function Documentation

7.18.2.1 AppInit()

```
void AppInit (  
    void )
```

Initializes the user program application. Should be called once during software program initialization.

Initializes the bootloader application. Should be called once during software program initialization.

Returns

none.

7.18.2.2 AppTask()

```
void AppTask (  
    void )
```

Task function of the user program application. Should be called continuously in the program loop.

Task function of the bootloader application. Should be called continuously in the program loop.

Returns

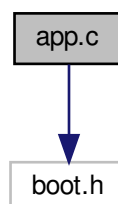
none.

7.19 app.c File Reference

Bootloader application source file.

```
#include "boot.h"
```

Include dependency graph for ARMCM4_STM32F3_Discovery_F303VC_CubeIDE/Boot/App/app.c:



Functions

- void [AppInit](#) (void)
Initializes the bootloader application. Should be called once during software program initialization.
- void [AppTask](#) (void)
Task function of the bootloader application. Should be called continuously in the program loop.

7.19.1 Detailed Description

Bootloader application source file.

7.19.2 Function Documentation

7.19.2.1 AppInit()

```
void AppInit (  
    void )
```

Initializes the bootloader application. Should be called once during software program initialization.

Returns

none.

7.19.2.2 AppTask()

```
void AppTask (  
    void )
```

Task function of the bootloader application. Should be called continuously in the program loop.

Returns

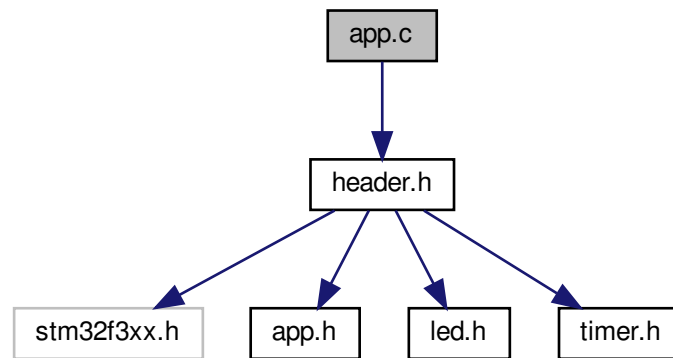
none.

7.20 app.c File Reference

User program application source file.

```
#include "header.h"
```

Include dependency graph for ARMCM4_STM32F3_Discovery_F303VC_CubeIDE/Prog/App/app.c:



Functions

- void [Applnit](#) (void)
Initializes the user program application. Should be called once during software program initialization.
- void [AppTask](#) (void)
Task function of the user program application. Should be called continuously in the program loop.

7.20.1 Detailed Description

User program application source file.

7.20.2 Function Documentation

7.20.2.1 Applnit()

```
void Applnit (
    void )
```

Initializes the user program application. Should be called once during software program initialization.

Initializes the bootloader application. Should be called once during software program initialization.

Returns

none.

7.20.2.2 AppTask()

```
void AppTask (
    void )
```

Task function of the user program application. Should be called continuously in the program loop.

Task function of the bootloader application. Should be called continuously in the program loop.

Returns

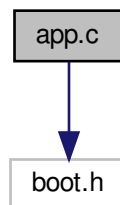
none.

7.21 app.c File Reference

Bootloader application source file.

```
#include "boot.h"
```

Include dependency graph for ARMCM4_STM32F3_Nucleo_F303K8_CubeIDE/Boot/App/app.c:



Functions

- void [AppInit](#) (void)
Initializes the bootloader application. Should be called once during software program initialization.
- void [AppTask](#) (void)
Task function of the bootloader application. Should be called continuously in the program loop.

7.21.1 Detailed Description

Bootloader application source file.

7.21.2 Function Documentation

7.21.2.1 AppInit()

```
void AppInit (
    void )
```

Initializes the bootloader application. Should be called once during software program initialization.

Returns

none.

7.21.2.2 AppTask()

```
void AppTask (
    void )
```

Task function of the bootloader application. Should be called continuously in the program loop.

Returns

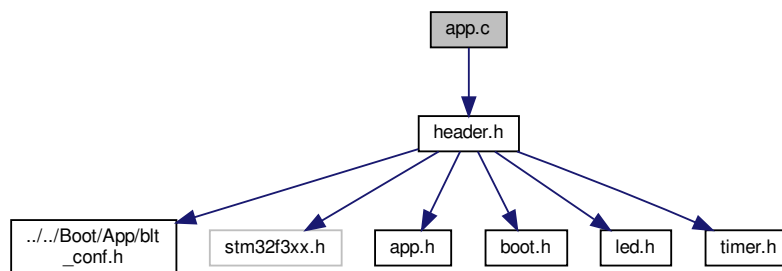
none.

7.22 app.c File Reference

User program application source file.

```
#include "header.h"
```

Include dependency graph for ARMCM4_STM32F3_Nucleo_F303K8_CubeIDE/Prog/App/app.c:



Functions

- void [AppInit](#) (void)
Initializes the user program application. Should be called once during software program initialization.
- void [AppTask](#) (void)
Task function of the user program application. Should be called continuously in the program loop.

7.22.1 Detailed Description

User program application source file.

7.22.2 Function Documentation

7.22.2.1 Applnit()

```
void AppInit (  
    void )
```

Initializes the user program application. Should be called once during software program initialization.

Initializes the bootloader application. Should be called once during software program initialization.

Returns

none.

7.22.2.2 AppTask()

```
void AppTask (  
    void )
```

Task function of the user program application. Should be called continuously in the program loop.

Task function of the bootloader application. Should be called continuously in the program loop.

Returns

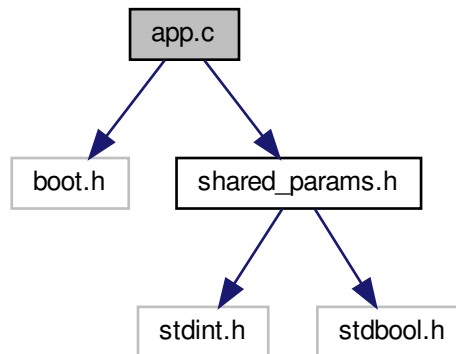
none.

7.23 app.c File Reference

Bootloader application source file.

```
#include "boot.h"
#include "shared_params.h"
```

Include dependency graph for ARMCM4_STM32F4_Nucleo_F429ZI_CubeIDE/Boot/App/app.c:



Functions

- void [AppInit](#) (void)
Initializes the bootloader application. Should be called once during software program initialization.
- void [AppTask](#) (void)
Task function of the bootloader application. Should be called continuously in the program loop.

7.23.1 Detailed Description

Bootloader application source file.

7.23.2 Function Documentation

7.23.2.1 AppInit()

```
void AppInit (
    void )
```

Initializes the bootloader application. Should be called once during software program initialization.

Returns

none.

7.23.2.2 AppTask()

```
void AppTask (
    void )
```

Task function of the bootloader application. Should be called continuously in the program loop.

Returns

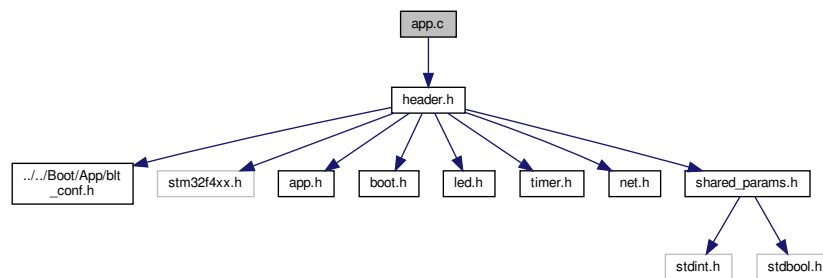
none.

7.24 app.c File Reference

User program application source file.

```
#include "header.h"
```

Include dependency graph for ARMC4_STM32F4_Nucleo_F429ZI_CubeIDE/Prog/App/app.c:



Functions

- void `AppInit` (void)
Initializes the user program application. Should be called once during software program initialization.
- void `AppTask` (void)
Task function of the user program application. Should be called continuously in the program loop.

7.24.1 Detailed Description

User program application source file.

7.24.2 Function Documentation

7.24.2.1 AppInit()

```
void AppInit (  
    void )
```

Initializes the user program application. Should be called once during software program initialization.

Initializes the bootloader application. Should be called once during software program initialization.

Returns

none.

7.24.2.2 AppTask()

```
void AppTask (  
    void )
```

Task function of the user program application. Should be called continuously in the program loop.

Task function of the bootloader application. Should be called continuously in the program loop.

Returns

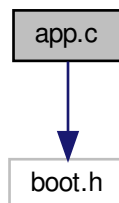
none.

7.25 app.c File Reference

Bootloader application source file.

```
#include "boot.h"
```

Include dependency graph for ARMCM4_STM32F4_Olimex_STM32P405_CubeIDE/Boot/App/app.c:



Functions

- void [AppInit](#) (void)
Initializes the bootloader application. Should be called once during software program initialization.
- void [AppTask](#) (void)
Task function of the bootloader application. Should be called continuously in the program loop.

7.25.1 Detailed Description

Bootloader application source file.

7.25.2 Function Documentation

7.25.2.1 AppInit()

```
void AppInit (  
    void )
```

Initializes the bootloader application. Should be called once during software program initialization.

Returns

none.

7.25.2.2 AppTask()

```
void AppTask (  
    void )
```

Task function of the bootloader application. Should be called continuously in the program loop.

Returns

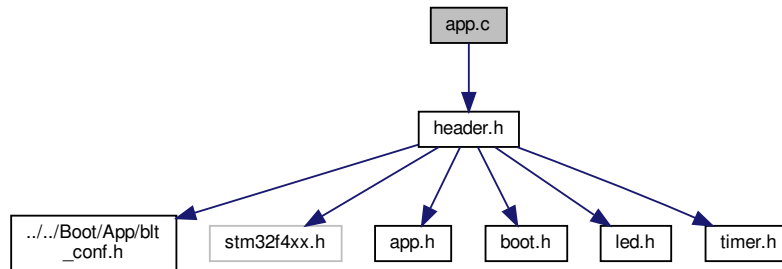
none.

7.26 app.c File Reference

User program application source file.

```
#include "header.h"
```

Include dependency graph for ARMCM4_STM32F4_Olimex_STM32P405_CubeIDE/Prog/App/app.c:



Functions

- void [AppInit](#) (void)
Initializes the user program application. Should be called once during software program initialization.
- void [AppTask](#) (void)
Task function of the user program application. Should be called continuously in the program loop.

7.26.1 Detailed Description

User program application source file.

7.26.2 Function Documentation

7.26.2.1 AppInit()

```
void AppInit (
    void )
```

Initializes the user program application. Should be called once during software program initialization.

Initializes the bootloader application. Should be called once during software program initialization.

Returns

none.

7.26.2.2 AppTask()

```
void AppTask (
    void )
```

Task function of the user program application. Should be called continuously in the program loop.

Task function of the bootloader application. Should be called continuously in the program loop.

Returns

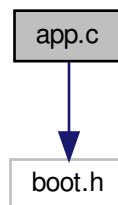
none.

7.27 app.c File Reference

Bootloader application source file.

```
#include "boot.h"
```

Include dependency graph for ARMCM4_STM32L4_Nucleo_L476RG_CubeIDE/Boot/App/app.c:



Functions

- void [AppInit](#) (void)
Initializes the bootloader application. Should be called once during software program initialization.
- void [AppTask](#) (void)
Task function of the bootloader application. Should be called continuously in the program loop.

7.27.1 Detailed Description

Bootloader application source file.

7.27.2 Function Documentation

7.27.2.1 AppInit()

```
void AppInit (
    void )
```

Initializes the bootloader application. Should be called once during software program initialization.

Returns

none.

7.27.2.2 AppTask()

```
void AppTask (
    void )
```

Task function of the bootloader application. Should be called continuously in the program loop.

Returns

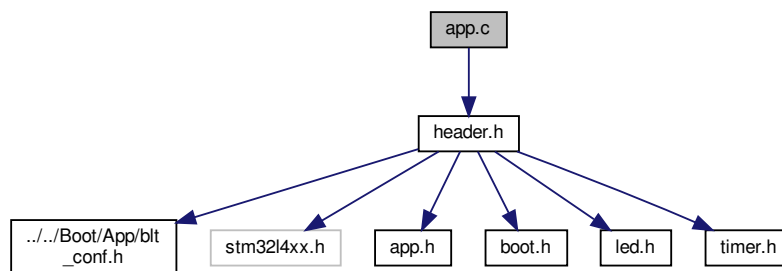
none.

7.28 app.c File Reference

User program application source file.

```
#include "header.h"
```

Include dependency graph for ARMCM4_STM32L4_Nucleo_L476RG_CubeIDE/Prog/App/app.c:



Functions

- void [AppInit](#) (void)
Initializes the user program application. Should be called once during software program initialization.
- void [AppTask](#) (void)
Task function of the user program application. Should be called continuously in the program loop.

7.28.1 Detailed Description

User program application source file.

7.28.2 Function Documentation

7.28.2.1 AppInit()

```
void AppInit (  
    void )
```

Initializes the user program application. Should be called once during software program initialization.

Initializes the bootloader application. Should be called once during software program initialization.

Returns

none.

7.28.2.2 AppTask()

```
void AppTask (  
    void )
```

Task function of the user program application. Should be called continuously in the program loop.

Task function of the bootloader application. Should be called continuously in the program loop.

Returns

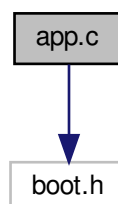
none.

7.29 app.c File Reference

Bootloader application source file.

```
#include "boot.h"
```

Include dependency graph for ARMCM7_STM32F7_Nucleo_F746ZG_CubeIDE/Boot/App/app.c:



Functions

- void [AppInit](#) (void)
Initializes the bootloader application. Should be called once during software program initialization.
- void [AppTask](#) (void)
Task function of the bootloader application. Should be called continuously in the program loop.

7.29.1 Detailed Description

Bootloader application source file.

7.29.2 Function Documentation

7.29.2.1 AppInit()

```
void AppInit (  
    void )
```

Initializes the bootloader application. Should be called once during software program initialization.

Returns

none.

7.29.2.2 AppTask()

```
void AppTask (  
    void )
```

Task function of the bootloader application. Should be called continuously in the program loop.

Returns

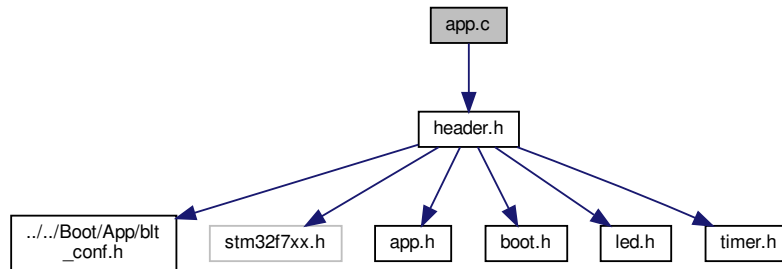
none.

7.30 app.c File Reference

User program application source file.

```
#include "header.h"
```

Include dependency graph for ARMCM7_STM32F7_Nucleo_F746ZG_CubeIDE/Prog/App/app.c:



Functions

- void `AppInit` (void)
Initializes the user program application. Should be called once during software program initialization.
- void `AppTask` (void)
Task function of the user program application. Should be called continuously in the program loop.

7.30.1 Detailed Description

User program application source file.

7.30.2 Function Documentation

7.30.2.1 AppInit()

```
void AppInit (
    void )
```

Initializes the user program application. Should be called once during software program initialization.

Initializes the bootloader application. Should be called once during software program initialization.

Returns

none.

7.30.2.2 AppTask()

```
void AppTask (  
    void )
```

Task function of the user program application. Should be called continuously in the program loop.

Task function of the bootloader application. Should be called continuously in the program loop.

Returns

none.

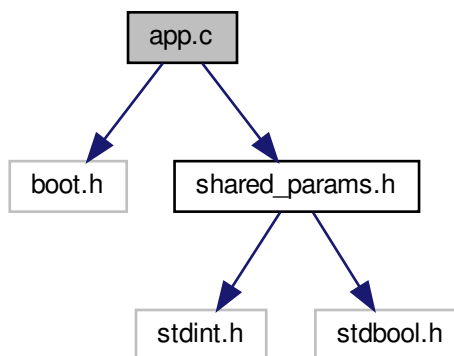
7.31 app.c File Reference

Bootloader application source file.

```
#include "boot.h"
```

```
#include "shared_params.h"
```

Include dependency graph for ARMCM7_STM32F7_Nucleo_F767ZI_CubeIDE/Boot/App/app.c:



Functions

- void [AppInit](#) (void)
Initializes the bootloader application. Should be called once during software program initialization.
- void [AppTask](#) (void)
Task function of the bootloader application. Should be called continuously in the program loop.

7.31.1 Detailed Description

Bootloader application source file.

7.31.2 Function Documentation

7.31.2.1 AppInit()

```
void AppInit (  
    void )
```

Initializes the bootloader application. Should be called once during software program initialization.

Returns

none.

7.31.2.2 AppTask()

```
void AppTask (  
    void )
```

Task function of the bootloader application. Should be called continuously in the program loop.

Returns

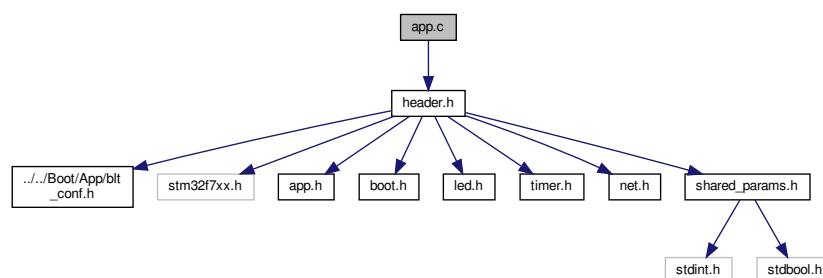
none.

7.32 app.c File Reference

User program application source file.

```
#include "header.h"
```

Include dependency graph for ARMCM7_STM32F7_Nucleo_F767ZI_CubeIDE/Prog/App/app.c:



Functions

- void [AppInit](#) (void)
Initializes the user program application. Should be called once during software program initialization.
- void [AppTask](#) (void)
Task function of the user program application. Should be called continuously in the program loop.

7.32.1 Detailed Description

User program application source file.

7.32.2 Function Documentation

7.32.2.1 AppInit()

```
void AppInit (  
    void )
```

Initializes the user program application. Should be called once during software program initialization.

Initializes the bootloader application. Should be called once during software program initialization.

Returns

none.

7.32.2.2 AppTask()

```
void AppTask (  
    void )
```

Task function of the user program application. Should be called continuously in the program loop.

Task function of the bootloader application. Should be called continuously in the program loop.

Returns

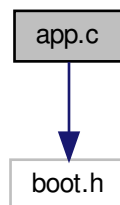
none.

7.33 app.c File Reference

Bootloader application source file.

```
#include "boot.h"
```

Include dependency graph for ARMC7_STM32H7_Nucleo_H743ZI_CubeIDE/Boot/App/app.c:



Functions

- void [AppInit](#) (void)
Initializes the bootloader application. Should be called once during software program initialization.
- void [AppTask](#) (void)
Task function of the bootloader application. Should be called continuously in the program loop.

7.33.1 Detailed Description

Bootloader application source file.

7.33.2 Function Documentation

7.33.2.1 AppInit()

```
void AppInit (  
    void )
```

Initializes the bootloader application. Should be called once during software program initialization.

Returns

none.

7.33.2.2 AppTask()

```
void AppTask (
    void )
```

Task function of the bootloader application. Should be called continuously in the program loop.

Returns

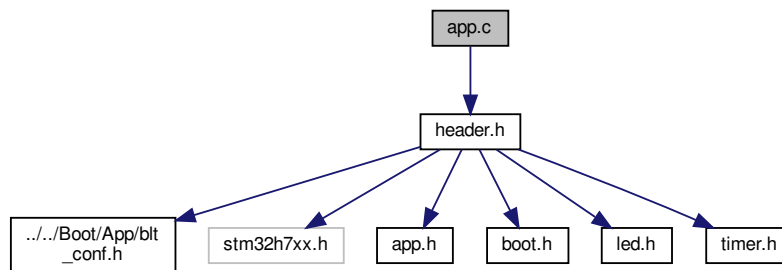
none.

7.34 app.c File Reference

User program application source file.

```
#include "header.h"
```

Include dependency graph for ARMCM7_STM32H7_Nucleo_H743ZI_CubeIDE/Prog/App/app.c:



Functions

- void [AppInit](#) (void)
Initializes the user program application. Should be called once during software program initialization.
- void [AppTask](#) (void)
Task function of the user program application. Should be called continuously in the program loop.

7.34.1 Detailed Description

User program application source file.

7.34.2 Function Documentation

7.34.2.1 AppInit()

```
void AppInit (
    void )
```

Initializes the user program application. Should be called once during software program initialization.

Initializes the bootloader application. Should be called once during software program initialization.

Returns

none.

7.34.2.2 AppTask()

```
void AppTask (
    void )
```

Task function of the user program application. Should be called continuously in the program loop.

Task function of the bootloader application. Should be called continuously in the program loop.

Returns

none.

7.35 app.h File Reference

Bootloader application header file.

Functions

- void [AppInit](#) (void)
Initializes the bootloader application. Should be called once during software program initialization.
- void [AppTask](#) (void)
Task function of the bootloader application. Should be called continuously in the program loop.

7.35.1 Detailed Description

Bootloader application header file.

7.35.2 Function Documentation

7.35.2.1 AppInit()

```
void AppInit (  
    void )
```

Initializes the bootloader application. Should be called once during software program initialization.

Returns

none.

Initializes the bootloader application. Should be called once during software program initialization.

Returns

none.

7.35.2.2 AppTask()

```
void AppTask (  
    void )
```

Task function of the bootloader application. Should be called continuously in the program loop.

Returns

none.

Task function of the bootloader application. Should be called continuously in the program loop.

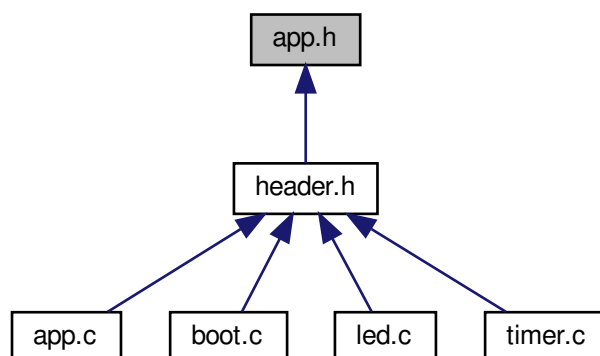
Returns

none.

7.36 app.h File Reference

User program application header file.

This graph shows which files directly or indirectly include this file:



Functions

- void [AppInit](#) (void)
Initializes the bootloader application. Should be called once during software program initialization.
- void [AppTask](#) (void)
Task function of the bootloader application. Should be called continuously in the program loop.

7.36.1 Detailed Description

User program application header file.

7.36.2 Function Documentation

7.36.2.1 AppInit()

```
void AppInit (  
    void )
```

Initializes the bootloader application. Should be called once during software program initialization.

Returns

none.

Initializes the bootloader application. Should be called once during software program initialization.

Returns

none.

7.36.2.2 AppTask()

```
void AppTask (  
    void )
```

Task function of the bootloader application. Should be called continuously in the program loop.

Returns

none.

Task function of the bootloader application. Should be called continuously in the program loop.

Returns

none.

7.37 app.h File Reference

Bootloader application header file.

Functions

- void [AppInit](#) (void)
Initializes the bootloader application. Should be called once during software program initialization.
- void [AppTask](#) (void)
Task function of the bootloader application. Should be called continuously in the program loop.

7.37.1 Detailed Description

Bootloader application header file.

7.37.2 Function Documentation

7.37.2.1 AppInit()

```
void AppInit (  
    void )
```

Initializes the bootloader application. Should be called once during software program initialization.

Returns

none.

Initializes the bootloader application. Should be called once during software program initialization.

Returns

none.

7.37.2.2 AppTask()

```
void AppTask (  
    void )
```

Task function of the bootloader application. Should be called continuously in the program loop.

Returns

none.

Task function of the bootloader application. Should be called continuously in the program loop.

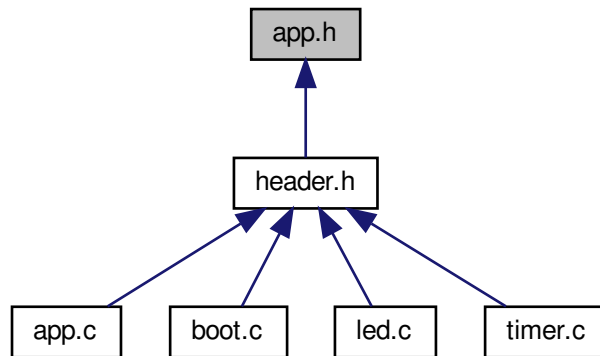
Returns

none.

7.38 app.h File Reference

User program application header file.

This graph shows which files directly or indirectly include this file:



Functions

- void [AppInit](#) (void)
Initializes the bootloader application. Should be called once during software program initialization.
- void [AppTask](#) (void)
Task function of the bootloader application. Should be called continuously in the program loop.

7.38.1 Detailed Description

User program application header file.

7.38.2 Function Documentation

7.38.2.1 AppInit()

```
void AppInit (  
    void )
```

Initializes the bootloader application. Should be called once during software program initialization.

Returns

none.

Initializes the bootloader application. Should be called once during software program initialization.

Returns

none.

7.38.2.2 AppTask()

```
void AppTask (
    void )
```

Task function of the bootloader application. Should be called continuously in the program loop.

Returns

none.

Task function of the bootloader application. Should be called continuously in the program loop.

Returns

none.

7.39 app.h File Reference

Bootloader application header file.

Functions

- void [AppInit](#) (void)
Initializes the bootloader application. Should be called once during software program initialization.
- void [AppTask](#) (void)
Task function of the bootloader application. Should be called continuously in the program loop.

7.39.1 Detailed Description

Bootloader application header file.

7.39.2 Function Documentation

7.39.2.1 AppInit()

```
void AppInit (
    void )
```

Initializes the bootloader application. Should be called once during software program initialization.

Returns

none.

Initializes the bootloader application. Should be called once during software program initialization.

Returns

none.

7.39.2.2 AppTask()

```
void AppTask (
    void )
```

Task function of the bootloader application. Should be called continuously in the program loop.

Returns

none.

Task function of the bootloader application. Should be called continuously in the program loop.

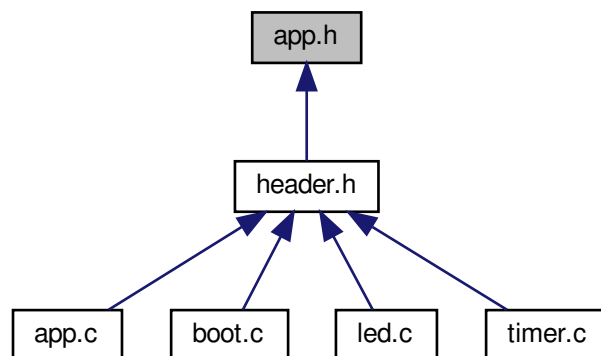
Returns

none.

7.40 app.h File Reference

User program application header file.

This graph shows which files directly or indirectly include this file:



Functions

- void [Applnit](#) (void)
Initializes the bootloader application. Should be called once during software program initialization.
- void [AppTask](#) (void)
Task function of the bootloader application. Should be called continuously in the program loop.

7.40.1 Detailed Description

User program application header file.

7.40.2 Function Documentation

7.40.2.1 AppInit()

```
void AppInit (  
    void )
```

Initializes the bootloader application. Should be called once during software program initialization.

Returns

none.

Initializes the bootloader application. Should be called once during software program initialization.

Returns

none.

7.40.2.2 AppTask()

```
void AppTask (  
    void )
```

Task function of the bootloader application. Should be called continuously in the program loop.

Returns

none.

Task function of the bootloader application. Should be called continuously in the program loop.

Returns

none.

7.41 app.h File Reference

Bootloader application header file.

Functions

- void [AppInit](#) (void)
Initializes the bootloader application. Should be called once during software program initialization.
- void [AppTask](#) (void)
Task function of the bootloader application. Should be called continuously in the program loop.

7.41.1 Detailed Description

Bootloader application header file.

7.41.2 Function Documentation

7.41.2.1 AppInit()

```
void AppInit (  
    void )
```

Initializes the bootloader application. Should be called once during software program initialization.

Returns

none.

Initializes the bootloader application. Should be called once during software program initialization.

Returns

none.

7.41.2.2 AppTask()

```
void AppTask (  
    void )
```

Task function of the bootloader application. Should be called continuously in the program loop.

Returns

none.

Task function of the bootloader application. Should be called continuously in the program loop.

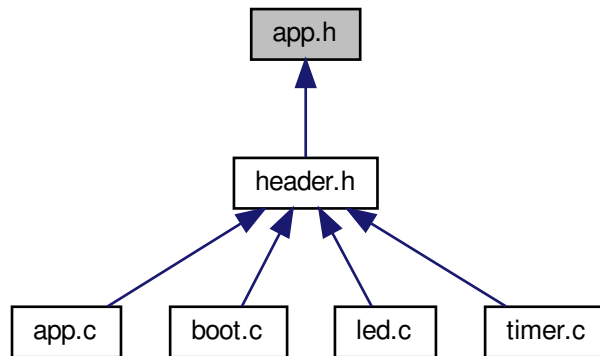
Returns

none.

7.42 app.h File Reference

User program application header file.

This graph shows which files directly or indirectly include this file:



Functions

- void [AppInit](#) (void)
Initializes the bootloader application. Should be called once during software program initialization.
- void [AppTask](#) (void)
Task function of the bootloader application. Should be called continuously in the program loop.

7.42.1 Detailed Description

User program application header file.

7.42.2 Function Documentation

7.42.2.1 AppInit()

```
void AppInit (  
    void )
```

Initializes the bootloader application. Should be called once during software program initialization.

Returns

none.

Initializes the bootloader application. Should be called once during software program initialization.

Returns

none.

7.42.2.2 AppTask()

```
void AppTask (
    void )
```

Task function of the bootloader application. Should be called continuously in the program loop.

Returns

none.

Task function of the bootloader application. Should be called continuously in the program loop.

Returns

none.

7.43 app.h File Reference

Bootloader application header file.

Functions

- void [Applnit](#) (void)
Initializes the bootloader application. Should be called once during software program initialization.
- void [AppTask](#) (void)
Task function of the bootloader application. Should be called continuously in the program loop.

7.43.1 Detailed Description

Bootloader application header file.

7.43.2 Function Documentation

7.43.2.1 Applnit()

```
void Applnit (
    void )
```

Initializes the bootloader application. Should be called once during software program initialization.

Returns

none.

Initializes the bootloader application. Should be called once during software program initialization.

Returns

none.

7.43.2.2 AppTask()

```
void AppTask (  
    void )
```

Task function of the bootloader application. Should be called continuously in the program loop.

Returns

none.

Task function of the bootloader application. Should be called continuously in the program loop.

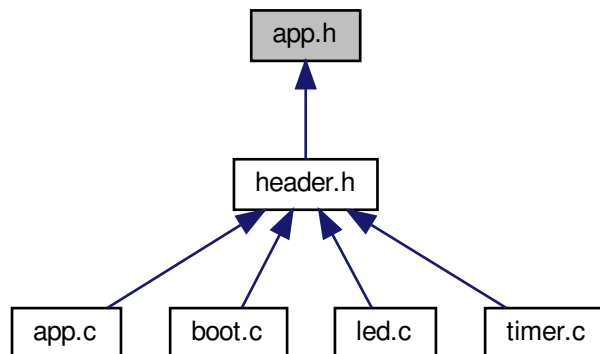
Returns

none.

7.44 app.h File Reference

User program application header file.

This graph shows which files directly or indirectly include this file:



Functions

- void `AppInit` (void)
Initializes the bootloader application. Should be called once during software program initialization.
- void `AppTask` (void)
Task function of the bootloader application. Should be called continuously in the program loop.

7.44.1 Detailed Description

User program application header file.

7.44.2 Function Documentation

7.44.2.1 AppInit()

```
void AppInit (
    void )
```

Initializes the bootloader application. Should be called once during software program initialization.

Returns

none.

Initializes the bootloader application. Should be called once during software program initialization.

Returns

none.

7.44.2.2 AppTask()

```
void AppTask (
    void )
```

Task function of the bootloader application. Should be called continuously in the program loop.

Returns

none.

Task function of the bootloader application. Should be called continuously in the program loop.

Returns

none.

7.45 app.h File Reference

Bootloader application header file.

Functions

- void [AppInit](#) (void)
Initializes the bootloader application. Should be called once during software program initialization.
- void [AppTask](#) (void)
Task function of the bootloader application. Should be called continuously in the program loop.

7.45.1 Detailed Description

Bootloader application header file.

7.45.2 Function Documentation

7.45.2.1 AppInit()

```
void AppInit (  
    void )
```

Initializes the bootloader application. Should be called once during software program initialization.

Returns

none.

Initializes the bootloader application. Should be called once during software program initialization.

Returns

none.

7.45.2.2 AppTask()

```
void AppTask (  
    void )
```

Task function of the bootloader application. Should be called continuously in the program loop.

Returns

none.

Task function of the bootloader application. Should be called continuously in the program loop.

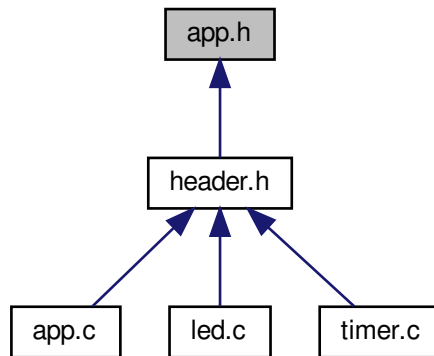
Returns

none.

7.46 app.h File Reference

User program application header file.

This graph shows which files directly or indirectly include this file:



Functions

- void [AppInit](#) (void)
Initializes the bootloader application. Should be called once during software program initialization.
- void [AppTask](#) (void)
Task function of the bootloader application. Should be called continuously in the program loop.

7.46.1 Detailed Description

User program application header file.

7.46.2 Function Documentation

7.46.2.1 AppInit()

```
void AppInit (  
    void )
```

Initializes the bootloader application. Should be called once during software program initialization.

Returns

none.

Initializes the bootloader application. Should be called once during software program initialization.

Returns

none.

7.46.2.2 AppTask()

```
void AppTask (
    void )
```

Task function of the bootloader application. Should be called continuously in the program loop.

Returns

none.

Task function of the bootloader application. Should be called continuously in the program loop.

Returns

none.

7.47 app.h File Reference

Bootloader application header file.

Functions

- void [Applnit](#) (void)
Initializes the bootloader application. Should be called once during software program initialization.
- void [AppTask](#) (void)
Task function of the bootloader application. Should be called continuously in the program loop.

7.47.1 Detailed Description

Bootloader application header file.

7.47.2 Function Documentation

7.47.2.1 Applnit()

```
void AppInit (
    void )
```

Initializes the bootloader application. Should be called once during software program initialization.

Returns

none.

Initializes the bootloader application. Should be called once during software program initialization.

Returns

none.

7.47.2.2 AppTask()

```
void AppTask (
    void )
```

Task function of the bootloader application. Should be called continuously in the program loop.

Returns

none.

Task function of the bootloader application. Should be called continuously in the program loop.

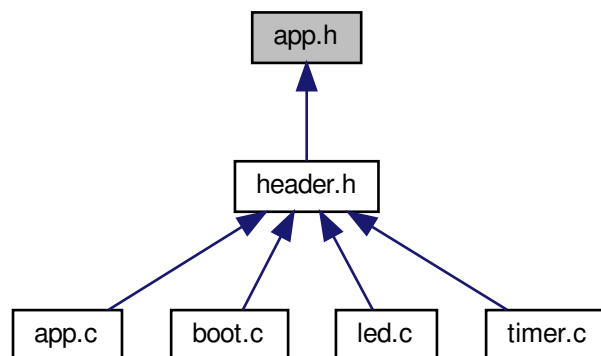
Returns

none.

7.48 app.h File Reference

User program application header file.

This graph shows which files directly or indirectly include this file:



Functions

- void [AppInit](#) (void)
Initializes the bootloader application. Should be called once during software program initialization.
- void [AppTask](#) (void)
Task function of the bootloader application. Should be called continuously in the program loop.

7.48.1 Detailed Description

User program application header file.

7.48.2 Function Documentation

7.48.2.1 AppInit()

```
void AppInit (
    void )
```

Initializes the bootloader application. Should be called once during software program initialization.

Returns

none.

Initializes the bootloader application. Should be called once during software program initialization.

Returns

none.

7.48.2.2 AppTask()

```
void AppTask (
    void )
```

Task function of the bootloader application. Should be called continuously in the program loop.

Returns

none.

Task function of the bootloader application. Should be called continuously in the program loop.

Returns

none.

7.49 app.h File Reference

Bootloader application header file.

Functions

- void [AppInit](#) (void)
Initializes the bootloader application. Should be called once during software program initialization.
- void [AppTask](#) (void)
Task function of the bootloader application. Should be called continuously in the program loop.

7.49.1 Detailed Description

Bootloader application header file.

7.49.2 Function Documentation

7.49.2.1 AppInit()

```
void AppInit (  
    void )
```

Initializes the bootloader application. Should be called once during software program initialization.

Returns

none.

Initializes the bootloader application. Should be called once during software program initialization.

Returns

none.

7.49.2.2 AppTask()

```
void AppTask (  
    void )
```

Task function of the bootloader application. Should be called continuously in the program loop.

Returns

none.

Task function of the bootloader application. Should be called continuously in the program loop.

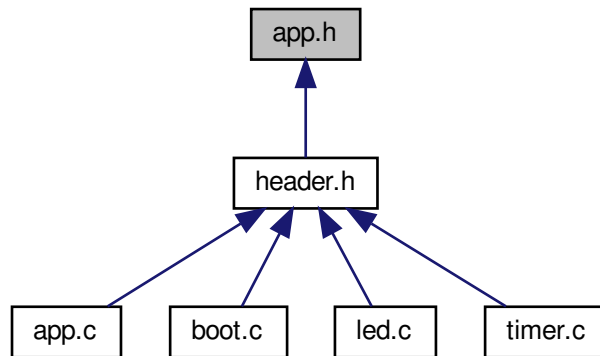
Returns

none.

7.50 app.h File Reference

User program application header file.

This graph shows which files directly or indirectly include this file:



Functions

- void [AppInit](#) (void)
Initializes the bootloader application. Should be called once during software program initialization.
- void [AppTask](#) (void)
Task function of the bootloader application. Should be called continuously in the program loop.

7.50.1 Detailed Description

User program application header file.

7.50.2 Function Documentation

7.50.2.1 AppInit()

```
void AppInit (  
    void )
```

Initializes the bootloader application. Should be called once during software program initialization.

Returns

none.

Initializes the bootloader application. Should be called once during software program initialization.

Returns

none.

7.50.2.2 AppTask()

```
void AppTask (
    void )
```

Task function of the bootloader application. Should be called continuously in the program loop.

Returns

none.

Task function of the bootloader application. Should be called continuously in the program loop.

Returns

none.

7.51 app.h File Reference

Bootloader application header file.

Functions

- void [Applnit](#) (void)
Initializes the bootloader application. Should be called once during software program initialization.
- void [AppTask](#) (void)
Task function of the bootloader application. Should be called continuously in the program loop.

7.51.1 Detailed Description

Bootloader application header file.

7.51.2 Function Documentation

7.51.2.1 Applnit()

```
void Applnit (
    void )
```

Initializes the bootloader application. Should be called once during software program initialization.

Returns

none.

Initializes the bootloader application. Should be called once during software program initialization.

Returns

none.

7.51.2.2 AppTask()

```
void AppTask (  
    void )
```

Task function of the bootloader application. Should be called continuously in the program loop.

Returns

none.

Task function of the bootloader application. Should be called continuously in the program loop.

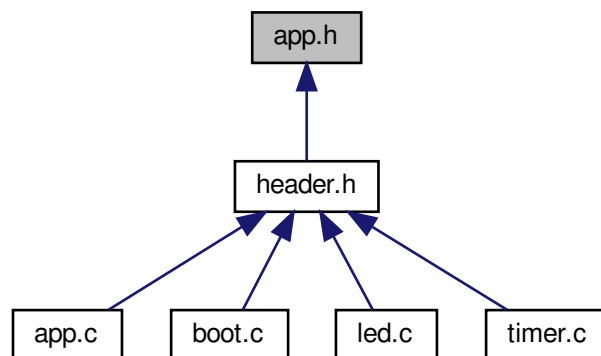
Returns

none.

7.52 app.h File Reference

User program application header file.

This graph shows which files directly or indirectly include this file:



Functions

- void [AppInit](#) (void)
Initializes the bootloader application. Should be called once during software program initialization.
- void [AppTask](#) (void)
Task function of the bootloader application. Should be called continuously in the program loop.

7.52.1 Detailed Description

User program application header file.

7.52.2 Function Documentation

7.52.2.1 Applnit()

```
void AppInit (  
    void )
```

Initializes the bootloader application. Should be called once during software program initialization.

Returns

none.

Initializes the bootloader application. Should be called once during software program initialization.

Returns

none.

7.52.2.2 AppTask()

```
void AppTask (  
    void )
```

Task function of the bootloader application. Should be called continuously in the program loop.

Returns

none.

Task function of the bootloader application. Should be called continuously in the program loop.

Returns

none.

7.53 app.h File Reference

Bootloader application header file.

Functions

- void [AppInit](#) (void)
Initializes the bootloader application. Should be called once during software program initialization.
- void [AppTask](#) (void)
Task function of the bootloader application. Should be called continuously in the program loop.

7.53.1 Detailed Description

Bootloader application header file.

7.53.2 Function Documentation

7.53.2.1 AppInit()

```
void AppInit (  
    void )
```

Initializes the bootloader application. Should be called once during software program initialization.

Returns

none.

Initializes the bootloader application. Should be called once during software program initialization.

Returns

none.

7.53.2.2 AppTask()

```
void AppTask (  
    void )
```

Task function of the bootloader application. Should be called continuously in the program loop.

Returns

none.

Task function of the bootloader application. Should be called continuously in the program loop.

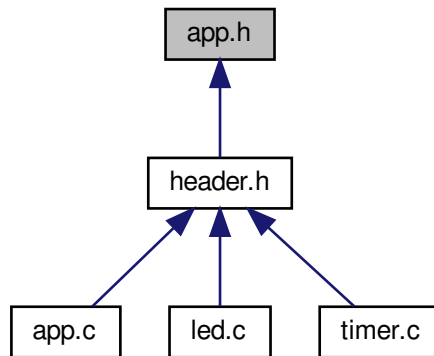
Returns

none.

7.54 app.h File Reference

User program application header file.

This graph shows which files directly or indirectly include this file:



Functions

- void [AppInit](#) (void)
Initializes the bootloader application. Should be called once during software program initialization.
- void [AppTask](#) (void)
Task function of the bootloader application. Should be called continuously in the program loop.

7.54.1 Detailed Description

User program application header file.

7.54.2 Function Documentation

7.54.2.1 AppInit()

```
void AppInit (  
    void )
```

Initializes the bootloader application. Should be called once during software program initialization.

Returns

none.

Initializes the bootloader application. Should be called once during software program initialization.

Returns

none.

7.54.2.2 AppTask()

```
void AppTask (
    void )
```

Task function of the bootloader application. Should be called continuously in the program loop.

Returns

none.

Task function of the bootloader application. Should be called continuously in the program loop.

Returns

none.

7.55 app.h File Reference

Bootloader application header file.

Functions

- void [AppInit](#) (void)
Initializes the bootloader application. Should be called once during software program initialization.
- void [AppTask](#) (void)
Task function of the bootloader application. Should be called continuously in the program loop.

7.55.1 Detailed Description

Bootloader application header file.

7.55.2 Function Documentation

7.55.2.1 AppInit()

```
void AppInit (
    void )
```

Initializes the bootloader application. Should be called once during software program initialization.

Returns

none.

Initializes the bootloader application. Should be called once during software program initialization.

Returns

none.

7.55.2.2 AppTask()

```
void AppTask (
    void )
```

Task function of the bootloader application. Should be called continuously in the program loop.

Returns

none.

Task function of the bootloader application. Should be called continuously in the program loop.

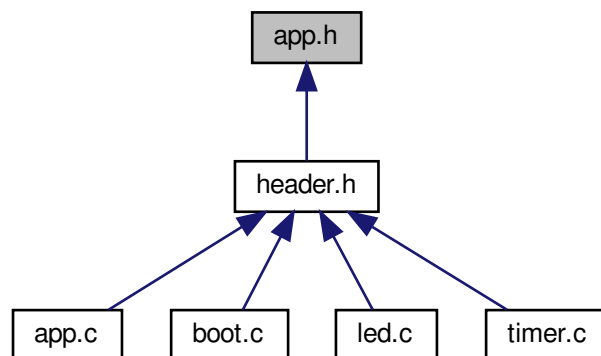
Returns

none.

7.56 app.h File Reference

User program application header file.

This graph shows which files directly or indirectly include this file:



Functions

- void [AppInit](#) (void)
Initializes the bootloader application. Should be called once during software program initialization.
- void [AppTask](#) (void)
Task function of the bootloader application. Should be called continuously in the program loop.

7.56.1 Detailed Description

User program application header file.

7.56.2 Function Documentation

7.56.2.1 AppInit()

```
void AppInit (
    void )
```

Initializes the bootloader application. Should be called once during software program initialization.

Returns

none.

Initializes the bootloader application. Should be called once during software program initialization.

Returns

none.

7.56.2.2 AppTask()

```
void AppTask (
    void )
```

Task function of the bootloader application. Should be called continuously in the program loop.

Returns

none.

Task function of the bootloader application. Should be called continuously in the program loop.

Returns

none.

7.57 app.h File Reference

Bootloader application header file.

Functions

- void [AppInit](#) (void)
Initializes the bootloader application. Should be called once during software program initialization.
- void [AppTask](#) (void)
Task function of the bootloader application. Should be called continuously in the program loop.

7.57.1 Detailed Description

Bootloader application header file.

7.57.2 Function Documentation

7.57.2.1 AppInit()

```
void AppInit (  
    void )
```

Initializes the bootloader application. Should be called once during software program initialization.

Returns

none.

Initializes the bootloader application. Should be called once during software program initialization.

Returns

none.

7.57.2.2 AppTask()

```
void AppTask (  
    void )
```

Task function of the bootloader application. Should be called continuously in the program loop.

Returns

none.

Task function of the bootloader application. Should be called continuously in the program loop.

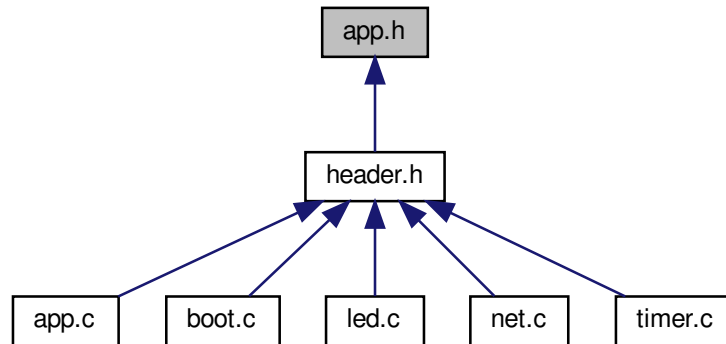
Returns

none.

7.58 app.h File Reference

User program application header file.

This graph shows which files directly or indirectly include this file:



Functions

- void [AppInit](#) (void)
Initializes the bootloader application. Should be called once during software program initialization.
- void [AppTask](#) (void)
Task function of the bootloader application. Should be called continuously in the program loop.

7.58.1 Detailed Description

User program application header file.

7.58.2 Function Documentation

7.58.2.1 AppInit()

```
void AppInit (  
    void )
```

Initializes the bootloader application. Should be called once during software program initialization.

Returns

none.

Initializes the bootloader application. Should be called once during software program initialization.

Returns

none.

7.58.2.2 AppTask()

```
void AppTask (
    void )
```

Task function of the bootloader application. Should be called continuously in the program loop.

Returns

none.

Task function of the bootloader application. Should be called continuously in the program loop.

Returns

none.

7.59 app.h File Reference

Bootloader application header file.

Functions

- void [Applnit](#) (void)
Initializes the bootloader application. Should be called once during software program initialization.
- void [AppTask](#) (void)
Task function of the bootloader application. Should be called continuously in the program loop.

7.59.1 Detailed Description

Bootloader application header file.

7.59.2 Function Documentation

7.59.2.1 Applnit()

```
void Applnit (
    void )
```

Initializes the bootloader application. Should be called once during software program initialization.

Returns

none.

Initializes the bootloader application. Should be called once during software program initialization.

Returns

none.

7.59.2.2 AppTask()

```
void AppTask (  
    void )
```

Task function of the bootloader application. Should be called continuously in the program loop.

Returns

none.

Task function of the bootloader application. Should be called continuously in the program loop.

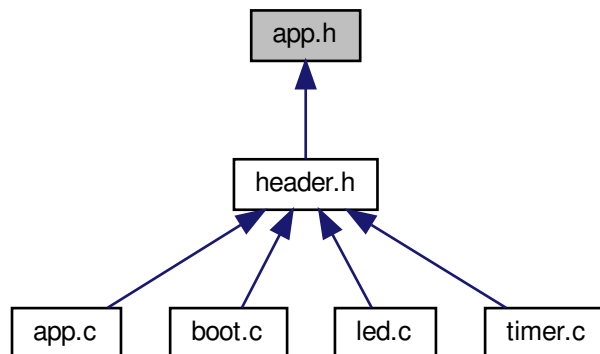
Returns

none.

7.60 app.h File Reference

User program application header file.

This graph shows which files directly or indirectly include this file:



Functions

- void [AppInit](#) (void)
Initializes the bootloader application. Should be called once during software program initialization.
- void [AppTask](#) (void)
Task function of the bootloader application. Should be called continuously in the program loop.

7.60.1 Detailed Description

User program application header file.

7.60.2 Function Documentation

7.60.2.1 AppInit()

```
void AppInit (  
    void )
```

Initializes the bootloader application. Should be called once during software program initialization.

Returns

none.

Initializes the bootloader application. Should be called once during software program initialization.

Returns

none.

7.60.2.2 AppTask()

```
void AppTask (  
    void )
```

Task function of the bootloader application. Should be called continuously in the program loop.

Returns

none.

Task function of the bootloader application. Should be called continuously in the program loop.

Returns

none.

7.61 app.h File Reference

Bootloader application header file.

Functions

- void [AppInit](#) (void)
Initializes the bootloader application. Should be called once during software program initialization.
- void [AppTask](#) (void)
Task function of the bootloader application. Should be called continuously in the program loop.

7.61.1 Detailed Description

Bootloader application header file.

7.61.2 Function Documentation

7.61.2.1 AppInit()

```
void AppInit (  
    void )
```

Initializes the bootloader application. Should be called once during software program initialization.

Returns

none.

Initializes the bootloader application. Should be called once during software program initialization.

Returns

none.

7.61.2.2 AppTask()

```
void AppTask (  
    void )
```

Task function of the bootloader application. Should be called continuously in the program loop.

Returns

none.

Task function of the bootloader application. Should be called continuously in the program loop.

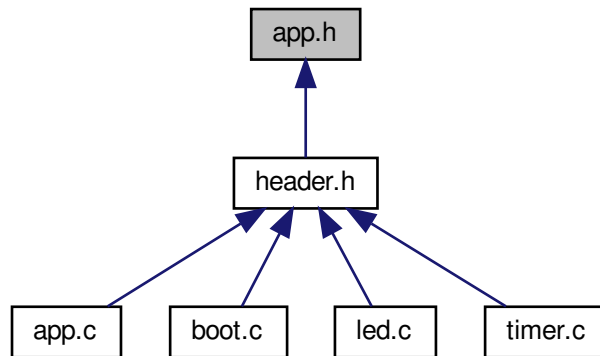
Returns

none.

7.62 app.h File Reference

User program application header file.

This graph shows which files directly or indirectly include this file:



Functions

- void [AppInit](#) (void)
Initializes the bootloader application. Should be called once during software program initialization.
- void [AppTask](#) (void)
Task function of the bootloader application. Should be called continuously in the program loop.

7.62.1 Detailed Description

User program application header file.

7.62.2 Function Documentation

7.62.2.1 AppInit()

```
void AppInit (  
    void )
```

Initializes the bootloader application. Should be called once during software program initialization.

Returns

none.

Initializes the bootloader application. Should be called once during software program initialization.

Returns

none.

7.62.2.2 AppTask()

```
void AppTask (
    void )
```

Task function of the bootloader application. Should be called continuously in the program loop.

Returns

none.

Task function of the bootloader application. Should be called continuously in the program loop.

Returns

none.

7.63 app.h File Reference

Bootloader application header file.

Functions

- void [Applnit](#) (void)
Initializes the bootloader application. Should be called once during software program initialization.
- void [AppTask](#) (void)
Task function of the bootloader application. Should be called continuously in the program loop.

7.63.1 Detailed Description

Bootloader application header file.

7.63.2 Function Documentation

7.63.2.1 Applnit()

```
void Applnit (
    void )
```

Initializes the bootloader application. Should be called once during software program initialization.

Returns

none.

Initializes the bootloader application. Should be called once during software program initialization.

Returns

none.

7.63.2.2 AppTask()

```
void AppTask (
    void )
```

Task function of the bootloader application. Should be called continuously in the program loop.

Returns

none.

Task function of the bootloader application. Should be called continuously in the program loop.

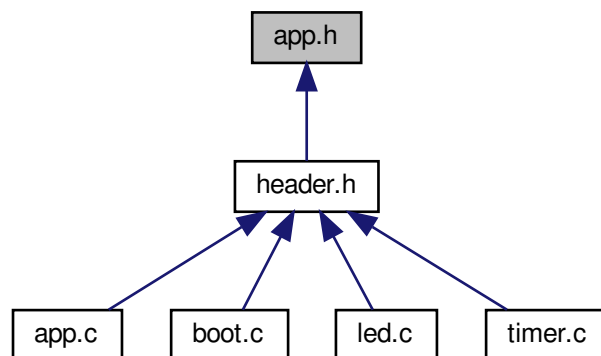
Returns

none.

7.64 app.h File Reference

User program application header file.

This graph shows which files directly or indirectly include this file:



Functions

- void [AppInit](#) (void)
Initializes the bootloader application. Should be called once during software program initialization.
- void [AppTask](#) (void)
Task function of the bootloader application. Should be called continuously in the program loop.

7.64.1 Detailed Description

User program application header file.

7.64.2 Function Documentation

7.64.2.1 AppInit()

```
void AppInit (
    void )
```

Initializes the bootloader application. Should be called once during software program initialization.

Returns

none.

Initializes the bootloader application. Should be called once during software program initialization.

Returns

none.

7.64.2.2 AppTask()

```
void AppTask (
    void )
```

Task function of the bootloader application. Should be called continuously in the program loop.

Returns

none.

Task function of the bootloader application. Should be called continuously in the program loop.

Returns

none.

7.65 app.h File Reference

Bootloader application header file.

Functions

- void [AppInit](#) (void)
Initializes the bootloader application. Should be called once during software program initialization.
- void [AppTask](#) (void)
Task function of the bootloader application. Should be called continuously in the program loop.

7.65.1 Detailed Description

Bootloader application header file.

7.65.2 Function Documentation

7.65.2.1 AppInit()

```
void AppInit (  
    void )
```

Initializes the bootloader application. Should be called once during software program initialization.

Returns

none.

Initializes the bootloader application. Should be called once during software program initialization.

Returns

none.

7.65.2.2 AppTask()

```
void AppTask (  
    void )
```

Task function of the bootloader application. Should be called continuously in the program loop.

Returns

none.

Task function of the bootloader application. Should be called continuously in the program loop.

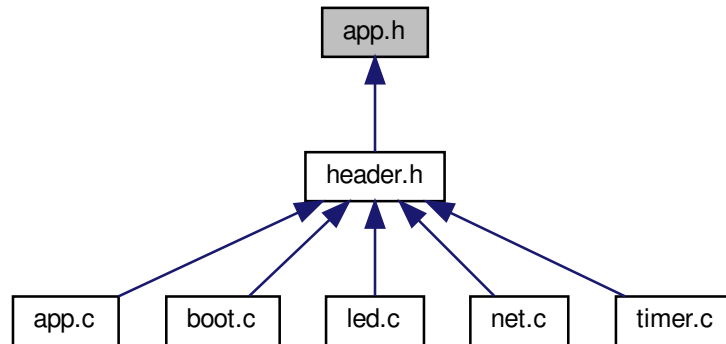
Returns

none.

7.66 app.h File Reference

User program application header file.

This graph shows which files directly or indirectly include this file:



Functions

- void [AppInit](#) (void)
Initializes the bootloader application. Should be called once during software program initialization.
- void [AppTask](#) (void)
Task function of the bootloader application. Should be called continuously in the program loop.

7.66.1 Detailed Description

User program application header file.

7.66.2 Function Documentation

7.66.2.1 AppInit()

```
void AppInit (
    void )
```

Initializes the bootloader application. Should be called once during software program initialization.

Returns

none.

Initializes the bootloader application. Should be called once during software program initialization.

Returns

none.

7.66.2.2 AppTask()

```
void AppTask (
    void )
```

Task function of the bootloader application. Should be called continuously in the program loop.

Returns

none.

Task function of the bootloader application. Should be called continuously in the program loop.

Returns

none.

7.67 app.h File Reference

Bootloader application header file.

Functions

- void [Applnit](#) (void)
Initializes the bootloader application. Should be called once during software program initialization.
- void [AppTask](#) (void)
Task function of the bootloader application. Should be called continuously in the program loop.

7.67.1 Detailed Description

Bootloader application header file.

7.67.2 Function Documentation

7.67.2.1 Applnit()

```
void Applnit (
    void )
```

Initializes the bootloader application. Should be called once during software program initialization.

Returns

none.

Initializes the bootloader application. Should be called once during software program initialization.

Returns

none.

7.67.2.2 AppTask()

```
void AppTask (  
    void )
```

Task function of the bootloader application. Should be called continuously in the program loop.

Returns

none.

Task function of the bootloader application. Should be called continuously in the program loop.

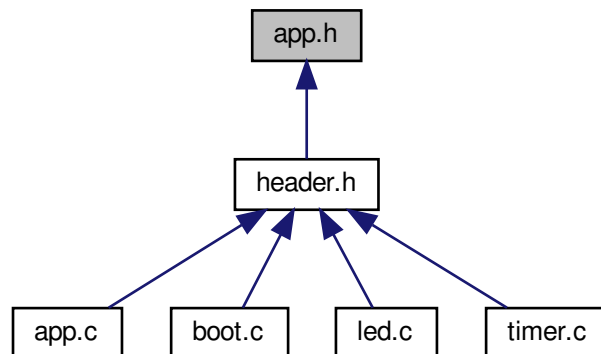
Returns

none.

7.68 app.h File Reference

User program application header file.

This graph shows which files directly or indirectly include this file:



Functions

- void [AppInit](#) (void)
Initializes the bootloader application. Should be called once during software program initialization.
- void [AppTask](#) (void)
Task function of the bootloader application. Should be called continuously in the program loop.

7.68.1 Detailed Description

User program application header file.

7.68.2 Function Documentation

7.68.2.1 AppInit()

```
void AppInit (  
    void )
```

Initializes the bootloader application. Should be called once during software program initialization.

Returns

none.

Initializes the bootloader application. Should be called once during software program initialization.

Returns

none.

7.68.2.2 AppTask()

```
void AppTask (  
    void )
```

Task function of the bootloader application. Should be called continuously in the program loop.

Returns

none.

Task function of the bootloader application. Should be called continuously in the program loop.

Returns

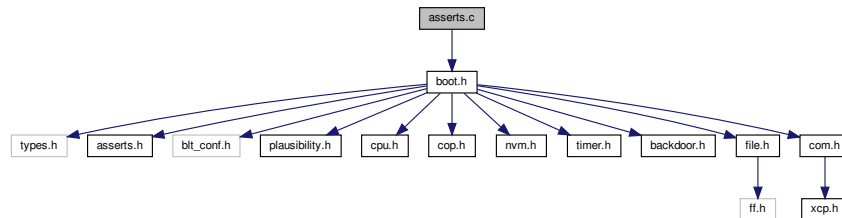
none.

7.69 asserts.c File Reference

Bootloader assertion module source file.

```
#include "boot.h"
```

Include dependency graph for asserts.c:



Functions

- void [AssertFailure](#) (blt_char *file, blt_int32u line)

Called when a runtime assertion failed. It stores information about where the assertion occurred and halts the software program.

7.69.1 Detailed Description

Bootloader assertion module source file.

7.69.2 Function Documentation

7.69.2.1 AssertFailure()

```
void AssertFailure (
    blt_char * file,
    blt_int32u line )
```

Called when a runtime assertion failed. It stores information about where the assertion occurred and halts the software program.

Parameters

<i>file</i>	Name of the source file where the assertion occurred.
<i>line</i>	Linenumber in the source file where the assertion occurred.

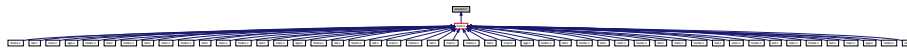
Returns

none

7.70 asserts.h File Reference

Bootloader assertion module header file.

This graph shows which files directly or indirectly include this file:



Macros

- `#define ASSERT_CT(cond) enum { ASSERT_CONCAT(assert_error_on_line_, __LINE__) = 1/(!(cond)) }`
Macro for assertions that can be performed at compile time.
- `#define ASSERT_RT(cond)`
Macro for assertions that can only be performed at run time.

Functions

- `void AssertFailure (blt_char *file, blt_int32u line)`
Called when a runtime assertion failed. It stores information about where the assertion occurred and halts the software program.

7.70.1 Detailed Description

Bootloader assertion module header file.

7.70.2 Function Documentation

7.70.2.1 AssertFailure()

```
void AssertFailure (
    blt_char * file,
    blt_int32u line )
```

Called when a runtime assertion failed. It stores information about where the assertion occurred and halts the software program.

Parameters

<i>file</i>	Name of the source file where the assertion occurred.
<i>line</i>	Linenumber in the source file where the assertion occurred.

Returns

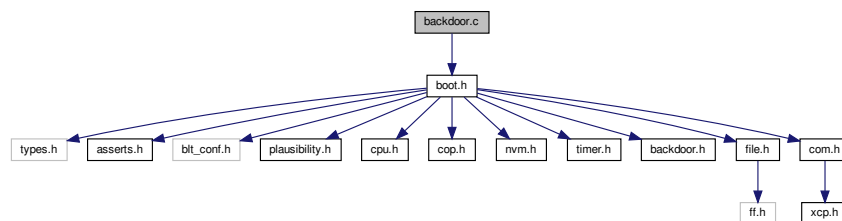
none

7.71 backdoor.c File Reference

Bootloader backdoor entry source file.

```
#include "boot.h"
```

Include dependency graph for backdoor.c:



Functions

- void [BackDoorInitHook](#) (void)
Initializes the backdoor entry option.
- [blt_bool BackDoorEntryHook](#) (void)
Checks if a backdoor entry is requested.
- void [BackDoorInit](#) (void)
Initializes the backdoor entry option.
- void [BackDoorCheck](#) (void)
The default backdoor entry feature keeps the bootloader active for a predetermined time after reset, allowing the host application to establish a connection and start a programming sequence. This function controls the opening/closing of the backdoor.

7.71.1 Detailed Description

Bootloader backdoor entry source file.

7.71.2 Function Documentation

7.71.2.1 BackDoorCheck()

```
void BackDoorCheck (
    void )
```

The default backdoor entry feature keeps the bootloader active for a predetermined time after reset, allowing the host application to establish a connection and start a programming sequence. This function controls the opening/closing of the backdoor.

Returns

none

Referenced by BackDoorInit(), and BootTask().

7.71.2.2 BackDoorEntryHook()

```
blt_bool BackDoorEntryHook (
    void )
```

Checks if a backdoor entry is requested.

Returns

BLT_TRUE if the backdoor entry is requested, BLT_FALSE otherwise.

Referenced by BackDoorInit(), and UsbLeaveLowPowerModeHook().

7.71.2.3 BackDoorInit()

```
void BackDoorInit (
    void )
```

Initializes the backdoor entry option.

Returns

none

Referenced by BootInit().

7.71.2.4 BackDoorInitHook()

```
void BackDoorInitHook (
    void )
```

Initializes the backdoor entry option.

Returns

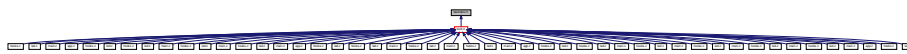
none.

Referenced by BackDoorInit(), and UsbLeaveLowPowerModeHook().

7.72 backdoor.h File Reference

Bootloader backdoor entry header file.

This graph shows which files directly or indirectly include this file:



Functions

- void [BackDoorInit](#) (void)
Initializes the backdoor entry option.
- void [BackDoorCheck](#) (void)
The default backdoor entry feature keeps the bootloader active for a predetermined time after reset, allowing the host application to establish a connection and start a programming sequence. This function controls the opening/closing of the backdoor.

7.72.1 Detailed Description

Bootloader backdoor entry header file.

7.72.2 Function Documentation

7.72.2.1 BackDoorCheck()

```
void BackDoorCheck (
    void )
```

The default backdoor entry feature keeps the bootloader active for a predetermined time after reset, allowing the host application to establish a connection and start a programming sequence. This function controls the opening/closing of the backdoor.

Returns

none

Referenced by BackDoorInit(), and BootTask().

7.72.2.2 BackDoorInit()

```
void BackDoorInit (
    void )
```

Initializes the backdoor entry option.

Returns

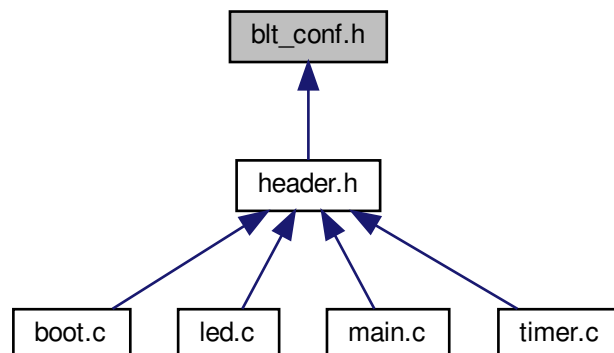
none

Referenced by BootInit().

7.73 blt_conf.h File Reference

Bootloader configuration header file.

This graph shows which files directly or indirectly include this file:



Macros

- #define `BOOT_CPU_XTAL_SPEED_KHZ` (8000)
Frequency of the external crystal oscillator.
- #define `BOOT_CPU_SYSTEM_SPEED_KHZ` (72000)
Desired system speed.
- #define `BOOT_CPU_BYTE_ORDER_MOTOROLA` (0)
Motorola or Intel style byte ordering.
- #define `BOOT_CPU_USER_PROGRAM_START_HOOK` (1)
Enable/disable hook function call right before user program start.
- #define `BOOT_COM_RS232_ENABLE` (1)
Enable/disable UART transport layer.
- #define `BOOT_COM_RS232_BAUDRATE` (57600)
Configure the desired communication speed.
- #define `BOOT_COM_RS232_TX_MAX_DATA` (64)
Configure number of bytes in the target->host data packet.
- #define `BOOT_COM_RS232_RX_MAX_DATA` (64)
Configure number of bytes in the host->target data packet.
- #define `BOOT_COM_RS232_CHANNEL_INDEX` (1)
Select the desired UART peripheral as a zero based index.
- #define `BOOT_BACKDOOR_HOOKS_ENABLE` (0)
Enable/disable the backdoor override hook functions.
- #define `BOOT_NVM_HOOKS_ENABLE` (0)
Enable/disable the NVM hook function for supporting additional memory devices.
- #define `BOOT_NVM_SIZE_KB` (128)
Configure the size of the default memory device (typically flash EEPROM).
- #define `BOOT_NVM_CHECKSUM_HOOKS_ENABLE` (0)
Enable/disable hooks functions to override the user program checksum handling.
- #define `BOOT_COP_HOOKS_ENABLE` (1)
Enable/disable the hook functions for controlling the watchdog.

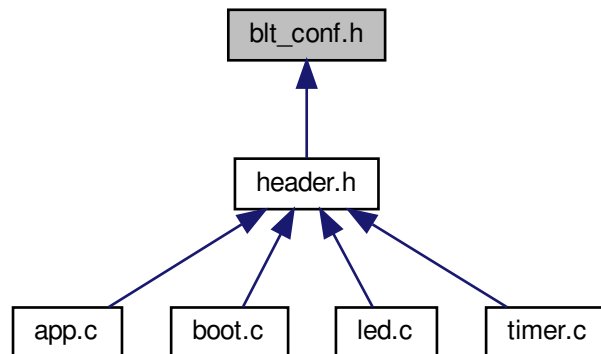
7.73.1 Detailed Description

Bootloader configuration header file.

7.74 blt_conf.h File Reference

Bootloader configuration header file.

This graph shows which files directly or indirectly include this file:



Macros

- `#define BOOT_CPU_XTAL_SPEED_KHZ (8000)`
Frequency of the external crystal oscillator.
- `#define BOOT_CPU_SYSTEM_SPEED_KHZ (48000)`
Desired system speed.
- `#define BOOT_CPU_BYTE_ORDER_MOTOROLA (0)`
Motorola or Intel style byte ordering.
- `#define BOOT_CPU_USER_PROGRAM_START_HOOK (1)`
Enable/disable hook function call right before user program start.
- `#define BOOT_COM_RS232_ENABLE (1)`
Enable/disable UART transport layer.
- `#define BOOT_COM_RS232_BAUDRATE (57600)`
Configure the desired communication speed.
- `#define BOOT_COM_RS232_TX_MAX_DATA (64)`
Configure number of bytes in the target->host data packet.
- `#define BOOT_COM_RS232_RX_MAX_DATA (64)`
Configure number of bytes in the host->target data packet.
- `#define BOOT_COM_RS232_CHANNEL_INDEX (1)`
Select the desired UART peripheral as a zero based index.
- `#define BOOT_BACKDOOR_HOOKS_ENABLE (0)`
Enable/disable the backdoor override hook functions.
- `#define BOOT_NVM_HOOKS_ENABLE (0)`
Enable/disable the NVM hook function for supporting additional memory devices.
- `#define BOOT_NVM_SIZE_KB (64)`
Configure the size of the default memory device (typically flash EEPROM).
- `#define BOOT_NVM_CHECKSUM_HOOKS_ENABLE (0)`
Enable/disable hooks functions to override the user program checksum handling.
- `#define BOOT_FLASH_CUSTOM_LAYOUT_ENABLE (1)`
Enable support for a custom flash layout table. It is located in flash_layout.c. This was done because the default flashLayout[] table in the bootloader's core has more flash memory reserved for the bootloader than is needed for this demo.
- `#define BOOT_COP_HOOKS_ENABLE (1)`
Enable/disable the hook functions for controlling the watchdog.

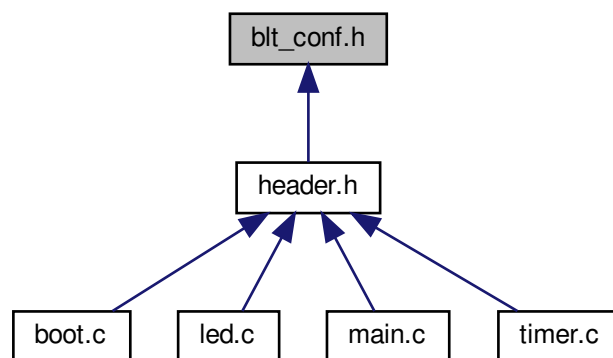
7.74.1 Detailed Description

Bootloader configuration header file.

7.75 blt_conf.h File Reference

Bootloader configuration header file.

This graph shows which files directly or indirectly include this file:



Macros

- `#define BOOT_CPU_XTAL_SPEED_KHZ (8000)`
Frequency of the external crystal oscillator.
- `#define BOOT_CPU_SYSTEM_SPEED_KHZ (48000)`
Desired system speed.
- `#define BOOT_CPU_BYTE_ORDER_MOTOROLA (0)`
Motorola or Intel style byte ordering.
- `#define BOOT_CPU_USER_PROGRAM_START_HOOK (1)`
Enable/disable hook function call right before user program start.
- `#define BOOT_COM_RS232_ENABLE (1)`
Enable/disable UART transport layer.
- `#define BOOT_COM_RS232_BAUDRATE (57600)`
Configure the desired communication speed.
- `#define BOOT_COM_RS232_TX_MAX_DATA (64)`
Configure number of bytes in the target->host data packet.
- `#define BOOT_COM_RS232_RX_MAX_DATA (64)`
Configure number of bytes in the host->target data packet.
- `#define BOOT_COM_RS232_CHANNEL_INDEX (1)`
Select the desired UART peripheral as a zero based index.
- `#define BOOT_BACKDOOR_HOOKS_ENABLE (0)`

- *Enable/disable the backdoor override hook functions.*
• #define `BOOT_NVM_HOOKS_ENABLE` (0)
Enable/disable the NVM hook function for supporting additional memory devices.
- #define `BOOT_NVM_SIZE_KB` (64)
Configure the size of the default memory device (typically flash EEPROM).
- #define `BOOT_NVM_CHECKSUM_HOOKS_ENABLE` (0)
Enable/disable hooks functions to override the user program checksum handling.
- #define `BOOT_FLASH_CUSTOM_LAYOUT_ENABLE` (1)
Enable support for a custom flash layout table. It is located in flash_layout.c. This was done because the default flashLayout[] table in the bootloader's core has more flash memory reserved for the bootloader than is needed for this demo.
- #define `BOOT_COP_HOOKS_ENABLE` (1)
Enable/disable the hook functions for controlling the watchdog.

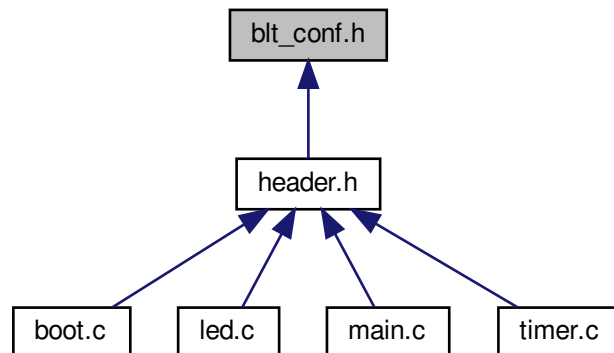
7.75.1 Detailed Description

Bootloader configuration header file.

7.76 blt_conf.h File Reference

Bootloader configuration header file.

This graph shows which files directly or indirectly include this file:



Macros

- #define `BOOT_CPU_XTAL_SPEED_KHZ` (8000)
Frequency of the external crystal oscillator.
- #define `BOOT_CPU_SYSTEM_SPEED_KHZ` (48000)
Desired system speed.
- #define `BOOT_CPU_BYTE_ORDER_MOTOROLA` (0)

Motorola or Intel style byte ordering.

- #define `BOOT_CPU_USER_PROGRAM_START_HOOK` (1)
Enable/disable hook function call right before user program start.
- #define `BOOT_COM_RS232_ENABLE` (1)
Enable/disable UART transport layer.
- #define `BOOT_COM_RS232_BAUDRATE` (57600)
Configure the desired communication speed.
- #define `BOOT_COM_RS232_TX_MAX_DATA` (64)
Configure number of bytes in the target->host data packet.
- #define `BOOT_COM_RS232_RX_MAX_DATA` (64)
Configure number of bytes in the host->target data packet.
- #define `BOOT_COM_RS232_CHANNEL_INDEX` (1)
Select the desired UART peripheral as a zero based index.
- #define `BOOT_BACKDOOR_HOOKS_ENABLE` (0)
Enable/disable the backdoor override hook functions.
- #define `BOOT_NVM_HOOKS_ENABLE` (0)
Enable/disable the NVM hook function for supporting additional memory devices.
- #define `BOOT_NVM_SIZE_KB` (64)
Configure the size of the default memory device (typically flash EEPROM).
- #define `BOOT_NVM_CHECKSUM_HOOKS_ENABLE` (0)
Enable/disable hooks functions to override the user program checksum handling.
- #define `BOOT_FLASH_CUSTOM_LAYOUT_ENABLE` (1)
Enable support for a custom flash layout table. It is located in flash_layout.c. This was done because the default flashLayout[] table in the bootloader's core has more flash memory reserved for the bootloader than is needed for this demo.
- #define `BOOT_COP_HOOKS_ENABLE` (1)
Enable/disable the hook functions for controlling the watchdog.

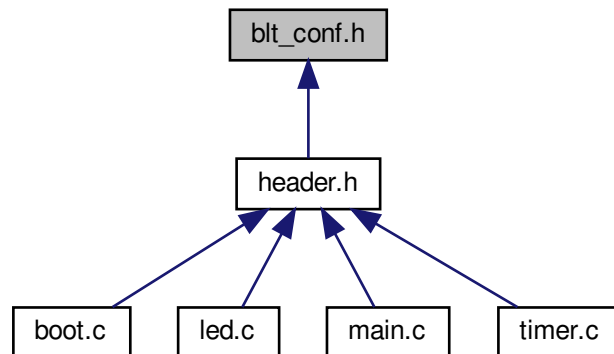
7.76.1 Detailed Description

Bootloader configuration header file.

7.77 blt_conf.h File Reference

Bootloader configuration header file.

This graph shows which files directly or indirectly include this file:



Macros

- `#define BOOT_CPU_XTAL_SPEED_KHZ (8000)`
Frequency of the external crystal oscillator.
- `#define BOOT_CPU_SYSTEM_SPEED_KHZ (48000)`
Desired system speed.
- `#define BOOT_CPU_BYTE_ORDER_MOTOROLA (0)`
Motorola or Intel style byte ordering.
- `#define BOOT_CPU_USER_PROGRAM_START_HOOK (1)`
Enable/disable hook function call right before user program start.
- `#define BOOT_COM_RS232_ENABLE (1)`
Enable/disable UART transport layer.
- `#define BOOT_COM_RS232_BAUDRATE (57600)`
Configure the desired communication speed.
- `#define BOOT_COM_RS232_TX_MAX_DATA (64)`
Configure number of bytes in the target->host data packet.
- `#define BOOT_COM_RS232_RX_MAX_DATA (64)`
Configure number of bytes in the host->target data packet.
- `#define BOOT_COM_RS232_CHANNEL_INDEX (1)`
Select the desired UART peripheral as a zero based index.
- `#define BOOT_BACKDOOR_HOOKS_ENABLE (0)`
Enable/disable the backdoor override hook functions.
- `#define BOOT_NVM_HOOKS_ENABLE (0)`
Enable/disable the NVM hook function for supporting additional memory devices.
- `#define BOOT_NVM_SIZE_KB (64)`
Configure the size of the default memory device (typically flash EEPROM).
- `#define BOOT_NVM_CHECKSUM_HOOKS_ENABLE (0)`
Enable/disable hooks functions to override the user program checksum handling.
- `#define BOOT_FLASH_CUSTOM_LAYOUT_ENABLE (1)`
Enable support for a custom flash layout table. It is located in flash_layout.c. This was done because the default flashLayout[] table in the bootloader's core has more flash memory reserved for the bootloader than is needed for this demo.
- `#define BOOT_COP_HOOKS_ENABLE (1)`
Enable/disable the hook functions for controlling the watchdog.

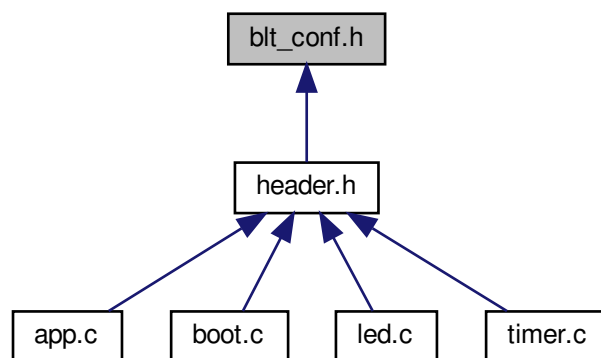
7.77.1 Detailed Description

Bootloader configuration header file.

7.78 blt_conf.h File Reference

Bootloader configuration header file.

This graph shows which files directly or indirectly include this file:



Macros

- #define `BOOT_CPU_XTAL_SPEED_KHZ` (8000)
Frequency of the external crystal oscillator.
- #define `BOOT_CPU_SYSTEM_SPEED_KHZ` (48000)
Desired system speed.
- #define `BOOT_CPU_BYTE_ORDER_MOTOROLA` (0)
Motorola or Intel style byte ordering.
- #define `BOOT_CPU_USER_PROGRAM_START_HOOK` (1)
Enable/disable hook function call right before user program start.
- #define `BOOT_COM_CAN_ENABLE` (1)
Enable/disable CAN transport layer.
- #define `BOOT_COM_CAN_BAUDRATE` (500000)
Configure the desired CAN baudrate.
- #define `BOOT_COM_CAN_TX_MSG_ID` (0x7E1 /*| 0x80000000*/) *Configure CAN message ID target->host.*
- #define `BOOT_COM_CAN_TX_MAX_DATA` (8)
Configure number of bytes in the target->host CAN message.
- #define `BOOT_COM_CAN_RX_MSG_ID` (0x667 /*| 0x80000000*/) *Configure CAN message ID host->target.*
- #define `BOOT_COM_CAN_RX_MAX_DATA` (8)

- *Configure number of bytes in the host->target CAN message.*
 • #define `BOOT_COM_CAN_CHANNEL_INDEX` (0)
Select the desired CAN peripheral as a zero based index.
- #define `BOOT_COM_RS232_ENABLE` (1)
Enable/disable UART transport layer.
- #define `BOOT_COM_RS232_BAUDRATE` (57600)
Configure the desired communication speed.
- #define `BOOT_COM_RS232_TX_MAX_DATA` (64)
Configure number of bytes in the target->host data packet.
- #define `BOOT_COM_RS232_RX_MAX_DATA` (64)
Configure number of bytes in the host->target data packet.
- #define `BOOT_COM_RS232_CHANNEL_INDEX` (1)
Select the desired UART peripheral as a zero based index.
- #define `BOOT_BACKDOOR_HOOKS_ENABLE` (0)
Enable/disable the backdoor override hook functions.
- #define `BOOT_NVM_HOOKS_ENABLE` (0)
Enable/disable the NVM hook function for supporting additional memory devices.
- #define `BOOT_NVM_SIZE_KB` (256)
Configure the size of the default memory device (typically flash EEPROM).
- #define `BOOT_NVM_CHECKSUM_HOOKS_ENABLE` (0)
Enable/disable hooks functions to override the user program checksum handling.
- #define `BOOT_COP_HOOKS_ENABLE` (1)
Enable/disable the hook functions for controlling the watchdog.

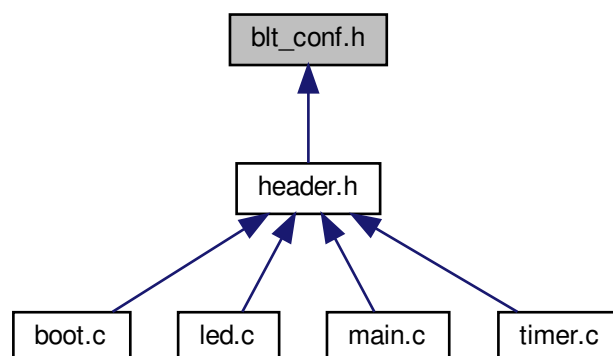
7.78.1 Detailed Description

Bootloader configuration header file.

7.79 blt_conf.h File Reference

Bootloader configuration header file.

This graph shows which files directly or indirectly include this file:



Macros

- #define `BOOT_CPU_XTAL_SPEED_KHZ` (8000)
Frequency of the external crystal oscillator.
- #define `BOOT_CPU_SYSTEM_SPEED_KHZ` (48000)
Desired system speed.
- #define `BOOT_CPU_BYTE_ORDER_MOTOROLA` (0)
Motorola or Intel style byte ordering.
- #define `BOOT_CPU_USER_PROGRAM_START_HOOK` (1)
Enable/disable hook function call right before user program start.
- #define `BOOT_COM_CAN_ENABLE` (1)
Enable/disable CAN transport layer.
- #define `BOOT_COM_CAN_BAUDRATE` (500000)
Configure the desired CAN baudrate.
- #define `BOOT_COM_CAN_TX_MSG_ID` (0x7E1 /*| 0x80000000*/) *Configure CAN message ID target->host.*
- #define `BOOT_COM_CAN_TX_MAX_DATA` (8)
Configure number of bytes in the target->host CAN message.
- #define `BOOT_COM_CAN_RX_MSG_ID` (0x667 /*| 0x80000000*/) *Configure CAN message ID host->target.*
- #define `BOOT_COM_CAN_RX_MAX_DATA` (8)
Configure number of bytes in the host->target CAN message.
- #define `BOOT_COM_CAN_CHANNEL_INDEX` (0)
Select the desired CAN peripheral as a zero based index.
- #define `BOOT_COM_RS232_ENABLE` (1)
Enable/disable UART transport layer.
- #define `BOOT_COM_RS232_BAUDRATE` (57600)
Configure the desired communication speed.
- #define `BOOT_COM_RS232_TX_MAX_DATA` (64)
Configure number of bytes in the target->host data packet.
- #define `BOOT_COM_RS232_RX_MAX_DATA` (64)
Configure number of bytes in the host->target data packet.
- #define `BOOT_COM_RS232_CHANNEL_INDEX` (1)
Select the desired UART peripheral as a zero based index.
- #define `BOOT_BACKDOOR_HOOKS_ENABLE` (0)
Enable/disable the backdoor override hook functions.
- #define `BOOT_NVM_HOOKS_ENABLE` (0)
Enable/disable the NVM hook function for supporting additional memory devices.
- #define `BOOT_NVM_SIZE_KB` (256)
Configure the size of the default memory device (typically flash EEPROM).
- #define `BOOT_NVM_CHECKSUM_HOOKS_ENABLE` (0)
Enable/disable hooks functions to override the user program checksum handling.
- #define `BOOT_COP_HOOKS_ENABLE` (1)
Enable/disable the hook functions for controlling the watchdog.

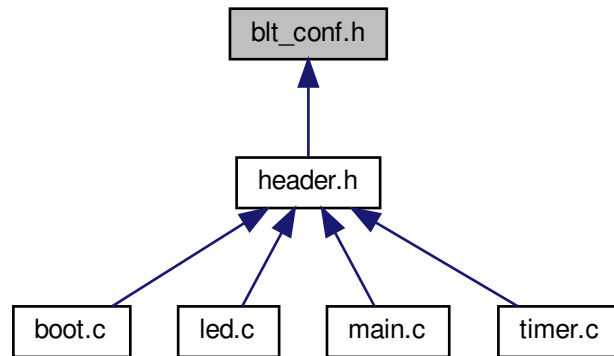
7.79.1 Detailed Description

Bootloader configuration header file.

7.80 blt_conf.h File Reference

Bootloader configuration header file.

This graph shows which files directly or indirectly include this file:



Macros

- `#define BOOT_CPU_XTAL_SPEED_KHZ (8000)`
Frequency of the external crystal oscillator.
- `#define BOOT_CPU_SYSTEM_SPEED_KHZ (48000)`
Desired system speed.
- `#define BOOT_CPU_BYTE_ORDER_MOTOROLA (0)`
Motorola or Intel style byte ordering.
- `#define BOOT_CPU_USER_PROGRAM_START_HOOK (1)`
Enable/disable hook function call right before user program start.
- `#define BOOT_COM_CAN_ENABLE (1)`
Enable/disable CAN transport layer.
- `#define BOOT_COM_CAN_BAUDRATE (500000)`
Configure the desired CAN baudrate.
- `#define BOOT_COM_CAN_TX_MSG_ID (0x7E1 /*| 0x80000000*/)`
Configure CAN message ID target->host.
- `#define BOOT_COM_CAN_TX_MAX_DATA (8)`
Configure number of bytes in the target->host CAN message.
- `#define BOOT_COM_CAN_RX_MSG_ID (0x667 /*| 0x80000000*/)`
Configure CAN message ID host->target.
- `#define BOOT_COM_CAN_RX_MAX_DATA (8)`
Configure number of bytes in the host->target CAN message.
- `#define BOOT_COM_CAN_CHANNEL_INDEX (0)`
Select the desired CAN peripheral as a zero based index.
- `#define BOOT_COM_RS232_ENABLE (1)`
Enable/disable UART transport layer.
- `#define BOOT_COM_RS232_BAUDRATE (57600)`

- Configure the desired communication speed.*

 - #define `BOOT_COM_RS232_TX_MAX_DATA` (64)

Configure number of bytes in the target->host data packet.
- #define `BOOT_COM_RS232_RX_MAX_DATA` (64)

Configure number of bytes in the host->target data packet.
- #define `BOOT_COM_RS232_CHANNEL_INDEX` (1)

Select the desired UART peripheral as a zero based index.
- #define `BOOT_BACKDOOR_HOOKS_ENABLE` (0)

Enable/disable the backdoor override hook functions.
- #define `BOOT_NVM_HOOKS_ENABLE` (0)

Enable/disable the NVM hook function for supporting additional memory devices.
- #define `BOOT_NVM_SIZE_KB` (256)

Configure the size of the default memory device (typically flash EEPROM).
- #define `BOOT_NVM_CHECKSUM_HOOKS_ENABLE` (0)

Enable/disable hooks functions to override the user program checksum handling.
- #define `BOOT_COP_HOOKS_ENABLE` (1)

Enable/disable the hook functions for controlling the watchdog.

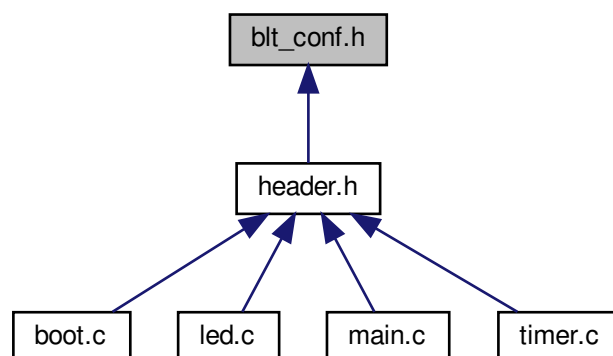
7.80.1 Detailed Description

Bootloader configuration header file.

7.81 blt_conf.h File Reference

Bootloader configuration header file.

This graph shows which files directly or indirectly include this file:



Macros

- #define `BOOT_CPU_XTAL_SPEED_KHZ` (8000)
Frequency of the external crystal oscillator.
- #define `BOOT_CPU_SYSTEM_SPEED_KHZ` (48000)
Desired system speed.
- #define `BOOT_CPU_BYTE_ORDER_MOTOROLA` (0)
Motorola or Intel style byte ordering.
- #define `BOOT_CPU_USER_PROGRAM_START_HOOK` (1)
Enable/disable hook function call right before user program start.
- #define `BOOT_COM_CAN_ENABLE` (1)
Enable/disable CAN transport layer.
- #define `BOOT_COM_CAN_BAUDRATE` (500000)
Configure the desired CAN baudrate.
- #define `BOOT_COM_CAN_TX_MSG_ID` (0x7E1 /*| 0x80000000*/) *Configure CAN message ID target->host.*
- #define `BOOT_COM_CAN_TX_MAX_DATA` (8)
Configure number of bytes in the target->host CAN message.
- #define `BOOT_COM_CAN_RX_MSG_ID` (0x667 /*| 0x80000000*/) *Configure CAN message ID host->target.*
- #define `BOOT_COM_CAN_RX_MAX_DATA` (8)
Configure number of bytes in the host->target CAN message.
- #define `BOOT_COM_CAN_CHANNEL_INDEX` (0)
Select the desired CAN peripheral as a zero based index.
- #define `BOOT_COM_RS232_ENABLE` (1)
Enable/disable UART transport layer.
- #define `BOOT_COM_RS232_BAUDRATE` (57600)
Configure the desired communication speed.
- #define `BOOT_COM_RS232_TX_MAX_DATA` (64)
Configure number of bytes in the target->host data packet.
- #define `BOOT_COM_RS232_RX_MAX_DATA` (64)
Configure number of bytes in the host->target data packet.
- #define `BOOT_COM_RS232_CHANNEL_INDEX` (1)
Select the desired UART peripheral as a zero based index.
- #define `BOOT_BACKDOOR_HOOKS_ENABLE` (0)
Enable/disable the backdoor override hook functions.
- #define `BOOT_NVM_HOOKS_ENABLE` (0)
Enable/disable the NVM hook function for supporting additional memory devices.
- #define `BOOT_NVM_SIZE_KB` (256)
Configure the size of the default memory device (typically flash EEPROM).
- #define `BOOT_NVM_CHECKSUM_HOOKS_ENABLE` (0)
Enable/disable hooks functions to override the user program checksum handling.
- #define `BOOT_COP_HOOKS_ENABLE` (1)
Enable/disable the hook functions for controlling the watchdog.

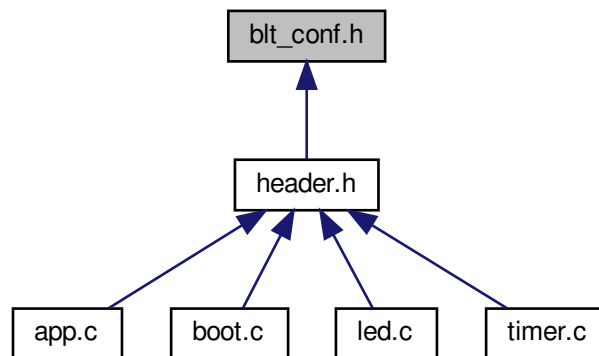
7.81.1 Detailed Description

Bootloader configuration header file.

7.82 blt_conf.h File Reference

Bootloader configuration header file.

This graph shows which files directly or indirectly include this file:



Macros

- `#define BOOT_CPU_XTAL_SPEED_KHZ (16000)`
Frequency of the external crystal oscillator.
- `#define BOOT_CPU_SYSTEM_SPEED_KHZ (64000)`
Desired system speed.
- `#define BOOT_CPU_BYTE_ORDER_MOTOROLA (0)`
Motorola or Intel style byte ordering.
- `#define BOOT_CPU_USER_PROGRAM_START_HOOK (1)`
Enable/disable hook function call right before user program start.
- `#define BOOT_COM_RS232_ENABLE (1)`
Enable/disable UART transport layer.
- `#define BOOT_COM_RS232_BAUDRATE (57600)`
Configure the desired communication speed.
- `#define BOOT_COM_RS232_TX_MAX_DATA (64)`
Configure number of bytes in the target->host data packet.
- `#define BOOT_COM_RS232_RX_MAX_DATA (64)`
Configure number of bytes in the host->target data packet.
- `#define BOOT_COM_RS232_CHANNEL_INDEX (1)`
Select the desired UART peripheral as a zero based index.
- `#define BOOT_BACKDOOR_HOOKS_ENABLE (0)`
Enable/disable the backdoor override hook functions.
- `#define BOOT_NVM_HOOKS_ENABLE (0)`
Enable/disable the NVM hook function for supporting additional memory devices.
- `#define BOOT_NVM_SIZE_KB (128)`
Configure the size of the default memory device (typically flash EEPROM).
- `#define BOOT_NVM_CHECKSUM_HOOKS_ENABLE (0)`
Enable/disable hooks functions to override the user program checksum handling.
- `#define BOOT_COP_HOOKS_ENABLE (1)`
Enable/disable the hook functions for controlling the watchdog.

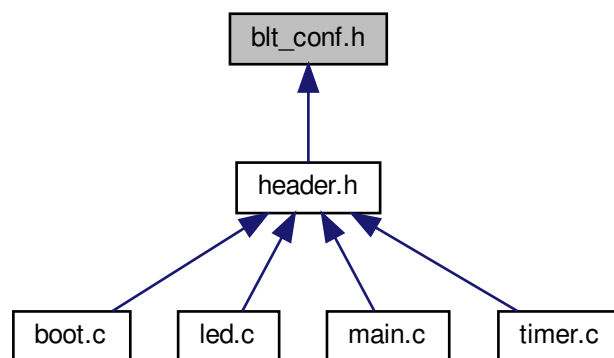
7.82.1 Detailed Description

Bootloader configuration header file.

7.83 blt_conf.h File Reference

Bootloader configuration header file.

This graph shows which files directly or indirectly include this file:



Macros

- `#define BOOT_CPU_XTAL_SPEED_KHZ (16000)`
Frequency of the external crystal oscillator.
- `#define BOOT_CPU_SYSTEM_SPEED_KHZ (64000)`
Desired system speed.
- `#define BOOT_CPU_BYTE_ORDER_MOTOROLA (0)`
Motorola or Intel style byte ordering.
- `#define BOOT_CPU_USER_PROGRAM_START_HOOK (1)`
Enable/disable hook function call right before user program start.
- `#define BOOT_COM_RS232_ENABLE (1)`
Enable/disable UART transport layer.
- `#define BOOT_COM_RS232_BAUDRATE (57600)`
Configure the desired communication speed.
- `#define BOOT_COM_RS232_TX_MAX_DATA (64)`
Configure number of bytes in the target->host data packet.
- `#define BOOT_COM_RS232_RX_MAX_DATA (64)`
Configure number of bytes in the host->target data packet.
- `#define BOOT_COM_RS232_CHANNEL_INDEX (1)`
Select the desired UART peripheral as a zero based index.
- `#define BOOT_BACKDOOR_HOOKS_ENABLE (0)`

- *Enable/disable the backdoor override hook functions.*
• #define `BOOT_NVM_HOOKS_ENABLE` (0)
Enable/disable the NVM hook function for supporting additional memory devices.
- #define `BOOT_NVM_SIZE_KB` (128)
Configure the size of the default memory device (typically flash EEPROM).
- #define `BOOT_NVM_CHECKSUM_HOOKS_ENABLE` (0)
Enable/disable hooks functions to override the user program checksum handling.
- #define `BOOT_COP_HOOKS_ENABLE` (1)
Enable/disable the hook functions for controlling the watchdog.

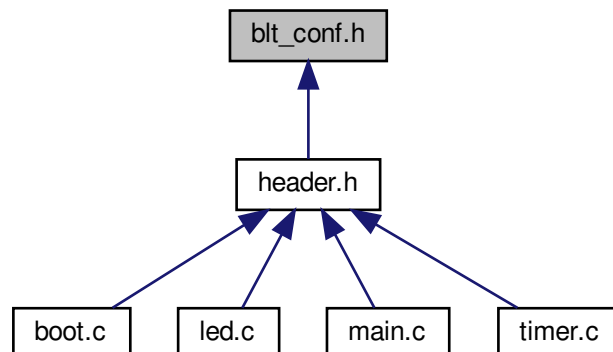
7.83.1 Detailed Description

Bootloader configuration header file.

7.84 blt_conf.h File Reference

Bootloader configuration header file.

This graph shows which files directly or indirectly include this file:



Macros

- #define `BOOT_CPU_XTAL_SPEED_KHZ` (16000)
Frequency of the external crystal oscillator.
- #define `BOOT_CPU_SYSTEM_SPEED_KHZ` (64000)
Desired system speed.
- #define `BOOT_CPU_BYTE_ORDER_MOTOROLA` (0)
Motorola or Intel style byte ordering.
- #define `BOOT_CPU_USER_PROGRAM_START_HOOK` (1)
Enable/disable hook function call right before user program start.

- `#define BOOT_COM_RS232_ENABLE` (1)
Enable/disable UART transport layer.
- `#define BOOT_COM_RS232_BAUDRATE` (57600)
Configure the desired communication speed.
- `#define BOOT_COM_RS232_TX_MAX_DATA` (64)
Configure number of bytes in the target->host data packet.
- `#define BOOT_COM_RS232_RX_MAX_DATA` (64)
Configure number of bytes in the host->target data packet.
- `#define BOOT_COM_RS232_CHANNEL_INDEX` (1)
Select the desired UART peripheral as a zero based index.
- `#define BOOT_BACKDOOR_HOOKS_ENABLE` (0)
Enable/disable the backdoor override hook functions.
- `#define BOOT_NVM_HOOKS_ENABLE` (0)
Enable/disable the NVM hook function for supporting additional memory devices.
- `#define BOOT_NVM_SIZE_KB` (128)
Configure the size of the default memory device (typically flash EEPROM).
- `#define BOOT_NVM_CHECKSUM_HOOKS_ENABLE` (0)
Enable/disable hooks functions to override the user program checksum handling.
- `#define BOOT_COP_HOOKS_ENABLE` (1)
Enable/disable the hook functions for controlling the watchdog.

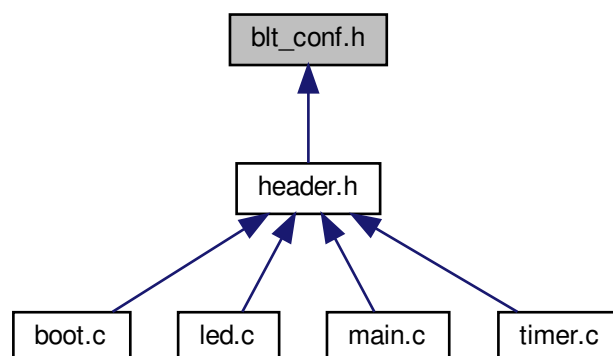
7.84.1 Detailed Description

Bootloader configuration header file.

7.85 blt_conf.h File Reference

Bootloader configuration header file.

This graph shows which files directly or indirectly include this file:



Macros

- #define `BOOT_CPU_XTAL_SPEED_KHZ` (16000)
Frequency of the external crystal oscillator.
- #define `BOOT_CPU_SYSTEM_SPEED_KHZ` (64000)
Desired system speed.
- #define `BOOT_CPU_BYTE_ORDER_MOTOROLA` (0)
Motorola or Intel style byte ordering.
- #define `BOOT_CPU_USER_PROGRAM_START_HOOK` (1)
Enable/disable hook function call right before user program start.
- #define `BOOT_COM_RS232_ENABLE` (1)
Enable/disable UART transport layer.
- #define `BOOT_COM_RS232_BAUDRATE` (57600)
Configure the desired communication speed.
- #define `BOOT_COM_RS232_TX_MAX_DATA` (64)
Configure number of bytes in the target->host data packet.
- #define `BOOT_COM_RS232_RX_MAX_DATA` (64)
Configure number of bytes in the host->target data packet.
- #define `BOOT_COM_RS232_CHANNEL_INDEX` (1)
Select the desired UART peripheral as a zero based index.
- #define `BOOT_BACKDOOR_HOOKS_ENABLE` (0)
Enable/disable the backdoor override hook functions.
- #define `BOOT_NVM_HOOKS_ENABLE` (0)
Enable/disable the NVM hook function for supporting additional memory devices.
- #define `BOOT_NVM_SIZE_KB` (128)
Configure the size of the default memory device (typically flash EEPROM).
- #define `BOOT_NVM_CHECKSUM_HOOKS_ENABLE` (0)
Enable/disable hooks functions to override the user program checksum handling.
- #define `BOOT_COP_HOOKS_ENABLE` (1)
Enable/disable the hook functions for controlling the watchdog.

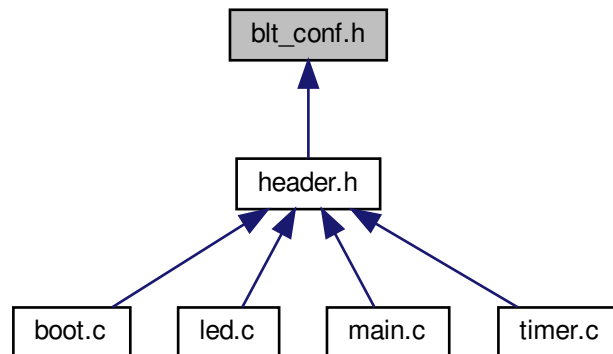
7.85.1 Detailed Description

Bootloader configuration header file.

7.86 blt_conf.h File Reference

Bootloader configuration header file.

This graph shows which files directly or indirectly include this file:



Macros

- `#define BOOT_CPU_XTAL_SPEED_KHZ (20000)`
Frequency of the external crystal oscillator.
- `#define BOOT_CPU_SYSTEM_SPEED_KHZ (48000)`
Desired system speed.
- `#define BOOT_CPU_BYTE_ORDER_MOTOROLA (0)`
Motorola or Intel style byte ordering.
- `#define BOOT_CPU_USER_PROGRAM_START_HOOK (1)`
Enable/disable hook function call right before user program start.
- `#define BOOT_COM_CAN_ENABLE (1)`
Enable/disable CAN transport layer.
- `#define BOOT_COM_CAN_BAUDRATE (500000)`
Configure the desired CAN baudrate.
- `#define BOOT_COM_CAN_TX_MSG_ID (0x7E1 /*| 0x80000000*/)`
Configure CAN message ID target->host.
- `#define BOOT_COM_CAN_TX_MAX_DATA (8)`
Configure number of bytes in the target->host CAN message.
- `#define BOOT_COM_CAN_RX_MSG_ID (0x667 /*| 0x80000000*/)`
Configure CAN message ID host->target.
- `#define BOOT_COM_CAN_RX_MAX_DATA (8)`
Configure number of bytes in the host->target CAN message.
- `#define BOOT_COM_CAN_CHANNEL_INDEX (1)`
Select the desired CAN peripheral as a zero based index.
- `#define BOOT_COM_RS232_ENABLE (1)`
Enable/disable UART transport layer.
- `#define BOOT_COM_RS232_BAUDRATE (57600)`
Configure the desired communication speed.
- `#define BOOT_COM_RS232_TX_MAX_DATA (64)`
Configure number of bytes in the target->host data packet.
- `#define BOOT_COM_RS232_RX_MAX_DATA (64)`

- Configure number of bytes in the host->target data packet.*

 - #define `BOOT_COM_RS232_CHANNEL_INDEX` (1)

Select the desired UART peripheral as a zero based index.
- #define `BOOT_BACKDOOR_HOOKS_ENABLE` (0)

Enable/disable the backdoor override hook functions.
- #define `BOOT_NVM_HOOKS_ENABLE` (0)

Enable/disable the NVM hook function for supporting additional memory devices.
- #define `BOOT_NVM_SIZE_KB` (200)

Configure the size of the default memory device (typically flash EEPROM).
- #define `BOOT_NVM_CHECKSUM_HOOKS_ENABLE` (0)

Enable/disable hooks functions to override the user program checksum handling.
- #define `BOOT_COP_HOOKS_ENABLE` (1)

Enable/disable the hook functions for controlling the watchdog.

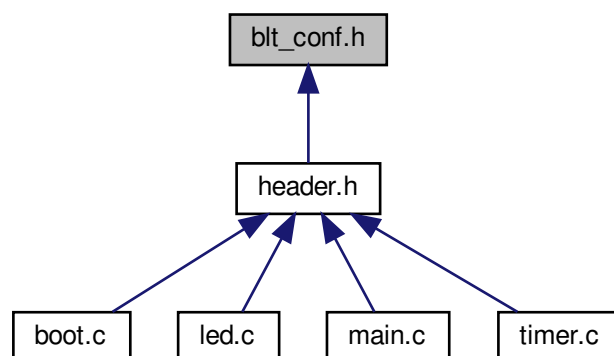
7.86.1 Detailed Description

Bootloader configuration header file.

7.87 blt_conf.h File Reference

Bootloader configuration header file.

This graph shows which files directly or indirectly include this file:



Macros

- #define `BOOT_CPU_XTAL_SPEED_KHZ` (20000)
Frequency of the external crystal oscillator.
- #define `BOOT_CPU_SYSTEM_SPEED_KHZ` (48000)
Desired system speed.
- #define `BOOT_CPU_BYTE_ORDER_MOTOROLA` (0)
Motorola or Intel style byte ordering.
- #define `BOOT_CPU_USER_PROGRAM_START_HOOK` (1)
Enable/disable hook function call right before user program start.
- #define `BOOT_COM_CAN_ENABLE` (1)
Enable/disable CAN transport layer.
- #define `BOOT_COM_CAN_BAUDRATE` (500000)
Configure the desired CAN baudrate.
- #define `BOOT_COM_CAN_TX_MSG_ID` (0x7E1 /*| 0x80000000*/) *Configure CAN message ID target->host.*
- #define `BOOT_COM_CAN_TX_MAX_DATA` (8)
Configure number of bytes in the target->host CAN message.
- #define `BOOT_COM_CAN_RX_MSG_ID` (0x667 /*| 0x80000000*/) *Configure CAN message ID host->target.*
- #define `BOOT_COM_CAN_RX_MAX_DATA` (8)
Configure number of bytes in the host->target CAN message.
- #define `BOOT_COM_CAN_CHANNEL_INDEX` (1)
Select the desired CAN peripheral as a zero based index.
- #define `BOOT_COM_RS232_ENABLE` (1)
Enable/disable UART transport layer.
- #define `BOOT_COM_RS232_BAUDRATE` (57600)
Configure the desired communication speed.
- #define `BOOT_COM_RS232_TX_MAX_DATA` (64)
Configure number of bytes in the target->host data packet.
- #define `BOOT_COM_RS232_RX_MAX_DATA` (64)
Configure number of bytes in the host->target data packet.
- #define `BOOT_COM_RS232_CHANNEL_INDEX` (1)
Select the desired UART peripheral as a zero based index.
- #define `BOOT_BACKDOOR_HOOKS_ENABLE` (0)
Enable/disable the backdoor override hook functions.
- #define `BOOT_NVM_HOOKS_ENABLE` (0)
Enable/disable the NVM hook function for supporting additional memory devices.
- #define `BOOT_NVM_SIZE_KB` (200)
Configure the size of the default memory device (typically flash EEPROM).
- #define `BOOT_NVM_CHECKSUM_HOOKS_ENABLE` (0)
Enable/disable hooks functions to override the user program checksum handling.
- #define `BOOT_COP_HOOKS_ENABLE` (1)
Enable/disable the hook functions for controlling the watchdog.

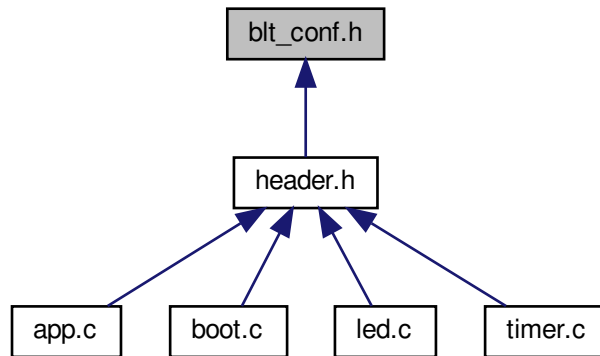
7.87.1 Detailed Description

Bootloader configuration header file.

7.88 blt_conf.h File Reference

Bootloader configuration header file.

This graph shows which files directly or indirectly include this file:



Macros

- `#define BOOT_CPU_XTAL_SPEED_KHZ (16000)`
Frequency of the external crystal or internal (HSI/MSI) oscillator.
- `#define BOOT_CPU_SYSTEM_SPEED_KHZ (96000)`
Desired system speed.
- `#define BOOT_CPU_BYTE_ORDER_MOTOROLA (0)`
Motorola or Intel style byte ordering.
- `#define BOOT_CPU_USER_PROGRAM_START_HOOK (1)`
Enable/disable hook function call right before user program start.
- `#define BOOT_COM_USB_ENABLE (1)`
Enable/disable USB transport layer.
- `#define BOOT_COM_USB_TX_MAX_DATA (63)`
Configure number of bytes in the target->host data packet.
- `#define BOOT_COM_USB_RX_MAX_DATA (63)`
Configure number of bytes in the host->target data packet.
- `#define BOOT_COM_CAN_ENABLE (1)`
Enable/disable CAN transport layer.
- `#define BOOT_COM_CAN_BAUDRATE (500000)`
Configure the desired CAN baudrate.
- `#define BOOT_COM_CAN_TX_MSG_ID (0x7E1 /*| 0x80000000*/)`
Configure CAN message ID target->host.
- `#define BOOT_COM_CAN_TX_MAX_DATA (8)`
Configure number of bytes in the target->host CAN message.
- `#define BOOT_COM_CAN_RX_MSG_ID (0x667 /*| 0x80000000*/)`
Configure CAN message ID host->target.
- `#define BOOT_COM_CAN_RX_MAX_DATA (8)`

- *Configure number of bytes in the host->target CAN message.*
 • #define `BOOT_COM_CAN_CHANNEL_INDEX` (0)
Select the desired CAN peripheral as a zero based index.
- #define `BOOT_COM_RS232_ENABLE` (1)
Enable/disable UART transport layer.
- #define `BOOT_COM_RS232_BAUDRATE` (57600)
Configure the desired communication speed.
- #define `BOOT_COM_RS232_TX_MAX_DATA` (64)
Configure number of bytes in the target->host data packet.
- #define `BOOT_COM_RS232_RX_MAX_DATA` (64)
Configure number of bytes in the host->target data packet.
- #define `BOOT_COM_RS232_CHANNEL_INDEX` (5)
Select the desired UART peripheral as a zero based index. Note that the LPUART peripherals come after the U(↔S)ART peripherals.
- #define `BOOT_BACKDOOR_HOOKS_ENABLE` (0)
Enable/disable the backdoor override hook functions.
- #define `BOOT_NVM_HOOKS_ENABLE` (0)
Enable/disable the NVM hook function for supporting additional memory devices.
- #define `BOOT_NVM_SIZE_KB` (512)
Configure the size of the default memory device (typically flash EEPROM).
- #define `BOOT_NVM_CHECKSUM_HOOKS_ENABLE` (0)
Enable/disable hooks functions to override the user program checksum handling.
- #define `BOOT_COP_HOOKS_ENABLE` (1)
Enable/disable the hook functions for controlling the watchdog.

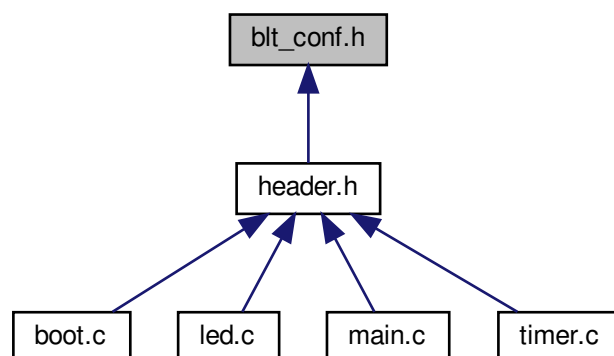
7.88.1 Detailed Description

Bootloader configuration header file.

7.89 blt_conf.h File Reference

Bootloader configuration header file.

This graph shows which files directly or indirectly include this file:



Macros

- `#define BOOT_CPU_XTAL_SPEED_KHZ (16000)`
Frequency of the external crystal or internal (HSI/MSI) oscillator.
- `#define BOOT_CPU_SYSTEM_SPEED_KHZ (96000)`
Desired system speed.
- `#define BOOT_CPU_BYTE_ORDER_MOTOROLA (0)`
Motorola or Intel style byte ordering.
- `#define BOOT_CPU_USER_PROGRAM_START_HOOK (1)`
Enable/disable hook function call right before user program start.
- `#define BOOT_COM_USB_ENABLE (1)`
Enable/disable USB transport layer.
- `#define BOOT_COM_USB_TX_MAX_DATA (63)`
Configure number of bytes in the target->host data packet.
- `#define BOOT_COM_USB_RX_MAX_DATA (63)`
Configure number of bytes in the host->target data packet.
- `#define BOOT_COM_CAN_ENABLE (1)`
Enable/disable CAN transport layer.
- `#define BOOT_COM_CAN_BAUDRATE (500000)`
Configure the desired CAN baudrate.
- `#define BOOT_COM_CAN_TX_MSG_ID (0x7E1 /*| 0x80000000*/)`
Configure CAN message ID target->host.
- `#define BOOT_COM_CAN_TX_MAX_DATA (8)`
Configure number of bytes in the target->host CAN message.
- `#define BOOT_COM_CAN_RX_MSG_ID (0x667 /*| 0x80000000*/)`
Configure CAN message ID host->target.
- `#define BOOT_COM_CAN_RX_MAX_DATA (8)`
Configure number of bytes in the host->target CAN message.
- `#define BOOT_COM_CAN_CHANNEL_INDEX (0)`
Select the desired CAN peripheral as a zero based index.
- `#define BOOT_COM_RS232_ENABLE (1)`
Enable/disable UART transport layer.
- `#define BOOT_COM_RS232_BAUDRATE (57600)`
Configure the desired communication speed.
- `#define BOOT_COM_RS232_TX_MAX_DATA (64)`
Configure number of bytes in the target->host data packet.
- `#define BOOT_COM_RS232_RX_MAX_DATA (64)`
Configure number of bytes in the host->target data packet.
- `#define BOOT_COM_RS232_CHANNEL_INDEX (5)`
Select the desired UART peripheral as a zero based index. Note that the LPUART peripherals come after the U(←S)ART peripherals.
- `#define BOOT_BACKDOOR_HOOKS_ENABLE (0)`
Enable/disable the backdoor override hook functions.
- `#define BOOT_NVM_HOOKS_ENABLE (0)`
Enable/disable the NVM hook function for supporting additional memory devices.
- `#define BOOT_NVM_SIZE_KB (512)`
Configure the size of the default memory device (typically flash EEPROM).
- `#define BOOT_NVM_CHECKSUM_HOOKS_ENABLE (0)`
Enable/disable hooks functions to override the user program checksum handling.
- `#define BOOT_COP_HOOKS_ENABLE (1)`
Enable/disable the hook functions for controlling the watchdog.

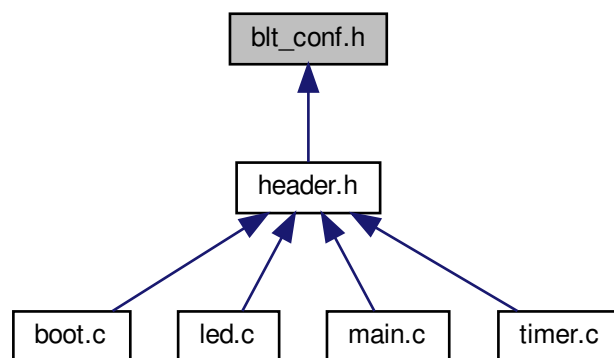
7.89.1 Detailed Description

Bootloader configuration header file.

7.90 blt_conf.h File Reference

Bootloader configuration header file.

This graph shows which files directly or indirectly include this file:



Macros

- `#define BOOT_CPU_XTAL_SPEED_KHZ (16000)`
Frequency of the external crystal or internal (HSI/MSI) oscillator.
- `#define BOOT_CPU_SYSTEM_SPEED_KHZ (96000)`
Desired system speed.
- `#define BOOT_CPU_BYTE_ORDER_MOTOROLA (0)`
Motorola or Intel style byte ordering.
- `#define BOOT_CPU_USER_PROGRAM_START_HOOK (1)`
Enable/disable hook function call right before user program start.
- `#define BOOT_COM_USB_ENABLE (1)`
Enable/disable USB transport layer.
- `#define BOOT_COM_USB_TX_MAX_DATA (63)`
Configure number of bytes in the target->host data packet.
- `#define BOOT_COM_USB_RX_MAX_DATA (63)`
Configure number of bytes in the host->target data packet.
- `#define BOOT_COM_CAN_ENABLE (1)`
Enable/disable CAN transport layer.
- `#define BOOT_COM_CAN_BAUDRATE (500000)`
Configure the desired CAN baudrate.
- `#define BOOT_COM_CAN_TX_MSG_ID (0x7E1 /*| 0x80000000*/)`

- Configure CAN message ID target->host.*

 - #define `BOOT_COM_CAN_TX_MAX_DATA` (8)

Configure number of bytes in the target->host CAN message.
- #define `BOOT_COM_CAN_RX_MSG_ID` (0x667 /*| 0x80000000*/)

Configure CAN message ID host->target.
- #define `BOOT_COM_CAN_RX_MAX_DATA` (8)

Configure number of bytes in the host->target CAN message.
- #define `BOOT_COM_CAN_CHANNEL_INDEX` (0)

Select the desired CAN peripheral as a zero based index.
- #define `BOOT_COM_RS232_ENABLE` (1)

Enable/disable UART transport layer.
- #define `BOOT_COM_RS232_BAUDRATE` (57600)

Configure the desired communication speed.
- #define `BOOT_COM_RS232_TX_MAX_DATA` (64)

Configure number of bytes in the target->host data packet.
- #define `BOOT_COM_RS232_RX_MAX_DATA` (64)

Configure number of bytes in the host->target data packet.
- #define `BOOT_COM_RS232_CHANNEL_INDEX` (5)

Select the desired UART peripheral as a zero based index. Note that the LPUART peripherals come after the U(↔S)ART peripherals.
- #define `BOOT_BACKDOOR_HOOKS_ENABLE` (0)

Enable/disable the backdoor override hook functions.
- #define `BOOT_NVM_HOOKS_ENABLE` (0)

Enable/disable the NVM hook function for supporting additional memory devices.
- #define `BOOT_NVM_SIZE_KB` (512)

Configure the size of the default memory device (typically flash EEPROM).
- #define `BOOT_NVM_CHECKSUM_HOOKS_ENABLE` (0)

Enable/disable hooks functions to override the user program checksum handling.
- #define `BOOT_COP_HOOKS_ENABLE` (1)

Enable/disable the hook functions for controlling the watchdog.

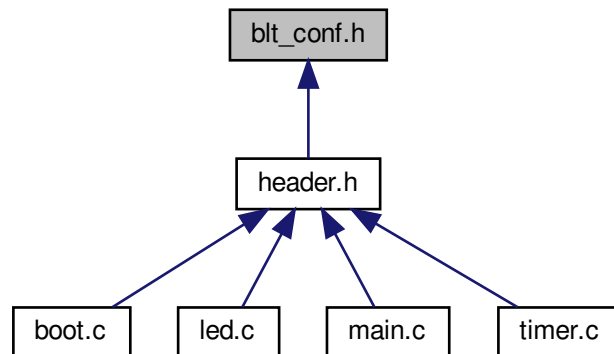
7.90.1 Detailed Description

Bootloader configuration header file.

7.91 blt_conf.h File Reference

Bootloader configuration header file.

This graph shows which files directly or indirectly include this file:



Macros

- `#define BOOT_CPU_XTAL_SPEED_KHZ (16000)`
Frequency of the external crystal or internal (HSI/MSI) oscillator.
- `#define BOOT_CPU_SYSTEM_SPEED_KHZ (96000)`
Desired system speed.
- `#define BOOT_CPU_BYTE_ORDER_MOTOROLA (0)`
Motorola or Intel style byte ordering.
- `#define BOOT_CPU_USER_PROGRAM_START_HOOK (1)`
Enable/disable hook function call right before user program start.
- `#define BOOT_COM_USB_ENABLE (1)`
Enable/disable USB transport layer.
- `#define BOOT_COM_USB_TX_MAX_DATA (63)`
Configure number of bytes in the target->host data packet.
- `#define BOOT_COM_USB_RX_MAX_DATA (63)`
Configure number of bytes in the host->target data packet.
- `#define BOOT_COM_CAN_ENABLE (1)`
Enable/disable CAN transport layer.
- `#define BOOT_COM_CAN_BAUDRATE (500000)`
Configure the desired CAN baudrate.
- `#define BOOT_COM_CAN_TX_MSG_ID (0x7E1 /*| 0x80000000*/)`
Configure CAN message ID target->host.
- `#define BOOT_COM_CAN_TX_MAX_DATA (8)`
Configure number of bytes in the target->host CAN message.
- `#define BOOT_COM_CAN_RX_MSG_ID (0x667 /*| 0x80000000*/)`
Configure CAN message ID host->target.
- `#define BOOT_COM_CAN_RX_MAX_DATA (8)`
Configure number of bytes in the host->target CAN message.
- `#define BOOT_COM_CAN_CHANNEL_INDEX (0)`
Select the desired CAN peripheral as a zero based index.
- `#define BOOT_COM_RS232_ENABLE (1)`

- *Enable/disable UART transport layer.*
- `#define BOOT_COM_RS232_BAUDRATE (57600)`
Configure the desired communication speed.
- `#define BOOT_COM_RS232_TX_MAX_DATA (64)`
Configure number of bytes in the target->host data packet.
- `#define BOOT_COM_RS232_RX_MAX_DATA (64)`
Configure number of bytes in the host->target data packet.
- `#define BOOT_COM_RS232_CHANNEL_INDEX (5)`
Select the desired UART peripheral as a zero based index. Note that the LPUART peripherals come after the U(←S)ART peripherals.
- `#define BOOT_BACKDOOR_HOOKS_ENABLE (0)`
Enable/disable the backdoor override hook functions.
- `#define BOOT_NVM_HOOKS_ENABLE (0)`
Enable/disable the NVM hook function for supporting additional memory devices.
- `#define BOOT_NVM_SIZE_KB (512)`
Configure the size of the default memory device (typically flash EEPROM).
- `#define BOOT_NVM_CHECKSUM_HOOKS_ENABLE (0)`
Enable/disable hooks functions to override the user program checksum handling.
- `#define BOOT_COP_HOOKS_ENABLE (1)`
Enable/disable the hook functions for controlling the watchdog.

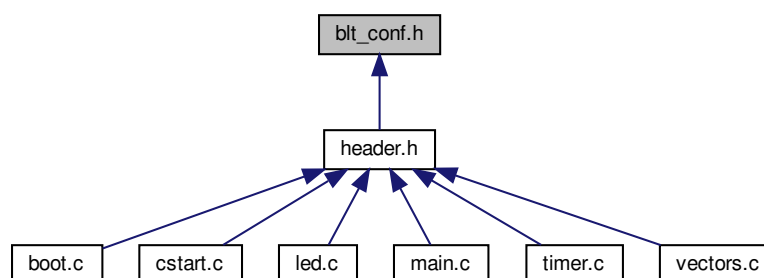
7.91.1 Detailed Description

Bootloader configuration header file.

7.92 blt_conf.h File Reference

Bootloader configuration header file.

This graph shows which files directly or indirectly include this file:



Macros

- #define `BOOT_CPU_XTAL_SPEED_KHZ` (32000)
Frequency of the external crystal oscillator.
- #define `BOOT_CPU_SYSTEM_SPEED_KHZ` (14000)
Desired system speed.
- #define `BOOT_CPU_BYTE_ORDER_MOTOROLA` (0)
Motorola or Intel style byte ordering.
- #define `BOOT_CPU_USER_PROGRAM_START_HOOK` (0)
Enable/disable hook function call right before user program start.
- #define `BOOT_COM_RS232_ENABLE` (1)
Enable/disable UART transport layer.
- #define `BOOT_COM_RS232_BAUDRATE` (9600)
Configure the desired communication speed.
- #define `BOOT_COM_RS232_TX_MAX_DATA` (64)
Configure number of bytes in the target->host data packet.
- #define `BOOT_COM_RS232_RX_MAX_DATA` (64)
Configure number of bytes in the host->target data packet.
- #define `BOOT_COM_RS232_CHANNEL_INDEX` (1)
Select the desired UART peripheral as a zero based index.
- #define `BOOT_BACKDOOR_HOOKS_ENABLE` (0)
Enable/disable the backdoor override hook functions.
- #define `BOOT_NVM_HOOKS_ENABLE` (0)
Enable/disable the NVM hook function for supporting additional memory devices.
- #define `BOOT_NVM_SIZE_KB` (128)
Configure the size of the default memory device (typically flash EEPROM).
- #define `BOOT_NVM_CHECKSUM_HOOKS_ENABLE` (0)
Enable/disable hooks functions to override the user program checksum handling.
- #define `BOOT_COP_HOOKS_ENABLE` (0)
Enable/disable the hook functions for controlling the watchdog.

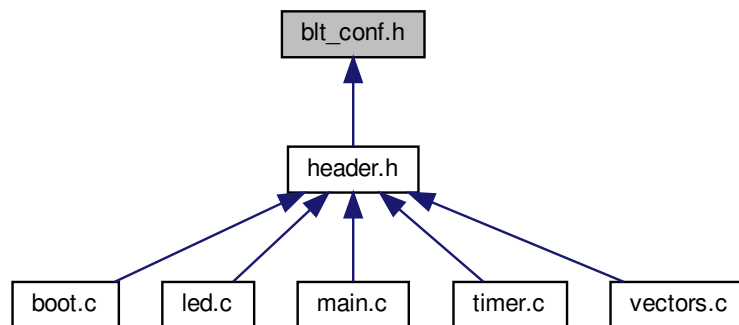
7.92.1 Detailed Description

Bootloader configuration header file.

7.93 blt_conf.h File Reference

Bootloader configuration header file.

This graph shows which files directly or indirectly include this file:



Macros

- `#define BOOT_CPU_XTAL_SPEED_KHZ (32000)`
Frequency of the external crystal oscillator.
- `#define BOOT_CPU_SYSTEM_SPEED_KHZ (14000)`
Desired system speed.
- `#define BOOT_CPU_BYTE_ORDER_MOTOROLA (0)`
Motorola or Intel style byte ordering.
- `#define BOOT_CPU_USER_PROGRAM_START_HOOK (0)`
Enable/disable hook function call right before user program start.
- `#define BOOT_COM_RS232_ENABLE (1)`
Enable/disable UART transport layer.
- `#define BOOT_COM_RS232_BAUDRATE (9600)`
Configure the desired communication speed.
- `#define BOOT_COM_RS232_TX_MAX_DATA (64)`
Configure number of bytes in the target->host data packet.
- `#define BOOT_COM_RS232_RX_MAX_DATA (64)`
Configure number of bytes in the host->target data packet.
- `#define BOOT_COM_RS232_CHANNEL_INDEX (1)`
Select the desired UART peripheral as a zero based index.
- `#define BOOT_BACKDOOR_HOOKS_ENABLE (0)`
Enable/disable the backdoor override hook functions.
- `#define BOOT_NVM_HOOKS_ENABLE (0)`
Enable/disable the NVM hook function for supporting additional memory devices.
- `#define BOOT_NVM_SIZE_KB (128)`
Configure the size of the default memory device (typically flash EEPROM).
- `#define BOOT_NVM_CHECKSUM_HOOKS_ENABLE (0)`
Enable/disable hooks functions to override the user program checksum handling.
- `#define BOOT_COP_HOOKS_ENABLE (0)`
Enable/disable the hook functions for controlling the watchdog.

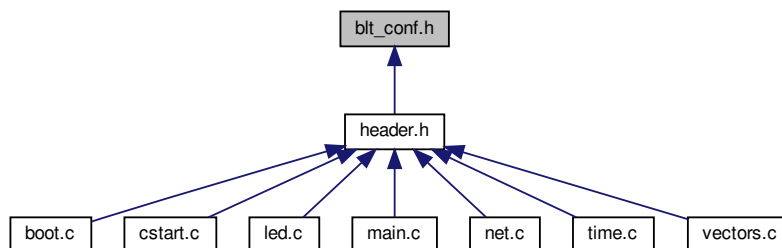
7.93.1 Detailed Description

Bootloader configuration header file.

7.94 blt_conf.h File Reference

Bootloader configuration header file.

This graph shows which files directly or indirectly include this file:



Macros

- `#define BOOT_CPU_XTAL_SPEED_KHZ (8000)`
Frequency of the external crystal oscillator.
- `#define BOOT_CPU_SYSTEM_SPEED_KHZ (50000)`
Desired system speed.
- `#define BOOT_CPU_BYTE_ORDER_MOTOROLA (0)`
Motorola or Intel style byte ordering.
- `#define BOOT_CPU_USER_PROGRAM_START_HOOK (1)`
Enable/disable hook function call right before user program start.
- `#define BOOT_COM_RS232_ENABLE (1)`
Enable/disable UART transport layer.
- `#define BOOT_COM_RS232_BAUDRATE (57600)`
Configure the desired communication speed.
- `#define BOOT_COM_RS232_TX_MAX_DATA (64)`
Configure number of bytes in the target->host data packet.
- `#define BOOT_COM_RS232_RX_MAX_DATA (64)`
Configure number of bytes in the host->target data packet.
- `#define BOOT_COM_RS232_CHANNEL_INDEX (0)`
Select the desired UART peripheral as a zero based index.
- `#define BOOT_COM_NET_ENABLE (1)`
Enable/disable the NET transport layer.
- `#define BOOT_COM_NET_TX_MAX_DATA (64)`
Configure number of bytes in the target->host data packet.
- `#define BOOT_COM_NET_RX_MAX_DATA (64)`
Configure number of bytes in the host->target data packet.

- `#define BOOT_COM_NET_PORT (1000)`
Configure the port that the TCP/IP server listens on.
- `#define BOOT_COM_NET_DHCP_ENABLE (1)`
Enable/disable DHCP client for automatically obtaining an IP address.
- `#define BOOT_COM_NET_IPADDR0 (192)`
Configure the 1st byte of the IP address.
- `#define BOOT_COM_NET_IPADDR1 (168)`
Configure the 2nd byte of the IP address.
- `#define BOOT_COM_NET_IPADDR2 (178)`
Configure the 3rd byte of the IP address.
- `#define BOOT_COM_NET_IPADDR3 (50)`
Configure the 4th byte of the IP address.
- `#define BOOT_COM_NET_NETMASK0 (255)`
Configure the 1st byte of the network mask.
- `#define BOOT_COM_NET_NETMASK1 (255)`
Configure the 2nd byte of the network mask.
- `#define BOOT_COM_NET_NETMASK2 (255)`
Configure the 3rd byte of the network mask.
- `#define BOOT_COM_NET_NETMASK3 (0)`
Configure the 4th byte of the network mask.
- `#define BOOT_COM_NET_GATEWAY0 (192)`
Configure the 1st byte of the gateway address.
- `#define BOOT_COM_NET_GATEWAY1 (168)`
Configure the 2nd byte of the gateway address.
- `#define BOOT_COM_NET_GATEWAY2 (178)`
Configure the 3rd byte of the gateway address.
- `#define BOOT_COM_NET_GATEWAY3 (1)`
Configure the 4th byte of the gateway address.
- `#define BOOT_COM_NET_DEFERRED_INIT_ENABLE (1)`
Enable/disable the deferred initialization mechanism. When enabled, the communication interface is only initialized when: (a) no valid user program is detected, or (b) when `CpuUserProgramStartHook()` returns `BLT_FALSE`. Your bootloader application can explicitly initialize the communication interface by calling `ComDeferredInit()`.
- `#define BOOT_FILE_SYS_ENABLE (1)`
Enable/disable support for firmware updates from a locally attached storage.
- `#define BOOT_FILE_LOGGING_ENABLE (1)`
Enable/disable logging messages during firmware updates.
- `#define BOOT_FILE_ERROR_HOOK_ENABLE (1)`
Enable/disable a hook function that is called upon detection of an error.
- `#define BOOT_FILE_STARTED_HOOK_ENABLE (1)`
Enable/disable a hook function that is called at the start of the update.
- `#define BOOT_FILE_COMPLETED_HOOK_ENABLE (1)`
Enable/disable a hook function that is called at the end of the update.
- `#define BOOT_BACKDOOR_HOOKS_ENABLE (0)`
Enable/disable the backdoor override hook functions.
- `#define BOOT_NVM_HOOKS_ENABLE (0)`
Enable/disable the NVM hook function for supporting additional memory devices.
- `#define BOOT_NVM_SIZE_KB (256)`
Configure the size of the default memory device (typically flash EEPROM).
- `#define BOOT_NVM_CHECKSUM_HOOKS_ENABLE (0)`
Enable/disable hooks functions to override the user program checksum handling.
- `#define BOOT_COP_HOOKS_ENABLE (0)`
Enable/disable the hook functions for controlling the watchdog.

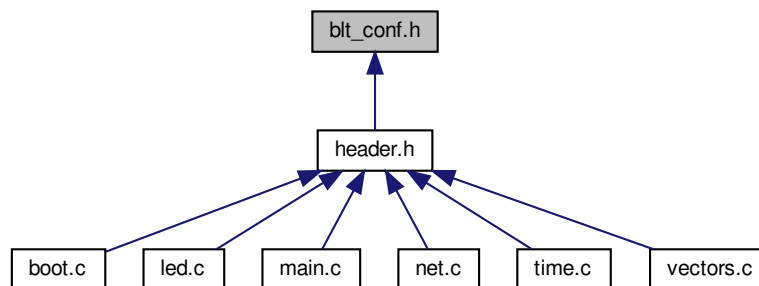
7.94.1 Detailed Description

Bootloader configuration header file.

7.95 blt_conf.h File Reference

Bootloader configuration header file.

This graph shows which files directly or indirectly include this file:



Macros

- `#define BOOT_CPU_XTAL_SPEED_KHZ (8000)`
Frequency of the external crystal oscillator.
- `#define BOOT_CPU_SYSTEM_SPEED_KHZ (50000)`
Desired system speed.
- `#define BOOT_CPU_BYTE_ORDER_MOTOROLA (0)`
Motorola or Intel style byte ordering.
- `#define BOOT_CPU_USER_PROGRAM_START_HOOK (1)`
Enable/disable hook function call right before user program start.
- `#define BOOT_COM_RS232_ENABLE (1)`
Enable/disable UART transport layer.
- `#define BOOT_COM_RS232_BAUDRATE (57600)`
Configure the desired communication speed.
- `#define BOOT_COM_RS232_TX_MAX_DATA (64)`
Configure number of bytes in the target->host data packet.
- `#define BOOT_COM_RS232_RX_MAX_DATA (64)`
Configure number of bytes in the host->target data packet.
- `#define BOOT_COM_RS232_CHANNEL_INDEX (0)`
Select the desired UART peripheral as a zero based index.
- `#define BOOT_COM_NET_ENABLE (1)`
Enable/disable the NET transport layer.
- `#define BOOT_COM_NET_TX_MAX_DATA (64)`
Configure number of bytes in the target->host data packet.

- #define [BOOT_COM_NET_RX_MAX_DATA](#) (64)
Configure number of bytes in the host->target data packet.
- #define [BOOT_COM_NET_PORT](#) (1000)
Configure the port that the TCP/IP server listens on.
- #define [BOOT_COM_NET_DHCP_ENABLE](#) (1)
Enable/disable DHCP client for automatically obtaining an IP address.
- #define [BOOT_COM_NET_IPADDR0](#) (192)
Configure the 1st byte of the IP address.
- #define [BOOT_COM_NET_IPADDR1](#) (168)
Configure the 2nd byte of the IP address.
- #define [BOOT_COM_NET_IPADDR2](#) (178)
Configure the 3rd byte of the IP address.
- #define [BOOT_COM_NET_IPADDR3](#) (50)
Configure the 4th byte of the IP address.
- #define [BOOT_COM_NET_NETMASK0](#) (255)
Configure the 1st byte of the network mask.
- #define [BOOT_COM_NET_NETMASK1](#) (255)
Configure the 2nd byte of the network mask.
- #define [BOOT_COM_NET_NETMASK2](#) (255)
Configure the 3rd byte of the network mask.
- #define [BOOT_COM_NET_NETMASK3](#) (0)
Configure the 4th byte of the network mask.
- #define [BOOT_COM_NET_GATEWAY0](#) (192)
Configure the 1st byte of the gateway address.
- #define [BOOT_COM_NET_GATEWAY1](#) (168)
Configure the 2nd byte of the gateway address.
- #define [BOOT_COM_NET_GATEWAY2](#) (178)
Configure the 3rd byte of the gateway address.
- #define [BOOT_COM_NET_GATEWAY3](#) (1)
Configure the 4th byte of the gateway address.
- #define [BOOT_COM_NET_DEFERRED_INIT_ENABLE](#) (1)
Enable/disable the deferred initialization mechanism. When enabled, the communication interface is only initialized when: (a) no valid user program is detected, or (b) when [CpuUserProgramStartHook\(\)](#) returns [BLT_FALSE](#). Your bootloader application can explicitly initialize the communication interface by calling [ComDeferredInit\(\)](#).
- #define [BOOT_FILE_SYS_ENABLE](#) (1)
Enable/disable support for firmware updates from a locally attached storage.
- #define [BOOT_FILE_LOGGING_ENABLE](#) (1)
Enable/disable logging messages during firmware updates.
- #define [BOOT_FILE_ERROR_HOOK_ENABLE](#) (1)
Enable/disable a hook function that is called upon detection of an error.
- #define [BOOT_FILE_STARTED_HOOK_ENABLE](#) (1)
Enable/disable a hook function that is called at the start of the update.
- #define [BOOT_FILE_COMPLETED_HOOK_ENABLE](#) (1)
Enable/disable a hook function that is called at the end of the update.
- #define [BOOT_BACKDOOR_HOOKS_ENABLE](#) (0)
Enable/disable the backdoor override hook functions.
- #define [BOOT_NVM_HOOKS_ENABLE](#) (0)
Enable/disable the NVM hook function for supporting additional memory devices.
- #define [BOOT_NVM_SIZE_KB](#) (256)
Configure the size of the default memory device (typically flash EEPROM).
- #define [BOOT_NVM_CHECKSUM_HOOKS_ENABLE](#) (0)
Enable/disable hooks functions to override the user program checksum handling.
- #define [BOOT_COP_HOOKS_ENABLE](#) (0)
Enable/disable the hook functions for controlling the watchdog.

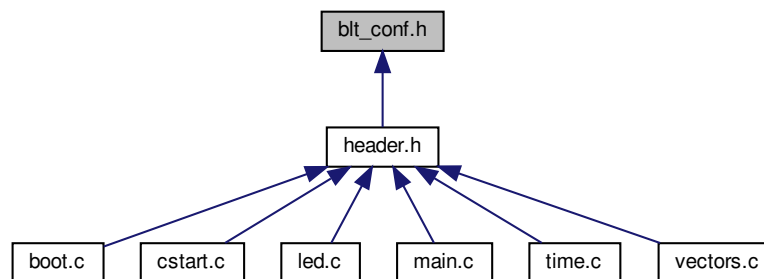
7.95.1 Detailed Description

Bootloader configuration header file.

7.96 blt_conf.h File Reference

Bootloader configuration header file.

This graph shows which files directly or indirectly include this file:



Macros

- `#define BOOT_CPU_XTAL_SPEED_KHZ (8000)`
Frequency of the external crystal oscillator.
- `#define BOOT_CPU_SYSTEM_SPEED_KHZ (50000)`
Desired system speed.
- `#define BOOT_CPU_BYTE_ORDER_MOTOROLA (0)`
Motorola or Intel style byte ordering.
- `#define BOOT_CPU_USER_PROGRAM_START_HOOK (0)`
Enable/disable hook function call right before user program start.
- `#define BOOT_COM_CAN_ENABLE (1)`
Enable/disable CAN transport layer.
- `#define BOOT_COM_CAN_BAUDRATE (500000)`
Configure the desired CAN baudrate.
- `#define BOOT_COM_CAN_TX_MSG_ID (0x7E1 /*| 0x80000000*/)`
Configure CAN message ID target->host.
- `#define BOOT_COM_CAN_TX_MAX_DATA (8)`
Configure number of bytes in the target->host CAN message.
- `#define BOOT_COM_CAN_RX_MSG_ID (0x667 /*| 0x80000000*/)`
Configure CAN message ID host->target.
- `#define BOOT_COM_CAN_RX_MAX_DATA (8)`
Configure number of bytes in the host->target CAN message.
- `#define BOOT_COM_CAN_CHANNEL_INDEX (0)`
Select the desired CAN peripheral as a zero based index.
- `#define BOOT_COM_RS232_ENABLE (1)`

- *Enable/disable UART transport layer.*
- `#define BOOT_COM_RS232_BAUDRATE (57600)`
Configure the desired communication speed.
- `#define BOOT_COM_RS232_TX_MAX_DATA (64)`
Configure number of bytes in the target->host data packet.
- `#define BOOT_COM_RS232_RX_MAX_DATA (64)`
Configure number of bytes in the host->target data packet.
- `#define BOOT_COM_RS232_CHANNEL_INDEX (0)`
Select the desired UART peripheral as a zero based index.
- `#define BOOT_BACKDOOR_HOOKS_ENABLE (0)`
Enable/disable the backdoor override hook functions.
- `#define BOOT_NVM_HOOKS_ENABLE (0)`
Enable/disable the NVM hook function for supporting additional memory devices.
- `#define BOOT_NVM_SIZE_KB (256)`
Configure the size of the default memory device (typically flash EEPROM).
- `#define BOOT_NVM_CHECKSUM_HOOKS_ENABLE (0)`
Enable/disable hooks functions to override the user program checksum handling.
- `#define BOOT_FLASH_CUSTOM_LAYOUT_ENABLE (1)`
Enable support for a custom flash layout table. It is located in flash_layout.c. This was done because the default flashLayout[] table in the bootloader's core has more flash memory reserved for the bootloader than is needed for this demo.
- `#define BOOT_COP_HOOKS_ENABLE (0)`
Enable/disable the hook functions for controlling the watchdog.

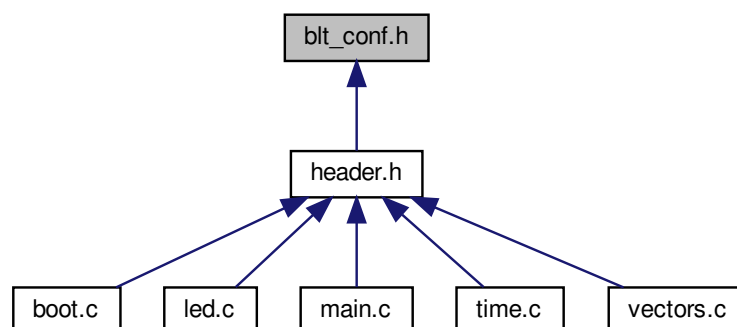
7.96.1 Detailed Description

Bootloader configuration header file.

7.97 blt_conf.h File Reference

Bootloader configuration header file.

This graph shows which files directly or indirectly include this file:



Macros

- #define `BOOT_CPU_XTAL_SPEED_KHZ` (8000)
Frequency of the external crystal oscillator.
- #define `BOOT_CPU_SYSTEM_SPEED_KHZ` (50000)
Desired system speed.
- #define `BOOT_CPU_BYTE_ORDER_MOTOROLA` (0)
Motorola or Intel style byte ordering.
- #define `BOOT_CPU_USER_PROGRAM_START_HOOK` (0)
Enable/disable hook function call right before user program start.
- #define `BOOT_COM_CAN_ENABLE` (1)
Enable/disable CAN transport layer.
- #define `BOOT_COM_CAN_BAUDRATE` (500000)
Configure the desired CAN baudrate.
- #define `BOOT_COM_CAN_TX_MSG_ID` (0x7E1 /*| 0x80000000*/) *Configure CAN message ID target->host.*
- #define `BOOT_COM_CAN_TX_MAX_DATA` (8)
Configure number of bytes in the target->host CAN message.
- #define `BOOT_COM_CAN_RX_MSG_ID` (0x667 /*| 0x80000000*/) *Configure CAN message ID host->target.*
- #define `BOOT_COM_CAN_RX_MAX_DATA` (8)
Configure number of bytes in the host->target CAN message.
- #define `BOOT_COM_CAN_CHANNEL_INDEX` (0)
Select the desired CAN peripheral as a zero based index.
- #define `BOOT_COM_RS232_ENABLE` (1)
Enable/disable UART transport layer.
- #define `BOOT_COM_RS232_BAUDRATE` (57600)
Configure the desired communication speed.
- #define `BOOT_COM_RS232_TX_MAX_DATA` (64)
Configure number of bytes in the target->host data packet.
- #define `BOOT_COM_RS232_RX_MAX_DATA` (64)
Configure number of bytes in the host->target data packet.
- #define `BOOT_COM_RS232_CHANNEL_INDEX` (0)
Select the desired UART peripheral as a zero based index.
- #define `BOOT_BACKDOOR_HOOKS_ENABLE` (0)
Enable/disable the backdoor override hook functions.
- #define `BOOT_NVM_HOOKS_ENABLE` (0)
Enable/disable the NVM hook function for supporting additional memory devices.
- #define `BOOT_NVM_SIZE_KB` (256)
Configure the size of the default memory device (typically flash EEPROM).
- #define `BOOT_NVM_CHECKSUM_HOOKS_ENABLE` (0)
Enable/disable hooks functions to override the user program checksum handling.
- #define `BOOT_FLASH_CUSTOM_LAYOUT_ENABLE` (1)
Enable support for a custom flash layout table. It is located in flash_layout.c. This was done because the default flashLayout[] table in the bootloader's core has more flash memory reserved for the bootloader than is needed for this demo.
- #define `BOOT_COP_HOOKS_ENABLE` (0)
Enable/disable the hook functions for controlling the watchdog.

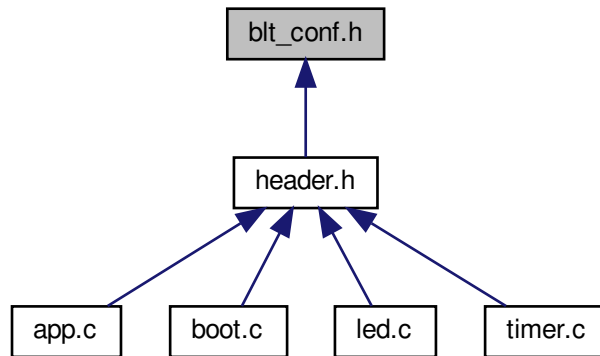
7.97.1 Detailed Description

Bootloader configuration header file.

7.98 blt_conf.h File Reference

Bootloader configuration header file.

This graph shows which files directly or indirectly include this file:



Macros

- `#define BOOT_CPU_XTAL_SPEED_KHZ (8000)`
Frequency of the external crystal oscillator.
- `#define BOOT_CPU_SYSTEM_SPEED_KHZ (72000)`
Desired system speed.
- `#define BOOT_CPU_BYTE_ORDER_MOTOROLA (0)`
Motorola or Intel style byte ordering.
- `#define BOOT_CPU_USER_PROGRAM_START_HOOK (1)`
Enable/disable hook function call right before user program start.
- `#define BOOT_COM_RS232_ENABLE (1)`
Enable/disable UART transport layer.
- `#define BOOT_COM_RS232_BAUDRATE (57600)`
Configure the desired communication speed.
- `#define BOOT_COM_RS232_TX_MAX_DATA (64)`
Configure number of bytes in the target->host data packet.
- `#define BOOT_COM_RS232_RX_MAX_DATA (64)`
Configure number of bytes in the host->target data packet.
- `#define BOOT_COM_RS232_CHANNEL_INDEX (1)`
Select the desired UART peripheral as a zero based index.
- `#define BOOT_BACKDOOR_HOOKS_ENABLE (0)`
Enable/disable the backdoor override hook functions.
- `#define BOOT_NVM_HOOKS_ENABLE (0)`
Enable/disable the NVM hook function for supporting additional memory devices.
- `#define BOOT_NVM_SIZE_KB (128)`
Configure the size of the default memory device (typically flash EEPROM).
- `#define BOOT_NVM_CHECKSUM_HOOKS_ENABLE (0)`

Enable/disable hooks functions to override the user program checksum handling.

- #define `BOOT_FLASH_VECTOR_TABLE_CS_OFFSET` (0x10c)
This microcontroller has a smaller vector table then the default STM32F1xx project as assumed in the bootloader's core. This means the user program has a different checksum location, because this one is added at the end of the user program's vector table.
- #define `BOOT_FLASH_CUSTOM_LAYOUT_ENABLE` (1)
Enable support for a custom flash layout table. It is located in `flash_layout.c`. This was done because the default `flashLayout[]` table in the bootloader's core has more flash memory reserved for the bootloader than is needed for this demo.
- #define `BOOT_COP_HOOKS_ENABLE` (1)
Enable/disable the hook functions for controlling the watchdog.

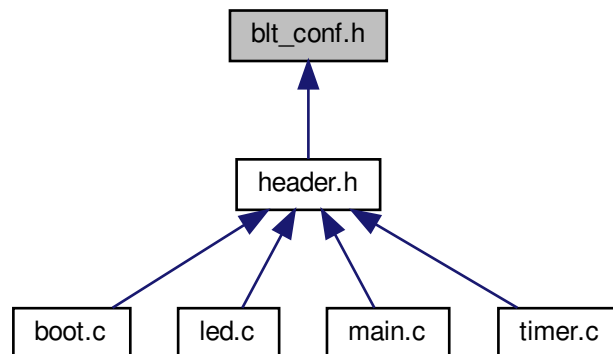
7.98.1 Detailed Description

Bootloader configuration header file.

7.99 blt_conf.h File Reference

Bootloader configuration header file.

This graph shows which files directly or indirectly include this file:



Macros

- #define `BOOT_CPU_XTAL_SPEED_KHZ` (8000)
Frequency of the external crystal oscillator.
- #define `BOOT_CPU_SYSTEM_SPEED_KHZ` (72000)
Desired system speed.
- #define `BOOT_CPU_BYTE_ORDER_MOTOROLA` (0)
Motorola or Intel style byte ordering.
- #define `BOOT_CPU_USER_PROGRAM_START_HOOK` (1)

- Enable/disable hook function call right before user program start.*

 - #define `BOOT_COM_RS232_ENABLE` (1)

Enable/disable UART transport layer.
- #define `BOOT_COM_RS232_BAUDRATE` (57600)

Configure the desired communication speed.
- #define `BOOT_COM_RS232_TX_MAX_DATA` (64)

Configure number of bytes in the target->host data packet.
- #define `BOOT_COM_RS232_RX_MAX_DATA` (64)

Configure number of bytes in the host->target data packet.
- #define `BOOT_COM_RS232_CHANNEL_INDEX` (1)

Select the desired UART peripheral as a zero based index.
- #define `BOOT_BACKDOOR_HOOKS_ENABLE` (0)

Enable/disable the backdoor override hook functions.
- #define `BOOT_NVM_HOOKS_ENABLE` (0)

Enable/disable the NVM hook function for supporting additional memory devices.
- #define `BOOT_NVM_SIZE_KB` (128)

Configure the size of the default memory device (typically flash EEPROM).
- #define `BOOT_NVM_CHECKSUM_HOOKS_ENABLE` (0)

Enable/disable hooks functions to override the user program checksum handling.
- #define `BOOT_FLASH_VECTOR_TABLE_CS_OFFSET` (0x10c)

This microcontroller has a smaller vector table then the default STM32F1xx project as assumed in the bootloader's core. This means the user program has a different checksum location, because this one is added at the end of the user program's vector table.
- #define `BOOT_FLASH_CUSTOM_LAYOUT_ENABLE` (1)

Enable support for a custom flash layout table. It is located in flash_layout.c. This was done because the default flashLayout[] table in the bootloader's core has more flash memory reserved for the bootloader than is needed for this demo.
- #define `BOOT_COP_HOOKS_ENABLE` (1)

Enable/disable the hook functions for controlling the watchdog.

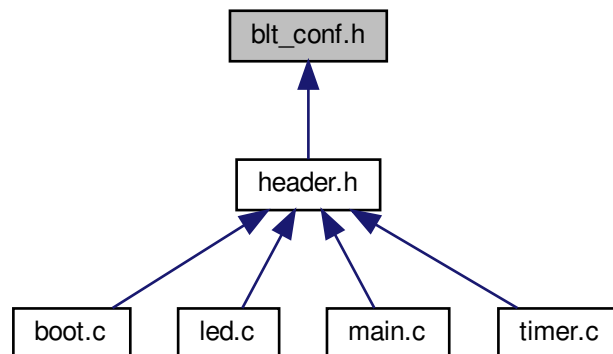
7.99.1 Detailed Description

Bootloader configuration header file.

7.100 blt_conf.h File Reference

Bootloader configuration header file.

This graph shows which files directly or indirectly include this file:



Macros

- `#define BOOT_CPU_XTAL_SPEED_KHZ (8000)`
Frequency of the external crystal oscillator.
- `#define BOOT_CPU_SYSTEM_SPEED_KHZ (72000)`
Desired system speed.
- `#define BOOT_CPU_BYTE_ORDER_MOTOROLA (0)`
Motorola or Intel style byte ordering.
- `#define BOOT_CPU_USER_PROGRAM_START_HOOK (1)`
Enable/disable hook function call right before user program start.
- `#define BOOT_COM_RS232_ENABLE (1)`
Enable/disable UART transport layer.
- `#define BOOT_COM_RS232_BAUDRATE (57600)`
Configure the desired communication speed.
- `#define BOOT_COM_RS232_TX_MAX_DATA (64)`
Configure number of bytes in the target->host data packet.
- `#define BOOT_COM_RS232_RX_MAX_DATA (64)`
Configure number of bytes in the host->target data packet.
- `#define BOOT_COM_RS232_CHANNEL_INDEX (1)`
Select the desired UART peripheral as a zero based index.
- `#define BOOT_BACKDOOR_HOOKS_ENABLE (0)`
Enable/disable the backdoor override hook functions.
- `#define BOOT_NVM_HOOKS_ENABLE (0)`
Enable/disable the NVM hook function for supporting additional memory devices.
- `#define BOOT_NVM_SIZE_KB (128)`
Configure the size of the default memory device (typically flash EEPROM).
- `#define BOOT_NVM_CHECKSUM_HOOKS_ENABLE (0)`
Enable/disable hooks functions to override the user program checksum handling.
- `#define BOOT_FLASH_VECTOR_TABLE_CS_OFFSET (0xec)`
This microcontroller has a smaller vector table then the default STM32F1xx project as assumed in the bootloader's core. This means the user program has a different checksum location, because this one is added at the end of the user program's vector table.

- `#define BOOT_FLASH_CUSTOM_LAYOUT_ENABLE (1)`
Enable support for a custom flash layout table. It is located in `flash_layout.c`. This was done because the default `flashLayout[]` table in the bootloader's core has more flash memory reserved for the bootloader than is needed for this demo.
- `#define BOOT_COP_HOOKS_ENABLE (1)`
Enable/disable the hook functions for controlling the watchdog.

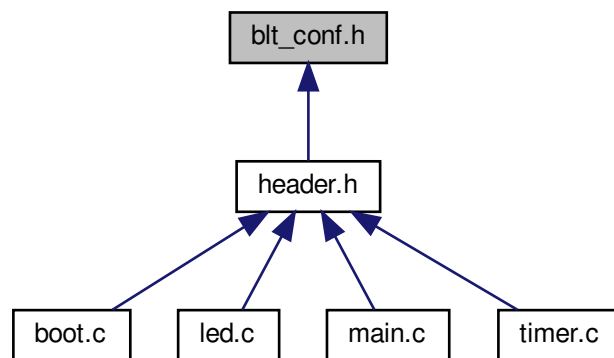
7.100.1 Detailed Description

Bootloader configuration header file.

7.101 blt_conf.h File Reference

Bootloader configuration header file.

This graph shows which files directly or indirectly include this file:



Macros

- `#define BOOT_CPU_XTAL_SPEED_KHZ (8000)`
Frequency of the external crystal oscillator.
- `#define BOOT_CPU_SYSTEM_SPEED_KHZ (72000)`
Desired system speed.
- `#define BOOT_CPU_BYTE_ORDER_MOTOROLA (0)`
Motorola or Intel style byte ordering.
- `#define BOOT_CPU_USER_PROGRAM_START_HOOK (1)`
Enable/disable hook function call right before user program start.
- `#define BOOT_COM_RS232_ENABLE (1)`
Enable/disable UART transport layer.
- `#define BOOT_COM_RS232_BAUDRATE (57600)`
Configure the desired communication speed.

- #define [BOOT_COM_RS232_TX_MAX_DATA](#) (64)
Configure number of bytes in the target->host data packet.
- #define [BOOT_COM_RS232_RX_MAX_DATA](#) (64)
Configure number of bytes in the host->target data packet.
- #define [BOOT_COM_RS232_CHANNEL_INDEX](#) (1)
Select the desired UART peripheral as a zero based index.
- #define [BOOT_BACKDOOR_HOOKS_ENABLE](#) (0)
Enable/disable the backdoor override hook functions.
- #define [BOOT_NVM_HOOKS_ENABLE](#) (0)
Enable/disable the NVM hook function for supporting additional memory devices.
- #define [BOOT_NVM_SIZE_KB](#) (128)
Configure the size of the default memory device (typically flash EEPROM).
- #define [BOOT_NVM_CHECKSUM_HOOKS_ENABLE](#) (0)
Enable/disable hooks functions to override the user program checksum handling.
- #define [BOOT_FLASH_VECTOR_TABLE_CS_OFFSET](#) (0xec)
This microcontroller has a smaller vector table then the default STM32F1xx project as assumed in the bootloader's core. This means the user program has a different checksum location, because this one is added at the end of the user program's vector table.
- #define [BOOT_FLASH_CUSTOM_LAYOUT_ENABLE](#) (1)
Enable support for a custom flash layout table. It is located in flash_layout.c. This was done because the default flashLayout[] table in the bootloader's core has more flash memory reserved for the bootloader than is needed for this demo.
- #define [BOOT_COP_HOOKS_ENABLE](#) (1)
Enable/disable the hook functions for controlling the watchdog.

7.101.1 Detailed Description

Bootloader configuration header file.

7.102 blt_conf.h File Reference

Bootloader configuration header file.

Macros

- #define [BOOT_CPU_XTAL_SPEED_KHZ](#) (8000)
Frequency of the external crystal oscillator.
- #define [BOOT_CPU_SYSTEM_SPEED_KHZ](#) (72000)
Desired system speed.
- #define [BOOT_CPU_BYTE_ORDER_MOTOROLA](#) (0)
Motorola or Intel style byte ordering.
- #define [BOOT_CPU_USER_PROGRAM_START_HOOK](#) (1)
Enable/disable hook function call right before user program start.
- #define [BOOT_COM_USB_ENABLE](#) (1)
Enable/disable USB transport layer.
- #define [BOOT_COM_USB_TX_MAX_DATA](#) (63)
Configure number of bytes in the target->host data packet.
- #define [BOOT_COM_USB_RX_MAX_DATA](#) (63)

- Configure number of bytes in the host->target data packet.*

• #define `BOOT_BACKDOOR_HOOKS_ENABLE` (0)

Enable/disable the backdoor override hook functions.
- #define `BOOT_NVM_HOOKS_ENABLE` (0)

Enable/disable the NVM hook function for supporting additional memory devices.
- #define `BOOT_NVM_SIZE_KB` (128)

Configure the size of the default memory device (typically flash EEPROM).
- #define `BOOT_NVM_CHECKSUM_HOOKS_ENABLE` (0)

Enable/disable hooks functions to override the user program checksum handling.
- #define `BOOT_FLASH_VECTOR_TABLE_CS_OFFSET` (0x10c)

This microcontroller has a smaller vector table then the default STM32F1xx project as assumed in the bootloader's core. This means the user program has a different checksum location, because this one is added at the end of the user program's vector table.
- #define `BOOT_FLASH_CUSTOM_LAYOUT_ENABLE` (1)

Enable support for a custom flash layout table. It is located in flash_layout.c. This was done because the default flashLayout[] table in the bootloader's core has more flash memory reserved for the bootloader than is needed for this demo.
- #define `BOOT_COP_HOOKS_ENABLE` (1)

Enable/disable the hook functions for controlling the watchdog.

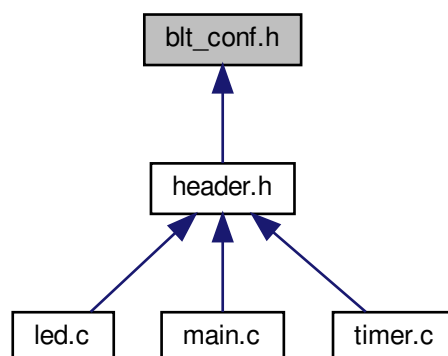
7.102.1 Detailed Description

Bootloader configuration header file.

7.103 blt_conf.h File Reference

Bootloader configuration header file.

This graph shows which files directly or indirectly include this file:



Macros

- #define `BOOT_CPU_XTAL_SPEED_KHZ` (8000)
Frequency of the external crystal oscillator.
- #define `BOOT_CPU_SYSTEM_SPEED_KHZ` (72000)
Desired system speed.
- #define `BOOT_CPU_BYTE_ORDER_MOTOROLA` (0)
Motorola or Intel style byte ordering.
- #define `BOOT_CPU_USER_PROGRAM_START_HOOK` (1)
Enable/disable hook function call right before user program start.
- #define `BOOT_COM_USB_ENABLE` (1)
Enable/disable USB transport layer.
- #define `BOOT_COM_USB_TX_MAX_DATA` (63)
Configure number of bytes in the target->host data packet.
- #define `BOOT_COM_USB_RX_MAX_DATA` (63)
Configure number of bytes in the host->target data packet.
- #define `BOOT_BACKDOOR_HOOKS_ENABLE` (0)
Enable/disable the backdoor override hook functions.
- #define `BOOT_NVM_HOOKS_ENABLE` (0)
Enable/disable the NVM hook function for supporting additional memory devices.
- #define `BOOT_NVM_SIZE_KB` (128)
Configure the size of the default memory device (typically flash EEPROM).
- #define `BOOT_NVM_CHECKSUM_HOOKS_ENABLE` (0)
Enable/disable hooks functions to override the user program checksum handling.
- #define `BOOT_FLASH_VECTOR_TABLE_CS_OFFSET` (0x10c)
This microcontroller has a smaller vector table then the default STM32F1xx project as assumed in the bootloader's core. This means the user program has a different checksum location, because this one is added at the end of the user program's vector table.
- #define `BOOT_FLASH_CUSTOM_LAYOUT_ENABLE` (1)
Enable support for a custom flash layout table. It is located in flash_layout.c. This was done because the default flashLayout[] table in the bootloader's core has more flash memory reserved for the bootloader than is needed for this demo.
- #define `BOOT_COP_HOOKS_ENABLE` (1)
Enable/disable the hook functions for controlling the watchdog.

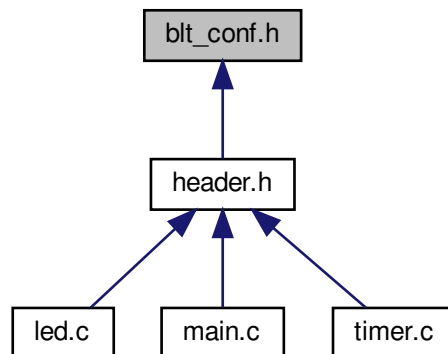
7.103.1 Detailed Description

Bootloader configuration header file.

7.104 blt_conf.h File Reference

Bootloader configuration header file.

This graph shows which files directly or indirectly include this file:



Macros

- `#define BOOT_CPU_XTAL_SPEED_KHZ (8000)`
Frequency of the external crystal oscillator.
- `#define BOOT_CPU_SYSTEM_SPEED_KHZ (72000)`
Desired system speed.
- `#define BOOT_CPU_BYTE_ORDER_MOTOROLA (0)`
Motorola or Intel style byte ordering.
- `#define BOOT_CPU_USER_PROGRAM_START_HOOK (1)`
Enable/disable hook function call right before user program start.
- `#define BOOT_COM_USB_ENABLE (1)`
Enable/disable USB transport layer.
- `#define BOOT_COM_USB_TX_MAX_DATA (63)`
Configure number of bytes in the target->host data packet.
- `#define BOOT_COM_USB_RX_MAX_DATA (63)`
Configure number of bytes in the host->target data packet.
- `#define BOOT_BACKDOOR_HOOKS_ENABLE (0)`
Enable/disable the backdoor override hook functions.
- `#define BOOT_NVM_HOOKS_ENABLE (0)`
Enable/disable the NVM hook function for supporting additional memory devices.
- `#define BOOT_NVM_SIZE_KB (128)`
Configure the size of the default memory device (typically flash EEPROM).
- `#define BOOT_NVM_CHECKSUM_HOOKS_ENABLE (0)`
Enable/disable hooks functions to override the user program checksum handling.
- `#define BOOT_FLASH_VECTOR_TABLE_CS_OFFSET (0xec)`
This microcontroller has a smaller vector table then the default STM32F1xx project as assumed in the bootloader's core. This means the user program has a different checksum location, because this one is added at the end of the user program's vector table.
- `#define BOOT_FLASH_CUSTOM_LAYOUT_ENABLE (1)`
Enable support for a custom flash layout table. It is located in flash_layout.c. This was done because the default flashLayout[] table in the bootloader's core has more flash memory reserved for the bootloader than is needed for this demo.
- `#define BOOT_COP_HOOKS_ENABLE (1)`
Enable/disable the hook functions for controlling the watchdog.

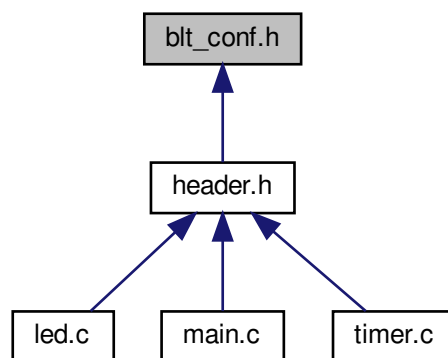
7.104.1 Detailed Description

Bootloader configuration header file.

7.105 blt_conf.h File Reference

Bootloader configuration header file.

This graph shows which files directly or indirectly include this file:



Macros

- #define `BOOT_CPU_XTAL_SPEED_KHZ` (8000)
Frequency of the external crystal oscillator.
- #define `BOOT_CPU_SYSTEM_SPEED_KHZ` (72000)
Desired system speed.
- #define `BOOT_CPU_BYTE_ORDER_MOTOROLA` (0)
Motorola or Intel style byte ordering.
- #define `BOOT_CPU_USER_PROGRAM_START_HOOK` (1)
Enable/disable hook function call right before user program start.
- #define `BOOT_COM_USB_ENABLE` (1)
Enable/disable USB transport layer.
- #define `BOOT_COM_USB_TX_MAX_DATA` (63)
Configure number of bytes in the target->host data packet.
- #define `BOOT_COM_USB_RX_MAX_DATA` (63)
Configure number of bytes in the host->target data packet.
- #define `BOOT_BACKDOOR_HOOKS_ENABLE` (0)
Enable/disable the backdoor override hook functions.
- #define `BOOT_NVM_HOOKS_ENABLE` (0)
Enable/disable the NVM hook function for supporting additional memory devices.
- #define `BOOT_NVM_SIZE_KB` (128)

- `#define BOOT_NVM_CHECKSUM_HOOKS_ENABLE (0)`
Configure the size of the default memory device (typically flash EEPROM).
- `#define BOOT_FLASH_VECTOR_TABLE_CS_OFFSET (0xec)`
Enable/disable hooks functions to override the user program checksum handling.
This microcontroller has a smaller vector table then the default STM32F1xx project as assumed in the bootloader's core. This means the user program has a different checksum location, because this one is added at the end of the user program's vector table.
- `#define BOOT_FLASH_CUSTOM_LAYOUT_ENABLE (1)`
Enable support for a custom flash layout table. It is located in flash_layout.c. This was done because the default flashLayout[] table in the bootloader's core has more flash memory reserved for the bootloader than is needed for this demo.
- `#define BOOT_COP_HOOKS_ENABLE (1)`
Enable/disable the hook functions for controlling the watchdog.

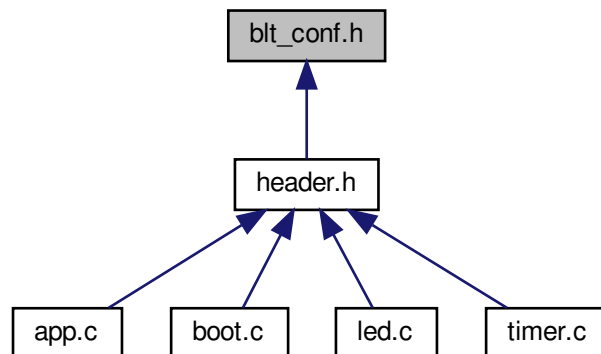
7.105.1 Detailed Description

Bootloader configuration header file.

7.106 blt_conf.h File Reference

Bootloader configuration header file.

This graph shows which files directly or indirectly include this file:



Macros

- `#define BOOT_CPU_XTAL_SPEED_KHZ (8000)`
Frequency of the external crystal oscillator.
- `#define BOOT_CPU_SYSTEM_SPEED_KHZ (72000)`
Desired system speed.
- `#define BOOT_CPU_BYTE_ORDER_MOTOROLA (0)`

- Motorola or Intel style byte ordering.*

 - #define [BOOT_CPU_USER_PROGRAM_START_HOOK](#) (1)
Enable/disable hook function call right before user program start.
 - #define [BOOT_COM_CAN_ENABLE](#) (1)
Enable/disable CAN transport layer.
 - #define [BOOT_COM_CAN_BAUDRATE](#) (500000)
Configure the desired CAN baudrate.
 - #define [BOOT_COM_CAN_TX_MSG_ID](#) (0x7E1 /*| 0x80000000*/) *Configure CAN message ID target->host.*
 - #define [BOOT_COM_CAN_TX_MAX_DATA](#) (8)
Configure number of bytes in the target->host CAN message.
 - #define [BOOT_COM_CAN_RX_MSG_ID](#) (0x667 /*| 0x80000000*/) *Configure CAN message ID host->target.*
 - #define [BOOT_COM_CAN_RX_MAX_DATA](#) (8)
Configure number of bytes in the host->target CAN message.
 - #define [BOOT_COM_CAN_CHANNEL_INDEX](#) (0)
Select the desired CAN peripheral as a zero based index.
 - #define [BOOT_COM_RS232_ENABLE](#) (1)
Enable/disable UART transport layer.
 - #define [BOOT_COM_RS232_BAUDRATE](#) (57600)
Configure the desired communication speed.
 - #define [BOOT_COM_RS232_TX_MAX_DATA](#) (64)
Configure number of bytes in the target->host data packet.
 - #define [BOOT_COM_RS232_RX_MAX_DATA](#) (64)
Configure number of bytes in the host->target data packet.
 - #define [BOOT_COM_RS232_CHANNEL_INDEX](#) (1)
Select the desired UART peripheral as a zero based index.
 - #define [BOOT_FILE_SYS_ENABLE](#) (1)
Enable/disable support for firmware updates from a locally attached storage.
 - #define [BOOT_FILE_LOGGING_ENABLE](#) (1)
Enable/disable logging messages during firmware updates.
 - #define [BOOT_FILE_ERROR_HOOK_ENABLE](#) (1)
Enable/disable a hook function that is called upon detection of an error.
 - #define [BOOT_FILE_STARTED_HOOK_ENABLE](#) (1)
Enable/disable a hook function that is called at the start of the update.
 - #define [BOOT_FILE_COMPLETED_HOOK_ENABLE](#) (1)
Enable/disable a hook function that is called at the end of the update.
 - #define [BOOT_BACKDOOR_HOOKS_ENABLE](#) (0)
Enable/disable the backdoor override hook functions.
 - #define [BOOT_NVM_HOOKS_ENABLE](#) (0)
Enable/disable the NVM hook function for supporting additional memory devices.
 - #define [BOOT_NVM_SIZE_KB](#) (128)
Configure the size of the default memory device (typically flash EEPROM).
 - #define [BOOT_NVM_CHECKSUM_HOOKS_ENABLE](#) (0)
Enable/disable hooks functions to override the user program checksum handling.
 - #define [BOOT_FLASH_VECTOR_TABLE_CS_OFFSET](#) (0x10c)
This microcontroller has a smaller vector table then the default STM32F1xx project as assumed in the bootloader's core. This means the user program has a different checksum location, because this one is added at the end of the user program's vector table.
 - #define [BOOT_COP_HOOKS_ENABLE](#) (1)
Enable/disable the hook functions for controlling the watchdog.

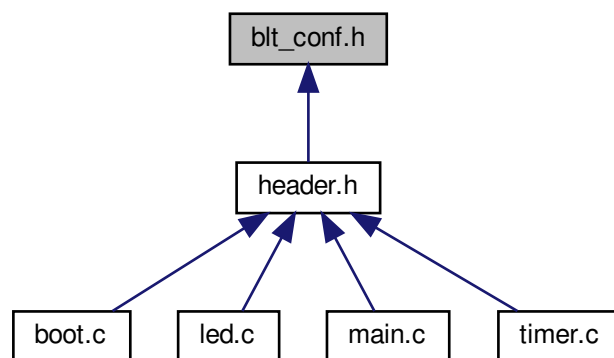
7.106.1 Detailed Description

Bootloader configuration header file.

7.107 blt_conf.h File Reference

Bootloader configuration header file.

This graph shows which files directly or indirectly include this file:



Macros

- `#define BOOT_CPU_XTAL_SPEED_KHZ (8000)`
Frequency of the external crystal oscillator.
- `#define BOOT_CPU_SYSTEM_SPEED_KHZ (72000)`
Desired system speed.
- `#define BOOT_CPU_BYTE_ORDER_MOTOROLA (0)`
Motorola or Intel style byte ordering.
- `#define BOOT_CPU_USER_PROGRAM_START_HOOK (1)`
Enable/disable hook function call right before user program start.
- `#define BOOT_COM_CAN_ENABLE (1)`
Enable/disable CAN transport layer.
- `#define BOOT_COM_CAN_BAUDRATE (500000)`
Configure the desired CAN baudrate.
- `#define BOOT_COM_CAN_TX_MSG_ID (0x7E1 /*| 0x80000000*/)`
Configure CAN message ID target->host.
- `#define BOOT_COM_CAN_TX_MAX_DATA (8)`
Configure number of bytes in the target->host CAN message.
- `#define BOOT_COM_CAN_RX_MSG_ID (0x667 /*| 0x80000000*/)`
Configure CAN message ID host->target.
- `#define BOOT_COM_CAN_RX_MAX_DATA (8)`

- Configure number of bytes in the host->target CAN message.*

 - #define `BOOT_COM_CAN_CHANNEL_INDEX` (0)

Select the desired CAN peripheral as a zero based index.
- #define `BOOT_COM_RS232_ENABLE` (1)

Enable/disable UART transport layer.
- #define `BOOT_COM_RS232_BAUDRATE` (57600)

Configure the desired communication speed.
- #define `BOOT_COM_RS232_TX_MAX_DATA` (64)

Configure number of bytes in the target->host data packet.
- #define `BOOT_COM_RS232_RX_MAX_DATA` (64)

Configure number of bytes in the host->target data packet.
- #define `BOOT_COM_RS232_CHANNEL_INDEX` (1)

Select the desired UART peripheral as a zero based index.
- #define `BOOT_FILE_SYS_ENABLE` (1)

Enable/disable support for firmware updates from a locally attached storage.
- #define `BOOT_FILE_LOGGING_ENABLE` (1)

Enable/disable logging messages during firmware updates.
- #define `BOOT_FILE_ERROR_HOOK_ENABLE` (1)

Enable/disable a hook function that is called upon detection of an error.
- #define `BOOT_FILE_STARTED_HOOK_ENABLE` (1)

Enable/disable a hook function that is called at the start of the update.
- #define `BOOT_FILE_COMPLETED_HOOK_ENABLE` (1)

Enable/disable a hook function that is called at the end of the update.
- #define `BOOT_BACKDOOR_HOOKS_ENABLE` (0)

Enable/disable the backdoor override hook functions.
- #define `BOOT_NVM_HOOKS_ENABLE` (0)

Enable/disable the NVM hook function for supporting additional memory devices.
- #define `BOOT_NVM_SIZE_KB` (128)

Configure the size of the default memory device (typically flash EEPROM).
- #define `BOOT_NVM_CHECKSUM_HOOKS_ENABLE` (0)

Enable/disable hooks functions to override the user program checksum handling.
- #define `BOOT_FLASH_VECTOR_TABLE_CS_OFFSET` (0x10c)

This microcontroller has a smaller vector table then the default STM32F1xx project as assumed in the bootloader's core. This means the user program has a different checksum location, because this one is added at the end of the user program's vector table.
- #define `BOOT_COP_HOOKS_ENABLE` (1)

Enable/disable the hook functions for controlling the watchdog.

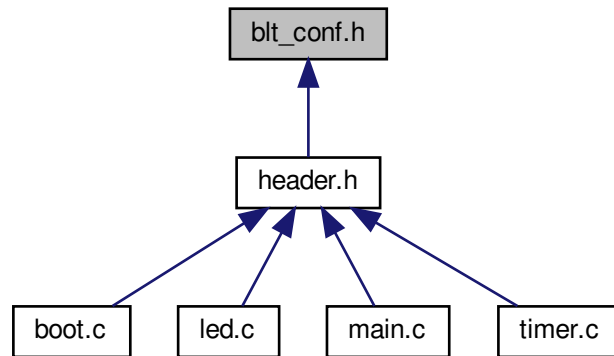
7.107.1 Detailed Description

Bootloader configuration header file.

7.108 blt_conf.h File Reference

Bootloader configuration header file.

This graph shows which files directly or indirectly include this file:



Macros

- `#define BOOT_CPU_XTAL_SPEED_KHZ (8000)`
Frequency of the external crystal oscillator.
- `#define BOOT_CPU_SYSTEM_SPEED_KHZ (72000)`
Desired system speed.
- `#define BOOT_CPU_BYTE_ORDER_MOTOROLA (0)`
Motorola or Intel style byte ordering.
- `#define BOOT_CPU_USER_PROGRAM_START_HOOK (1)`
Enable/disable hook function call right before user program start.
- `#define BOOT_COM_CAN_ENABLE (1)`
Enable/disable CAN transport layer.
- `#define BOOT_COM_CAN_BAUDRATE (500000)`
Configure the desired CAN baudrate.
- `#define BOOT_COM_CAN_TX_MSG_ID (0x7E1 /*| 0x80000000*/)`
Configure CAN message ID target->host.
- `#define BOOT_COM_CAN_TX_MAX_DATA (8)`
Configure number of bytes in the target->host CAN message.
- `#define BOOT_COM_CAN_RX_MSG_ID (0x667 /*| 0x80000000*/)`
Configure CAN message ID host->target.
- `#define BOOT_COM_CAN_RX_MAX_DATA (8)`
Configure number of bytes in the host->target CAN message.
- `#define BOOT_COM_CAN_CHANNEL_INDEX (0)`
Select the desired CAN peripheral as a zero based index.
- `#define BOOT_COM_RS232_ENABLE (1)`
Enable/disable UART transport layer.
- `#define BOOT_COM_RS232_BAUDRATE (57600)`

- Configure the desired communication speed.*

 - #define `BOOT_COM_RS232_TX_MAX_DATA` (64)

Configure number of bytes in the target->host data packet.
- #define `BOOT_COM_RS232_RX_MAX_DATA` (64)

Configure number of bytes in the host->target data packet.
- #define `BOOT_COM_RS232_CHANNEL_INDEX` (1)

Select the desired UART peripheral as a zero based index.
- #define `BOOT_FILE_SYS_ENABLE` (1)

Enable/disable support for firmware updates from a locally attached storage.
- #define `BOOT_FILE_LOGGING_ENABLE` (1)

Enable/disable logging messages during firmware updates.
- #define `BOOT_FILE_ERROR_HOOK_ENABLE` (1)

Enable/disable a hook function that is called upon detection of an error.
- #define `BOOT_FILE_STARTED_HOOK_ENABLE` (1)

Enable/disable a hook function that is called at the start of the update.
- #define `BOOT_FILE_COMPLETED_HOOK_ENABLE` (1)

Enable/disable a hook function that is called at the end of the update.
- #define `BOOT_BACKDOOR_HOOKS_ENABLE` (0)

Enable/disable the backdoor override hook functions.
- #define `BOOT_NVM_HOOKS_ENABLE` (0)

Enable/disable the NVM hook function for supporting additional memory devices.
- #define `BOOT_NVM_SIZE_KB` (128)

Configure the size of the default memory device (typically flash EEPROM).
- #define `BOOT_NVM_CHECKSUM_HOOKS_ENABLE` (0)

Enable/disable hooks functions to override the user program checksum handling.
- #define `BOOT_FLASH_VECTOR_TABLE_CS_OFFSET` (0xec)

This microcontroller has a smaller vector table then the default STM32F1xx project as assumed in the bootloader's core. This means the user program has a different checksum location, because this one is added at the end of the user program's vector table.
- #define `BOOT_COP_HOOKS_ENABLE` (1)

Enable/disable the hook functions for controlling the watchdog.

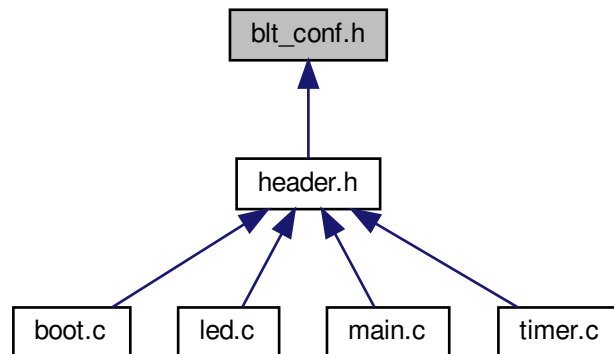
7.108.1 Detailed Description

Bootloader configuration header file.

7.109 blt_conf.h File Reference

Bootloader configuration header file.

This graph shows which files directly or indirectly include this file:



Macros

- `#define BOOT_CPU_XTAL_SPEED_KHZ (8000)`
Frequency of the external crystal oscillator.
- `#define BOOT_CPU_SYSTEM_SPEED_KHZ (72000)`
Desired system speed.
- `#define BOOT_CPU_BYTE_ORDER_MOTOROLA (0)`
Motorola or Intel style byte ordering.
- `#define BOOT_CPU_USER_PROGRAM_START_HOOK (1)`
Enable/disable hook function call right before user program start.
- `#define BOOT_COM_CAN_ENABLE (1)`
Enable/disable CAN transport layer.
- `#define BOOT_COM_CAN_BAUDRATE (500000)`
Configure the desired CAN baudrate.
- `#define BOOT_COM_CAN_TX_MSG_ID (0x7E1 /*| 0x80000000*/)`
Configure CAN message ID target->host.
- `#define BOOT_COM_CAN_TX_MAX_DATA (8)`
Configure number of bytes in the target->host CAN message.
- `#define BOOT_COM_CAN_RX_MSG_ID (0x667 /*| 0x80000000*/)`
Configure CAN message ID host->target.
- `#define BOOT_COM_CAN_RX_MAX_DATA (8)`
Configure number of bytes in the host->target CAN message.
- `#define BOOT_COM_CAN_CHANNEL_INDEX (0)`
Select the desired CAN peripheral as a zero based index.
- `#define BOOT_COM_RS232_ENABLE (1)`
Enable/disable UART transport layer.
- `#define BOOT_COM_RS232_BAUDRATE (57600)`
Configure the desired communication speed.
- `#define BOOT_COM_RS232_TX_MAX_DATA (64)`
Configure number of bytes in the target->host data packet.
- `#define BOOT_COM_RS232_RX_MAX_DATA (64)`

Configure number of bytes in the host->target data packet.

- #define `BOOT_COM_RS232_CHANNEL_INDEX` (1)

Select the desired UART peripheral as a zero based index.

- #define `BOOT_FILE_SYS_ENABLE` (1)

Enable/disable support for firmware updates from a locally attached storage.

- #define `BOOT_FILE_LOGGING_ENABLE` (1)

Enable/disable logging messages during firmware updates.

- #define `BOOT_FILE_ERROR_HOOK_ENABLE` (1)

Enable/disable a hook function that is called upon detection of an error.

- #define `BOOT_FILE_STARTED_HOOK_ENABLE` (1)

Enable/disable a hook function that is called at the start of the update.

- #define `BOOT_FILE_COMPLETED_HOOK_ENABLE` (1)

Enable/disable a hook function that is called at the end of the update.

- #define `BOOT_BACKDOOR_HOOKS_ENABLE` (0)

Enable/disable the backdoor override hook functions.

- #define `BOOT_NVM_HOOKS_ENABLE` (0)

Enable/disable the NVM hook function for supporting additional memory devices.

- #define `BOOT_NVM_SIZE_KB` (128)

Configure the size of the default memory device (typically flash EEPROM).

- #define `BOOT_NVM_CHECKSUM_HOOKS_ENABLE` (0)

Enable/disable hooks functions to override the user program checksum handling.

- #define `BOOT_FLASH_VECTOR_TABLE_CS_OFFSET` (0xec)

This microcontroller has a smaller vector table then the default STM32F1xx project as assumed in the bootloader's core. This means the user program has a different checksum location, because this one is added at the end of the user program's vector table.

- #define `BOOT_COP_HOOKS_ENABLE` (1)

Enable/disable the hook functions for controlling the watchdog.

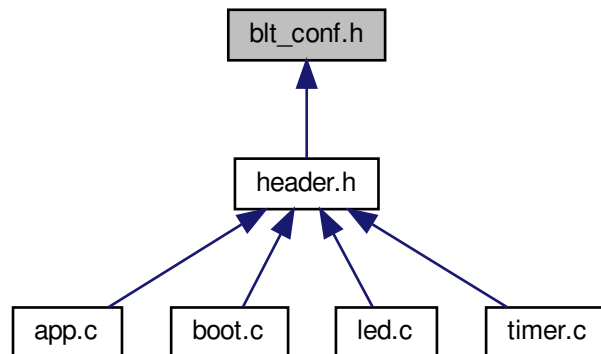
7.109.1 Detailed Description

Bootloader configuration header file.

7.110 blt_conf.h File Reference

Bootloader configuration header file.

This graph shows which files directly or indirectly include this file:



Macros

- `#define BOOT_CPU_XTAL_SPEED_KHZ (8000)`
Frequency of the external crystal oscillator.
- `#define BOOT_CPU_SYSTEM_SPEED_KHZ (72000)`
Desired system speed.
- `#define BOOT_CPU_BYTE_ORDER_MOTOROLA (0)`
Motorola or Intel style byte ordering.
- `#define BOOT_CPU_USER_PROGRAM_START_HOOK (1)`
Enable/disable hook function call right before user program start.
- `#define BOOT_COM_CAN_ENABLE (0)`
Enable/disable CAN transport layer.
- `#define BOOT_COM_CAN_BAUDRATE (500000)`
Configure the desired CAN baudrate.
- `#define BOOT_COM_CAN_TX_MSG_ID (0x7E1 /*| 0x80000000*/)`
Configure CAN message ID target->host.
- `#define BOOT_COM_CAN_TX_MAX_DATA (8)`
Configure number of bytes in the target->host CAN message.
- `#define BOOT_COM_CAN_RX_MSG_ID (0x667 /*| 0x80000000*/)`
Configure CAN message ID host->target.
- `#define BOOT_COM_CAN_RX_MAX_DATA (8)`
Configure number of bytes in the host->target CAN message.
- `#define BOOT_COM_CAN_CHANNEL_INDEX (0)`
Select the desired CAN peripheral as a zero based index.
- `#define BOOT_COM_USB_ENABLE (1)`
Enable/disable USB transport layer.
- `#define BOOT_COM_USB_TX_MAX_DATA (63)`
Configure number of bytes in the target->host data packet.
- `#define BOOT_COM_USB_RX_MAX_DATA (63)`
Configure number of bytes in the host->target data packet.
- `#define BOOT_FILE_SYS_ENABLE (1)`

- Enable/disable support for firmware updates from a locally attached storage.*

 - #define `BOOT_FILE_LOGGING_ENABLE` (1)

Enable/disable logging messages during firmware updates.
- #define `BOOT_FILE_ERROR_HOOK_ENABLE` (1)

Enable/disable a hook function that is called upon detection of an error.
- #define `BOOT_FILE_STARTED_HOOK_ENABLE` (1)

Enable/disable a hook function that is called at the start of the update.
- #define `BOOT_FILE_COMPLETED_HOOK_ENABLE` (1)

Enable/disable a hook function that is called at the end of the update.
- #define `BOOT_BACKDOOR_HOOKS_ENABLE` (0)

Enable/disable the backdoor override hook functions.
- #define `BOOT_NVM_HOOKS_ENABLE` (0)

Enable/disable the NVM hook function for supporting additional memory devices.
- #define `BOOT_NVM_SIZE_KB` (128)

Configure the size of the default memory device (typically flash EEPROM).
- #define `BOOT_NVM_CHECKSUM_HOOKS_ENABLE` (0)

Enable/disable hooks functions to override the user program checksum handling.
- #define `BOOT_FLASH_VECTOR_TABLE_CS_OFFSET` (0x10c)

This microcontroller has a smaller vector table then the default STM32F1xx project as assumed in the bootloader's core. This means the user program has a different checksum location, because this one is added at the end of the user program's vector table.
- #define `BOOT_FLASH_CUSTOM_LAYOUT_ENABLE` (1)

Enable support for a custom flash layout table. It is located in flash_layout.c. This was done because the default flashLayout[] table in the bootloader's core has more flash memory reserved for the bootloader than is needed for this demo.
- #define `BOOT_COP_HOOKS_ENABLE` (1)

Enable/disable the hook functions for controlling the watchdog.

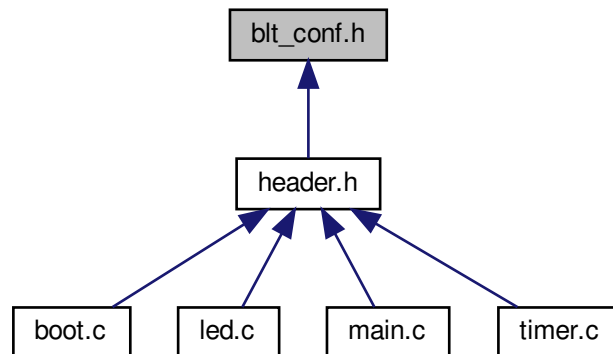
7.110.1 Detailed Description

Bootloader configuration header file.

7.111 blt_conf.h File Reference

Bootloader configuration header file.

This graph shows which files directly or indirectly include this file:



Macros

- `#define BOOT_CPU_XTAL_SPEED_KHZ (8000)`
Frequency of the external crystal oscillator.
- `#define BOOT_CPU_SYSTEM_SPEED_KHZ (72000)`
Desired system speed.
- `#define BOOT_CPU_BYTE_ORDER_MOTOROLA (0)`
Motorola or Intel style byte ordering.
- `#define BOOT_CPU_USER_PROGRAM_START_HOOK (1)`
Enable/disable hook function call right before user program start.
- `#define BOOT_COM_CAN_ENABLE (0)`
Enable/disable CAN transport layer.
- `#define BOOT_COM_CAN_BAUDRATE (500000)`
Configure the desired CAN baudrate.
- `#define BOOT_COM_CAN_TX_MSG_ID (0x7E1 /*| 0x80000000*/)`
Configure CAN message ID target->host.
- `#define BOOT_COM_CAN_TX_MAX_DATA (8)`
Configure number of bytes in the target->host CAN message.
- `#define BOOT_COM_CAN_RX_MSG_ID (0x667 /*| 0x80000000*/)`
Configure CAN message ID host->target.
- `#define BOOT_COM_CAN_RX_MAX_DATA (8)`
Configure number of bytes in the host->target CAN message.
- `#define BOOT_COM_CAN_CHANNEL_INDEX (0)`
Select the desired CAN peripheral as a zero based index.
- `#define BOOT_COM_USB_ENABLE (1)`
Enable/disable USB transport layer.
- `#define BOOT_COM_USB_TX_MAX_DATA (63)`
Configure number of bytes in the target->host data packet.
- `#define BOOT_COM_USB_RX_MAX_DATA (63)`
Configure number of bytes in the host->target data packet.
- `#define BOOT_FILE_SYS_ENABLE (1)`

- Enable/disable support for firmware updates from a locally attached storage.*

 - #define `BOOT_FILE_LOGGING_ENABLE` (1)

Enable/disable logging messages during firmware updates.
- #define `BOOT_FILE_ERROR_HOOK_ENABLE` (1)

Enable/disable a hook function that is called upon detection of an error.
- #define `BOOT_FILE_STARTED_HOOK_ENABLE` (1)

Enable/disable a hook function that is called at the start of the update.
- #define `BOOT_FILE_COMPLETED_HOOK_ENABLE` (1)

Enable/disable a hook function that is called at the end of the update.
- #define `BOOT_BACKDOOR_HOOKS_ENABLE` (0)

Enable/disable the backdoor override hook functions.
- #define `BOOT_NVM_HOOKS_ENABLE` (0)

Enable/disable the NVM hook function for supporting additional memory devices.
- #define `BOOT_NVM_SIZE_KB` (128)

Configure the size of the default memory device (typically flash EEPROM).
- #define `BOOT_NVM_CHECKSUM_HOOKS_ENABLE` (0)

Enable/disable hooks functions to override the user program checksum handling.
- #define `BOOT_FLASH_VECTOR_TABLE_CS_OFFSET` (0x10c)

This microcontroller has a smaller vector table then the default STM32F1xx project as assumed in the bootloader's core. This means the user program has a different checksum location, because this one is added at the end of the user program's vector table.
- #define `BOOT_FLASH_CUSTOM_LAYOUT_ENABLE` (1)

Enable support for a custom flash layout table. It is located in flash_layout.c. This was done because the default flashLayout[] table in the bootloader's core has more flash memory reserved for the bootloader than is needed for this demo.
- #define `BOOT_COP_HOOKS_ENABLE` (1)

Enable/disable the hook functions for controlling the watchdog.

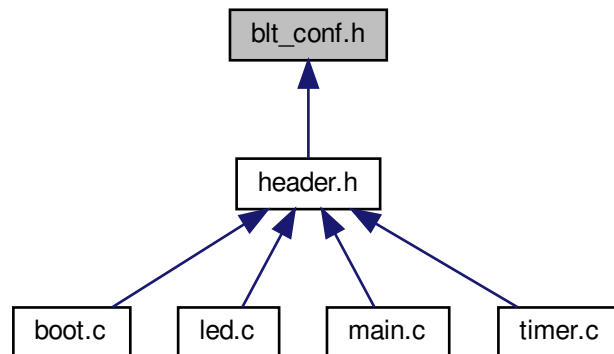
7.111.1 Detailed Description

Bootloader configuration header file.

7.112 blt_conf.h File Reference

Bootloader configuration header file.

This graph shows which files directly or indirectly include this file:



Macros

- `#define BOOT_CPU_XTAL_SPEED_KHZ (8000)`
Frequency of the external crystal oscillator.
- `#define BOOT_CPU_SYSTEM_SPEED_KHZ (72000)`
Desired system speed.
- `#define BOOT_CPU_BYTE_ORDER_MOTOROLA (0)`
Motorola or Intel style byte ordering.
- `#define BOOT_CPU_USER_PROGRAM_START_HOOK (1)`
Enable/disable hook function call right before user program start.
- `#define BOOT_COM_CAN_ENABLE (0)`
Enable/disable CAN transport layer.
- `#define BOOT_COM_CAN_BAUDRATE (500000)`
Configure the desired CAN baudrate.
- `#define BOOT_COM_CAN_TX_MSG_ID (0x7E1 /*| 0x80000000*/)`
Configure CAN message ID target->host.
- `#define BOOT_COM_CAN_TX_MAX_DATA (8)`
Configure number of bytes in the target->host CAN message.
- `#define BOOT_COM_CAN_RX_MSG_ID (0x667 /*| 0x80000000*/)`
Configure CAN message ID host->target.
- `#define BOOT_COM_CAN_RX_MAX_DATA (8)`
Configure number of bytes in the host->target CAN message.
- `#define BOOT_COM_CAN_CHANNEL_INDEX (0)`
Select the desired CAN peripheral as a zero based index.
- `#define BOOT_COM_USB_ENABLE (1)`
Enable/disable USB transport layer.
- `#define BOOT_COM_USB_TX_MAX_DATA (63)`
Configure number of bytes in the target->host data packet.
- `#define BOOT_COM_USB_RX_MAX_DATA (63)`
Configure number of bytes in the host->target data packet.
- `#define BOOT_FILE_SYS_ENABLE (1)`

- Enable/disable support for firmware updates from a locally attached storage.*

 - #define `BOOT_FILE_LOGGING_ENABLE` (1)

Enable/disable logging messages during firmware updates.
- #define `BOOT_FILE_ERROR_HOOK_ENABLE` (1)

Enable/disable a hook function that is called upon detection of an error.
- #define `BOOT_FILE_STARTED_HOOK_ENABLE` (1)

Enable/disable a hook function that is called at the start of the update.
- #define `BOOT_FILE_COMPLETED_HOOK_ENABLE` (1)

Enable/disable a hook function that is called at the end of the update.
- #define `BOOT_BACKDOOR_HOOKS_ENABLE` (0)

Enable/disable the backdoor override hook functions.
- #define `BOOT_NVM_HOOKS_ENABLE` (0)

Enable/disable the NVM hook function for supporting additional memory devices.
- #define `BOOT_NVM_SIZE_KB` (128)

Configure the size of the default memory device (typically flash EEPROM).
- #define `BOOT_NVM_CHECKSUM_HOOKS_ENABLE` (0)

Enable/disable hooks functions to override the user program checksum handling.
- #define `BOOT_FLASH_VECTOR_TABLE_CS_OFFSET` (0xec)

This microcontroller has a smaller vector table then the default STM32F1xx project as assumed in the bootloader's core. This means the user program has a different checksum location, because this one is added at the end of the user program's vector table.
- #define `BOOT_FLASH_CUSTOM_LAYOUT_ENABLE` (1)

Enable support for a custom flash layout table. It is located in flash_layout.c. This was done because the default flashLayout[] table in the bootloader's core has more flash memory reserved for the bootloader than is needed for this demo.
- #define `BOOT_COP_HOOKS_ENABLE` (1)

Enable/disable the hook functions for controlling the watchdog.

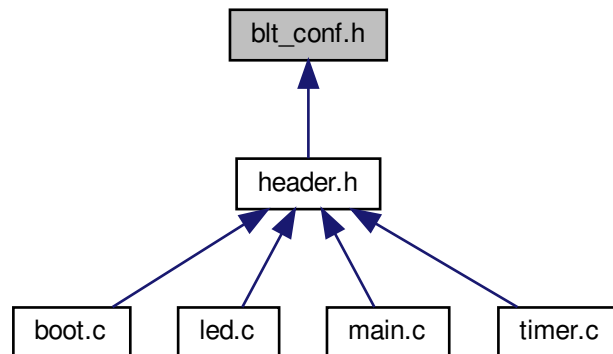
7.112.1 Detailed Description

Bootloader configuration header file.

7.113 blt_conf.h File Reference

Bootloader configuration header file.

This graph shows which files directly or indirectly include this file:



Macros

- `#define BOOT_CPU_XTAL_SPEED_KHZ (8000)`
Frequency of the external crystal oscillator.
- `#define BOOT_CPU_SYSTEM_SPEED_KHZ (72000)`
Desired system speed.
- `#define BOOT_CPU_BYTE_ORDER_MOTOROLA (0)`
Motorola or Intel style byte ordering.
- `#define BOOT_CPU_USER_PROGRAM_START_HOOK (1)`
Enable/disable hook function call right before user program start.
- `#define BOOT_COM_CAN_ENABLE (0)`
Enable/disable CAN transport layer.
- `#define BOOT_COM_CAN_BAUDRATE (500000)`
Configure the desired CAN baudrate.
- `#define BOOT_COM_CAN_TX_MSG_ID (0x7E1 /*| 0x80000000*/)`
Configure CAN message ID target->host.
- `#define BOOT_COM_CAN_TX_MAX_DATA (8)`
Configure number of bytes in the target->host CAN message.
- `#define BOOT_COM_CAN_RX_MSG_ID (0x667 /*| 0x80000000*/)`
Configure CAN message ID host->target.
- `#define BOOT_COM_CAN_RX_MAX_DATA (8)`
Configure number of bytes in the host->target CAN message.
- `#define BOOT_COM_CAN_CHANNEL_INDEX (0)`
Select the desired CAN peripheral as a zero based index.
- `#define BOOT_COM_USB_ENABLE (1)`
Enable/disable USB transport layer.
- `#define BOOT_COM_USB_TX_MAX_DATA (63)`
Configure number of bytes in the target->host data packet.
- `#define BOOT_COM_USB_RX_MAX_DATA (63)`
Configure number of bytes in the host->target data packet.
- `#define BOOT_FILE_SYS_ENABLE (1)`

- Enable/disable support for firmware updates from a locally attached storage.*

 - #define `BOOT_FILE_LOGGING_ENABLE` (1)

Enable/disable logging messages during firmware updates.
- #define `BOOT_FILE_ERROR_HOOK_ENABLE` (1)

Enable/disable a hook function that is called upon detection of an error.
- #define `BOOT_FILE_STARTED_HOOK_ENABLE` (1)

Enable/disable a hook function that is called at the start of the update.
- #define `BOOT_FILE_COMPLETED_HOOK_ENABLE` (1)

Enable/disable a hook function that is called at the end of the update.
- #define `BOOT_BACKDOOR_HOOKS_ENABLE` (0)

Enable/disable the backdoor override hook functions.
- #define `BOOT_NVM_HOOKS_ENABLE` (0)

Enable/disable the NVM hook function for supporting additional memory devices.
- #define `BOOT_NVM_SIZE_KB` (128)

Configure the size of the default memory device (typically flash EEPROM).
- #define `BOOT_NVM_CHECKSUM_HOOKS_ENABLE` (0)

Enable/disable hooks functions to override the user program checksum handling.
- #define `BOOT_FLASH_VECTOR_TABLE_CS_OFFSET` (0xec)

This microcontroller has a smaller vector table then the default STM32F1xx project as assumed in the bootloader's core. This means the user program has a different checksum location, because this one is added at the end of the user program's vector table.
- #define `BOOT_FLASH_CUSTOM_LAYOUT_ENABLE` (1)

Enable support for a custom flash layout table. It is located in flash_layout.c. This was done because the default flashLayout[] table in the bootloader's core has more flash memory reserved for the bootloader than is needed for this demo.
- #define `BOOT_COP_HOOKS_ENABLE` (1)

Enable/disable the hook functions for controlling the watchdog.

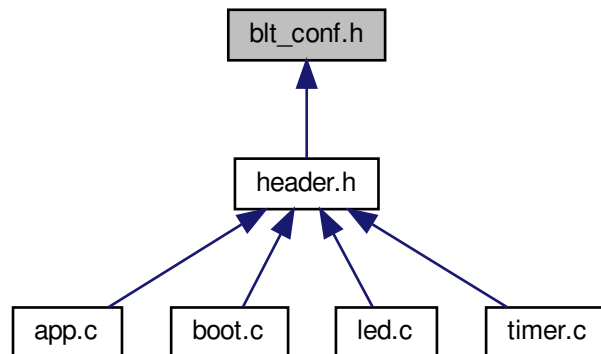
7.113.1 Detailed Description

Bootloader configuration header file.

7.114 blt_conf.h File Reference

Bootloader configuration header file.

This graph shows which files directly or indirectly include this file:



Macros

- `#define BOOT_CPU_XTAL_SPEED_KHZ (25000)`
Frequency of the external crystal oscillator.
- `#define BOOT_CPU_SYSTEM_SPEED_KHZ (120000)`
Desired system speed.
- `#define BOOT_CPU_BYTE_ORDER_MOTOROLA (0)`
Motorola or Intel style byte ordering.
- `#define BOOT_CPU_USER_PROGRAM_START_HOOK (1)`
Enable/disable hook function call right before user program start.
- `#define BOOT_COM_CAN_ENABLE (1)`
Enable/disable CAN transport layer.
- `#define BOOT_COM_CAN_BAUDRATE (500000)`
Configure the desired CAN baudrate.
- `#define BOOT_COM_CAN_TX_MSG_ID (0x7E1 /*| 0x80000000*/)`
Configure CAN message ID target->host.
- `#define BOOT_COM_CAN_TX_MAX_DATA (8)`
Configure number of bytes in the target->host CAN message.
- `#define BOOT_COM_CAN_RX_MSG_ID (0x667 /*| 0x80000000*/)`
Configure CAN message ID host->target.
- `#define BOOT_COM_CAN_RX_MAX_DATA (8)`
Configure number of bytes in the host->target CAN message.
- `#define BOOT_COM_CAN_CHANNEL_INDEX (0)`
Select the desired CAN peripheral as a zero based index.
- `#define BOOT_COM_RS232_ENABLE (1)`
Enable/disable UART transport layer.
- `#define BOOT_COM_RS232_BAUDRATE (57600)`
Configure the desired communication speed.
- `#define BOOT_COM_RS232_TX_MAX_DATA (64)`
Configure number of bytes in the target->host data packet.
- `#define BOOT_COM_RS232_RX_MAX_DATA (64)`

- Configure number of bytes in the host->target data packet.*

 - #define `BOOT_COM_RS232_CHANNEL_INDEX` (2)

Select the desired UART peripheral as a zero based index.
- #define `BOOT_FILE_SYS_ENABLE` (1)

Enable/disable support for firmware updates from a locally attached storage.
- #define `BOOT_FILE_LOGGING_ENABLE` (1)

Enable/disable logging messages during firmware updates.
- #define `BOOT_FILE_ERROR_HOOK_ENABLE` (1)

Enable/disable a hook function that is called upon detection of an error.
- #define `BOOT_FILE_STARTED_HOOK_ENABLE` (1)

Enable/disable a hook function that is called at the start of the update.
- #define `BOOT_FILE_COMPLETED_HOOK_ENABLE` (1)

Enable/disable a hook function that is called at the end of the update.
- #define `BOOT_BACKDOOR_HOOKS_ENABLE` (0)

Enable/disable the backdoor override hook functions.
- #define `BOOT_NVM_HOOKS_ENABLE` (0)

Enable/disable the NVM hook function for supporting additional memory devices.
- #define `BOOT_NVM_SIZE_KB` (512)

Configure the size of the default memory device (typically flash EEPROM).
- #define `BOOT_NVM_CHECKSUM_HOOKS_ENABLE` (0)

Enable/disable hooks functions to override the user program checksum handling.
- #define `BOOT_COP_HOOKS_ENABLE` (1)

Enable/disable the hook functions for controlling the watchdog.

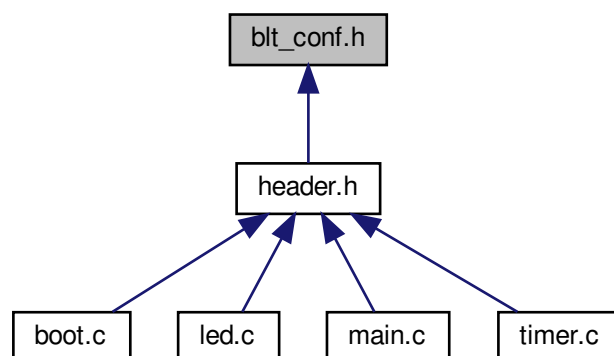
7.114.1 Detailed Description

Bootloader configuration header file.

7.115 blt_conf.h File Reference

Bootloader configuration header file.

This graph shows which files directly or indirectly include this file:



Macros

- #define `BOOT_CPU_XTAL_SPEED_KHZ` (25000)
Frequency of the external crystal oscillator.
- #define `BOOT_CPU_SYSTEM_SPEED_KHZ` (120000)
Desired system speed.
- #define `BOOT_CPU_BYTE_ORDER_MOTOROLA` (0)
Motorola or Intel style byte ordering.
- #define `BOOT_CPU_USER_PROGRAM_START_HOOK` (1)
Enable/disable hook function call right before user program start.
- #define `BOOT_COM_CAN_ENABLE` (1)
Enable/disable CAN transport layer.
- #define `BOOT_COM_CAN_BAUDRATE` (500000)
Configure the desired CAN baudrate.
- #define `BOOT_COM_CAN_TX_MSG_ID` (0x7E1 /*| 0x80000000*/) *Configure CAN message ID target->host.*
- #define `BOOT_COM_CAN_TX_MAX_DATA` (8)
Configure number of bytes in the target->host CAN message.
- #define `BOOT_COM_CAN_RX_MSG_ID` (0x667 /*| 0x80000000*/) *Configure CAN message ID host->target.*
- #define `BOOT_COM_CAN_RX_MAX_DATA` (8)
Configure number of bytes in the host->target CAN message.
- #define `BOOT_COM_CAN_CHANNEL_INDEX` (0)
Select the desired CAN peripheral as a zero based index.
- #define `BOOT_COM_RS232_ENABLE` (1)
Enable/disable UART transport layer.
- #define `BOOT_COM_RS232_BAUDRATE` (57600)
Configure the desired communication speed.
- #define `BOOT_COM_RS232_TX_MAX_DATA` (64)
Configure number of bytes in the target->host data packet.
- #define `BOOT_COM_RS232_RX_MAX_DATA` (64)
Configure number of bytes in the host->target data packet.
- #define `BOOT_COM_RS232_CHANNEL_INDEX` (2)
Select the desired UART peripheral as a zero based index.
- #define `BOOT_FILE_SYS_ENABLE` (1)
Enable/disable support for firmware updates from a locally attached storage.
- #define `BOOT_FILE_LOGGING_ENABLE` (1)
Enable/disable logging messages during firmware updates.
- #define `BOOT_FILE_ERROR_HOOK_ENABLE` (1)
Enable/disable a hook function that is called upon detection of an error.
- #define `BOOT_FILE_STARTED_HOOK_ENABLE` (1)
Enable/disable a hook function that is called at the start of the update.
- #define `BOOT_FILE_COMPLETED_HOOK_ENABLE` (1)
Enable/disable a hook function that is called at the end of the update.
- #define `BOOT_BACKDOOR_HOOKS_ENABLE` (0)
Enable/disable the backdoor override hook functions.
- #define `BOOT_NVM_HOOKS_ENABLE` (0)
Enable/disable the NVM hook function for supporting additional memory devices.
- #define `BOOT_NVM_SIZE_KB` (512)
Configure the size of the default memory device (typically flash EEPROM).
- #define `BOOT_NVM_CHECKSUM_HOOKS_ENABLE` (0)
Enable/disable hooks functions to override the user program checksum handling.
- #define `BOOT_COP_HOOKS_ENABLE` (1)
Enable/disable the hook functions for controlling the watchdog.

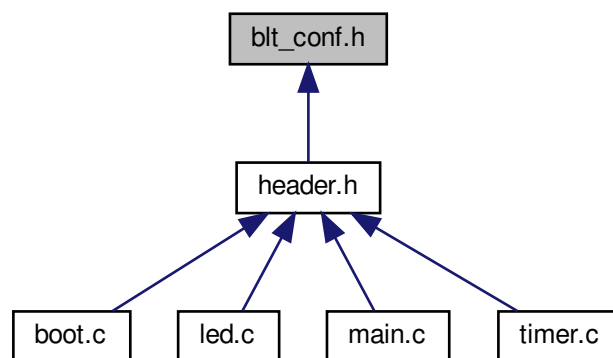
7.115.1 Detailed Description

Bootloader configuration header file.

7.116 blt_conf.h File Reference

Bootloader configuration header file.

This graph shows which files directly or indirectly include this file:



Macros

- `#define BOOT_CPU_XTAL_SPEED_KHZ (25000)`
Frequency of the external crystal oscillator.
- `#define BOOT_CPU_SYSTEM_SPEED_KHZ (120000)`
Desired system speed.
- `#define BOOT_CPU_BYTE_ORDER_MOTOROLA (0)`
Motorola or Intel style byte ordering.
- `#define BOOT_CPU_USER_PROGRAM_START_HOOK (1)`
Enable/disable hook function call right before user program start.
- `#define BOOT_COM_CAN_ENABLE (1)`
Enable/disable CAN transport layer.
- `#define BOOT_COM_CAN_BAUDRATE (500000)`
Configure the desired CAN baudrate.
- `#define BOOT_COM_CAN_TX_MSG_ID (0x7E1 /*| 0x80000000*/)`
Configure CAN message ID target->host.
- `#define BOOT_COM_CAN_TX_MAX_DATA (8)`
Configure number of bytes in the target->host CAN message.
- `#define BOOT_COM_CAN_RX_MSG_ID (0x667 /*| 0x80000000*/)`
Configure CAN message ID host->target.
- `#define BOOT_COM_CAN_RX_MAX_DATA (8)`

- Configure number of bytes in the host->target CAN message.*

 - #define `BOOT_COM_CAN_CHANNEL_INDEX` (0)

Select the desired CAN peripheral as a zero based index.
- #define `BOOT_COM_RS232_ENABLE` (1)

Enable/disable UART transport layer.
- #define `BOOT_COM_RS232_BAUDRATE` (57600)

Configure the desired communication speed.
- #define `BOOT_COM_RS232_TX_MAX_DATA` (64)

Configure number of bytes in the target->host data packet.
- #define `BOOT_COM_RS232_RX_MAX_DATA` (64)

Configure number of bytes in the host->target data packet.
- #define `BOOT_COM_RS232_CHANNEL_INDEX` (2)

Select the desired UART peripheral as a zero based index.
- #define `BOOT_FILE_SYS_ENABLE` (1)

Enable/disable support for firmware updates from a locally attached storage.
- #define `BOOT_FILE_LOGGING_ENABLE` (1)

Enable/disable logging messages during firmware updates.
- #define `BOOT_FILE_ERROR_HOOK_ENABLE` (1)

Enable/disable a hook function that is called upon detection of an error.
- #define `BOOT_FILE_STARTED_HOOK_ENABLE` (1)

Enable/disable a hook function that is called at the start of the update.
- #define `BOOT_FILE_COMPLETED_HOOK_ENABLE` (1)

Enable/disable a hook function that is called at the end of the update.
- #define `BOOT_BACKDOOR_HOOKS_ENABLE` (0)

Enable/disable the backdoor override hook functions.
- #define `BOOT_NVM_HOOKS_ENABLE` (0)

Enable/disable the NVM hook function for supporting additional memory devices.
- #define `BOOT_NVM_SIZE_KB` (512)

Configure the size of the default memory device (typically flash EEPROM).
- #define `BOOT_NVM_CHECKSUM_HOOKS_ENABLE` (0)

Enable/disable hooks functions to override the user program checksum handling.
- #define `BOOT_COP_HOOKS_ENABLE` (1)

Enable/disable the hook functions for controlling the watchdog.

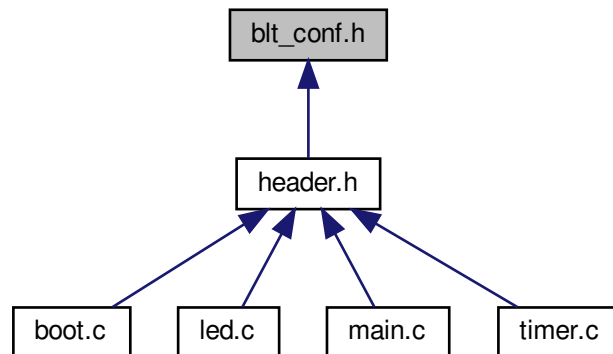
7.116.1 Detailed Description

Bootloader configuration header file.

7.117 blt_conf.h File Reference

Bootloader configuration header file.

This graph shows which files directly or indirectly include this file:



Macros

- #define `BOOT_CPU_XTAL_SPEED_KHZ` (25000)
Frequency of the external crystal oscillator.
- #define `BOOT_CPU_SYSTEM_SPEED_KHZ` (120000)
Desired system speed.
- #define `BOOT_CPU_BYTE_ORDER_MOTOROLA` (0)
Motorola or Intel style byte ordering.
- #define `BOOT_CPU_USER_PROGRAM_START_HOOK` (1)
Enable/disable hook function call right before user program start.
- #define `BOOT_COM_CAN_ENABLE` (1)
Enable/disable CAN transport layer.
- #define `BOOT_COM_CAN_BAUDRATE` (500000)
Configure the desired CAN baudrate.
- #define `BOOT_COM_CAN_TX_MSG_ID` (0x7E1 /*| 0x80000000*/) *Configure CAN message ID target->host.*
- #define `BOOT_COM_CAN_TX_MAX_DATA` (8)
Configure number of bytes in the target->host CAN message.
- #define `BOOT_COM_CAN_RX_MSG_ID` (0x667 /*| 0x80000000*/) *Configure CAN message ID host->target.*
- #define `BOOT_COM_CAN_RX_MAX_DATA` (8)
Configure number of bytes in the host->target CAN message.
- #define `BOOT_COM_CAN_CHANNEL_INDEX` (0)
Select the desired CAN peripheral as a zero based index.
- #define `BOOT_COM_RS232_ENABLE` (1)
Enable/disable UART transport layer.
- #define `BOOT_COM_RS232_BAUDRATE` (57600)
Configure the desired communication speed.
- #define `BOOT_COM_RS232_TX_MAX_DATA` (64)
Configure number of bytes in the target->host data packet.
- #define `BOOT_COM_RS232_RX_MAX_DATA` (64)

- Configure number of bytes in the host->target data packet.*

 - #define `BOOT_COM_RS232_CHANNEL_INDEX` (2)

Select the desired UART peripheral as a zero based index.
- #define `BOOT_FILE_SYS_ENABLE` (1)

Enable/disable support for firmware updates from a locally attached storage.
- #define `BOOT_FILE_LOGGING_ENABLE` (1)

Enable/disable logging messages during firmware updates.
- #define `BOOT_FILE_ERROR_HOOK_ENABLE` (1)

Enable/disable a hook function that is called upon detection of an error.
- #define `BOOT_FILE_STARTED_HOOK_ENABLE` (1)

Enable/disable a hook function that is called at the start of the update.
- #define `BOOT_FILE_COMPLETED_HOOK_ENABLE` (1)

Enable/disable a hook function that is called at the end of the update.
- #define `BOOT_BACKDOOR_HOOKS_ENABLE` (0)

Enable/disable the backdoor override hook functions.
- #define `BOOT_NVM_HOOKS_ENABLE` (0)

Enable/disable the NVM hook function for supporting additional memory devices.
- #define `BOOT_NVM_SIZE_KB` (512)

Configure the size of the default memory device (typically flash EEPROM).
- #define `BOOT_NVM_CHECKSUM_HOOKS_ENABLE` (0)

Enable/disable hooks functions to override the user program checksum handling.
- #define `BOOT_COP_HOOKS_ENABLE` (1)

Enable/disable the hook functions for controlling the watchdog.

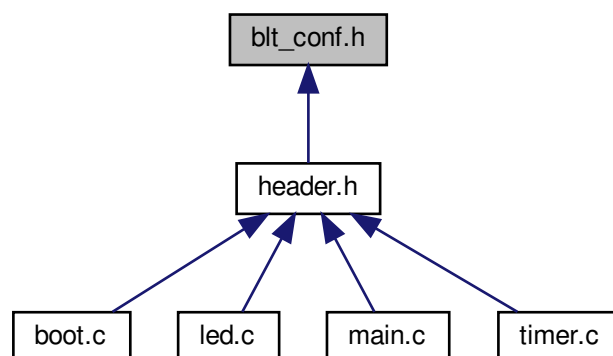
7.117.1 Detailed Description

Bootloader configuration header file.

7.118 blt_conf.h File Reference

Bootloader configuration header file.

This graph shows which files directly or indirectly include this file:



Macros

- #define [BOOT_CPU_XTAL_SPEED_KHZ](#) (8000)
Frequency of the external crystal oscillator.
- #define [BOOT_CPU_SYSTEM_SPEED_KHZ](#) (80000)
Desired system speed.
- #define [BOOT_CPU_BYTE_ORDER_MOTOROLA](#) (0)
Motorola or Intel style byte ordering.
- #define [BOOT_CPU_USER_PROGRAM_START_HOOK](#) (1)
Enable/disable hook function call right before user program start.
- #define [BOOT_COM_RS232_ENABLE](#) (1)
Enable/disable UART transport layer.
- #define [BOOT_COM_RS232_BAUDRATE](#) (57600)
Configure the desired communication speed.
- #define [BOOT_COM_RS232_TX_MAX_DATA](#) (64)
Configure number of bytes in the target->host data packet.
- #define [BOOT_COM_RS232_RX_MAX_DATA](#) (64)
Configure number of bytes in the host->target data packet.
- #define [BOOT_COM_RS232_CHANNEL_INDEX](#) (1)
Select the desired UART peripheral as a zero based index.
- #define [BOOT_COM_CAN_ENABLE](#) (1)
Enable/disable CAN transport layer.
- #define [BOOT_COM_CAN_BAUDRATE](#) (500000)
Configure the desired CAN baudrate.
- #define [BOOT_COM_CAN_TX_MSG_ID](#) (0x7E1 /*| 0x80000000*/) *Configure CAN message ID target->host.*
- #define [BOOT_COM_CAN_TX_MAX_DATA](#) (8)
Configure number of bytes in the target->host CAN message.
- #define [BOOT_COM_CAN_RX_MSG_ID](#) (0x667 /*| 0x80000000*/) *Configure CAN message ID host->target.*
- #define [BOOT_COM_CAN_RX_MAX_DATA](#) (8)
Configure number of bytes in the host->target CAN message.
- #define [BOOT_COM_CAN_CHANNEL_INDEX](#) (0)
Select the desired CAN peripheral as a zero based index.
- #define [BOOT_BACKDOOR_HOOKS_ENABLE](#) (0)
Enable/disable the backdoor override hook functions.
- #define [BOOT_NVM_HOOKS_ENABLE](#) (0)
Enable/disable the NVM hook function for supporting additional memory devices.
- #define [BOOT_NVM_SIZE_KB](#) (512)
Configure the size of the default memory device (typically flash EEPROM).
- #define [BOOT_NVM_CHECKSUM_HOOKS_ENABLE](#) (0)
Enable/disable hooks functions to override the user program checksum handling.
- #define [BOOT_COP_HOOKS_ENABLE](#) (1)
Enable/disable the hook functions for controlling the watchdog.

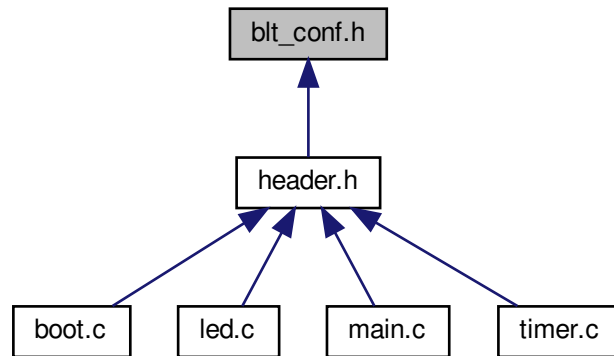
7.118.1 Detailed Description

Bootloader configuration header file.

7.119 blt_conf.h File Reference

Bootloader configuration header file.

This graph shows which files directly or indirectly include this file:



Macros

- #define `BOOT_CPU_XTAL_SPEED_KHZ` (8000)
Frequency of the external crystal oscillator.
- #define `BOOT_CPU_SYSTEM_SPEED_KHZ` (80000)
Desired system speed.
- #define `BOOT_CPU_BYTE_ORDER_MOTOROLA` (0)
Motorola or Intel style byte ordering.
- #define `BOOT_CPU_USER_PROGRAM_START_HOOK` (1)
Enable/disable hook function call right before user program start.
- #define `BOOT_COM_RS232_ENABLE` (1)
Enable/disable UART transport layer.
- #define `BOOT_COM_RS232_BAUDRATE` (57600)
Configure the desired communication speed.
- #define `BOOT_COM_RS232_TX_MAX_DATA` (64)
Configure number of bytes in the target->host data packet.
- #define `BOOT_COM_RS232_RX_MAX_DATA` (64)
Configure number of bytes in the host->target data packet.
- #define `BOOT_COM_RS232_CHANNEL_INDEX` (1)
Select the desired UART peripheral as a zero based index.
- #define `BOOT_COM_CAN_ENABLE` (1)
Enable/disable CAN transport layer.
- #define `BOOT_COM_CAN_BAUDRATE` (500000)
Configure the desired CAN baudrate.
- #define `BOOT_COM_CAN_TX_MSG_ID` (0x7E1 /*| 0x80000000*/) *Configure CAN message ID target->host.*
- #define `BOOT_COM_CAN_TX_MAX_DATA` (8)

- *Configure number of bytes in the target->host CAN message.*
 • #define `BOOT_COM_CAN_RX_MSG_ID` (0x667 /*| 0x80000000*/)
- *Configure CAN message ID host->target.*
 • #define `BOOT_COM_CAN_RX_MAX_DATA` (8)
- *Configure number of bytes in the host->target CAN message.*
 • #define `BOOT_COM_CAN_CHANNEL_INDEX` (0)
- *Select the desired CAN peripheral as a zero based index.*
 • #define `BOOT_BACKDOOR_HOOKS_ENABLE` (0)
- *Enable/disable the backdoor override hook functions.*
 • #define `BOOT_NVM_HOOKS_ENABLE` (0)
- *Enable/disable the NVM hook function for supporting additional memory devices.*
 • #define `BOOT_NVM_SIZE_KB` (512)
- *Configure the size of the default memory device (typically flash EEPROM).*
 • #define `BOOT_NVM_CHECKSUM_HOOKS_ENABLE` (0)
- *Enable/disable hooks functions to override the user program checksum handling.*
 • #define `BOOT_COP_HOOKS_ENABLE` (1)
- *Enable/disable the hook functions for controlling the watchdog.*

7.119.1 Detailed Description

Bootloader configuration header file.

7.120 blt_conf.h File Reference

Bootloader configuration header file.

Macros

- #define `BOOT_CPU_XTAL_SPEED_KHZ` (8000)
Frequency of the external crystal oscillator.
- #define `BOOT_CPU_SYSTEM_SPEED_KHZ` (72000)
Desired system speed.
- #define `BOOT_CPU_BYTE_ORDER_MOTOROLA` (0)
Motorola or Intel style byte ordering.
- #define `BOOT_CPU_USER_PROGRAM_START_HOOK` (1)
Enable/disable hook function call right before user program start.
- #define `BOOT_COM_USB_ENABLE` (1)
Enable/disable USB transport layer.
- #define `BOOT_COM_USB_TX_MAX_DATA` (63)
Configure number of bytes in the target->host data packet.
- #define `BOOT_COM_USB_RX_MAX_DATA` (63)
Configure number of bytes in the host->target data packet.
- #define `BOOT_COM_RS232_ENABLE` (0)
Enable/disable UART transport layer.
- #define `BOOT_COM_RS232_BAUDRATE` (57600)
Configure the desired communication speed.
- #define `BOOT_COM_RS232_TX_MAX_DATA` (64)

- *Configure number of bytes in the target->host data packet.*
- `#define BOOT_COM_RS232_RX_MAX_DATA (64)`
- *Configure number of bytes in the host->target data packet.*
- `#define BOOT_COM_RS232_CHANNEL_INDEX (1)`
- *Select the desired UART peripheral as a zero based index.*
- `#define BOOT_BACKDOOR_HOOKS_ENABLE (0)`
- *Enable/disable the backdoor override hook functions.*
- `#define BOOT_NVM_HOOKS_ENABLE (0)`
- *Enable/disable the NVM hook function for supporting additional memory devices.*
- `#define BOOT_NVM_SIZE_KB (256)`
- *Configure the size of the default memory device (typically flash EEPROM).*
- `#define BOOT_NVM_CHECKSUM_HOOKS_ENABLE (0)`
- *Enable/disable hooks functions to override the user program checksum handling.*
- `#define BOOT_FLASH_CUSTOM_LAYOUT_ENABLE (1)`
- *Enable support for a custom flash layout table. It is located in flash_layout.c. This was done because the default flashLayout[] table in the bootloader's core has more flash memory reserved for the bootloader than is needed for this demo.*
- `#define BOOT_COP_HOOKS_ENABLE (1)`
- *Enable/disable the hook functions for controlling the watchdog.*

7.120.1 Detailed Description

Bootloader configuration header file.

7.121 blt_conf.h File Reference

Bootloader configuration header file.

Macros

- `#define BOOT_CPU_XTAL_SPEED_KHZ (8000)`
- *Frequency of the external crystal oscillator.*
- `#define BOOT_CPU_SYSTEM_SPEED_KHZ (72000)`
- *Desired system speed.*
- `#define BOOT_CPU_BYTE_ORDER_MOTOROLA (0)`
- *Motorola or Intel style byte ordering.*
- `#define BOOT_CPU_USER_PROGRAM_START_HOOK (1)`
- *Enable/disable hook function call right before user program start.*
- `#define BOOT_COM_USB_ENABLE (1)`
- *Enable/disable USB transport layer.*
- `#define BOOT_COM_USB_TX_MAX_DATA (63)`
- *Configure number of bytes in the target->host data packet.*
- `#define BOOT_COM_USB_RX_MAX_DATA (63)`
- *Configure number of bytes in the host->target data packet.*
- `#define BOOT_COM_RS232_ENABLE (0)`
- *Enable/disable UART transport layer.*
- `#define BOOT_COM_RS232_BAUDRATE (57600)`

- Configure the desired communication speed.*

 - #define `BOOT_COM_RS232_TX_MAX_DATA` (64)

Configure number of bytes in the target->host data packet.

- #define `BOOT_COM_RS232_RX_MAX_DATA` (64)

Configure number of bytes in the host->target data packet.

- #define `BOOT_COM_RS232_CHANNEL_INDEX` (1)

Select the desired UART peripheral as a zero based index.

- #define `BOOT_BACKDOOR_HOOKS_ENABLE` (0)

Enable/disable the backdoor override hook functions.

- #define `BOOT_NVM_HOOKS_ENABLE` (0)

Enable/disable the NVM hook function for supporting additional memory devices.

- #define `BOOT_NVM_SIZE_KB` (256)

Configure the size of the default memory device (typically flash EEPROM).

- #define `BOOT_NVM_CHECKSUM_HOOKS_ENABLE` (0)

Enable/disable hooks functions to override the user program checksum handling.

- #define `BOOT_FLASH_CUSTOM_LAYOUT_ENABLE` (1)

Enable support for a custom flash layout table. It is located in flash_layout.c. This was done because the default flashLayout[] table in the bootloader's core has more flash memory reserved for the bootloader than is needed for this demo.

- #define `BOOT_COP_HOOKS_ENABLE` (1)

Enable/disable the hook functions for controlling the watchdog.

7.121.1 Detailed Description

Bootloader configuration header file.

7.122 blt_conf.h File Reference

Bootloader configuration header file.

Macros

- #define `BOOT_CPU_XTAL_SPEED_KHZ` (8000)

Frequency of the external crystal oscillator.

- #define `BOOT_CPU_SYSTEM_SPEED_KHZ` (72000)

Desired system speed.

- #define `BOOT_CPU_BYTE_ORDER_MOTOROLA` (0)

Motorola or Intel style byte ordering.

- #define `BOOT_CPU_USER_PROGRAM_START_HOOK` (1)

Enable/disable hook function call right before user program start.

- #define `BOOT_COM_USB_ENABLE` (1)

Enable/disable USB transport layer.

- #define `BOOT_COM_USB_TX_MAX_DATA` (63)

Configure number of bytes in the target->host data packet.

- #define `BOOT_COM_USB_RX_MAX_DATA` (63)

Configure number of bytes in the host->target data packet.

- #define `BOOT_COM_RS232_ENABLE` (0)

- *Enable/disable UART transport layer.*
- #define [BOOT_COM_RS232_BAUDRATE](#) (57600)
Configure the desired communication speed.
- #define [BOOT_COM_RS232_TX_MAX_DATA](#) (64)
Configure number of bytes in the target->host data packet.
- #define [BOOT_COM_RS232_RX_MAX_DATA](#) (64)
Configure number of bytes in the host->target data packet.
- #define [BOOT_COM_RS232_CHANNEL_INDEX](#) (1)
Select the desired UART peripheral as a zero based index.
- #define [BOOT_BACKDOOR_HOOKS_ENABLE](#) (0)
Enable/disable the backdoor override hook functions.
- #define [BOOT_NVM_HOOKS_ENABLE](#) (0)
Enable/disable the NVM hook function for supporting additional memory devices.
- #define [BOOT_NVM_SIZE_KB](#) (256)
Configure the size of the default memory device (typically flash EEPROM).
- #define [BOOT_NVM_CHECKSUM_HOOKS_ENABLE](#) (0)
Enable/disable hooks functions to override the user program checksum handling.
- #define [BOOT_FLASH_CUSTOM_LAYOUT_ENABLE](#) (1)
Enable support for a custom flash layout table. It is located in flash_layout.c. This was done because the default flashLayout[] table in the bootloader's core has more flash memory reserved for the bootloader than is needed for this demo.
- #define [BOOT_COP_HOOKS_ENABLE](#) (1)
Enable/disable the hook functions for controlling the watchdog.

7.122.1 Detailed Description

Bootloader configuration header file.

7.123 blt_conf.h File Reference

Bootloader configuration header file.

Macros

- #define [BOOT_CPU_XTAL_SPEED_KHZ](#) (8000)
Frequency of the external crystal oscillator.
- #define [BOOT_CPU_SYSTEM_SPEED_KHZ](#) (72000)
Desired system speed.
- #define [BOOT_CPU_BYTE_ORDER_MOTOROLA](#) (0)
Motorola or Intel style byte ordering.
- #define [BOOT_CPU_USER_PROGRAM_START_HOOK](#) (1)
Enable/disable hook function call right before user program start.
- #define [BOOT_COM_USB_ENABLE](#) (1)
Enable/disable USB transport layer.
- #define [BOOT_COM_USB_TX_MAX_DATA](#) (63)
Configure number of bytes in the target->host data packet.
- #define [BOOT_COM_USB_RX_MAX_DATA](#) (63)

- *Configure number of bytes in the host->target data packet.*
• #define [BOOT_COM_RS232_ENABLE](#) (0)
Enable/disable UART transport layer.
- #define [BOOT_COM_RS232_BAUDRATE](#) (57600)
Configure the desired communication speed.
- #define [BOOT_COM_RS232_TX_MAX_DATA](#) (64)
Configure number of bytes in the target->host data packet.
- #define [BOOT_COM_RS232_RX_MAX_DATA](#) (64)
Configure number of bytes in the host->target data packet.
- #define [BOOT_COM_RS232_CHANNEL_INDEX](#) (1)
Select the desired UART peripheral as a zero based index.
- #define [BOOT_BACKDOOR_HOOKS_ENABLE](#) (0)
Enable/disable the backdoor override hook functions.
- #define [BOOT_NVM_HOOKS_ENABLE](#) (0)
Enable/disable the NVM hook function for supporting additional memory devices.
- #define [BOOT_NVM_SIZE_KB](#) (256)
Configure the size of the default memory device (typically flash EEPROM).
- #define [BOOT_NVM_CHECKSUM_HOOKS_ENABLE](#) (0)
Enable/disable hooks functions to override the user program checksum handling.
- #define [BOOT_FLASH_CUSTOM_LAYOUT_ENABLE](#) (1)
Enable support for a custom flash layout table. It is located in flash_layout.c. This was done because the default flashLayout[] table in the bootloader's core has more flash memory reserved for the bootloader than is needed for this demo.
- #define [BOOT_COP_HOOKS_ENABLE](#) (1)
Enable/disable the hook functions for controlling the watchdog.

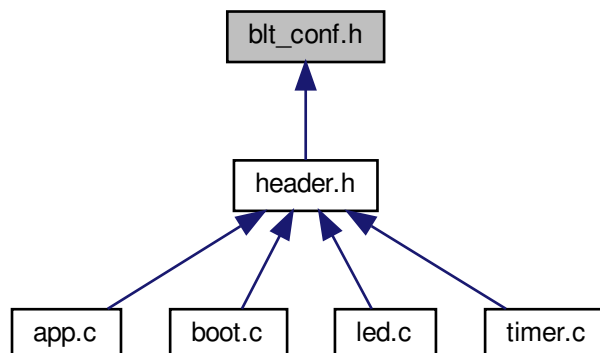
7.123.1 Detailed Description

Bootloader configuration header file.

7.124 blt_conf.h File Reference

Bootloader configuration header file.

This graph shows which files directly or indirectly include this file:



Macros

- #define `BOOT_CPU_XTAL_SPEED_KHZ` (8000)
Frequency of the external crystal oscillator.
- #define `BOOT_CPU_SYSTEM_SPEED_KHZ` (64000)
Desired system speed.
- #define `BOOT_CPU_BYTE_ORDER_MOTOROLA` (0)
Motorola or Intel style byte ordering.
- #define `BOOT_CPU_USER_PROGRAM_START_HOOK` (1)
Enable/disable hook function call right before user program start.
- #define `BOOT_COM_CAN_ENABLE` (1)
Enable/disable CAN transport layer.
- #define `BOOT_COM_CAN_BAUDRATE` (500000)
Configure the desired CAN baudrate.
- #define `BOOT_COM_CAN_TX_MSG_ID` (0x7E1 /*| 0x80000000*/) *Configure CAN message ID target->host.*
- #define `BOOT_COM_CAN_TX_MAX_DATA` (8)
Configure number of bytes in the target->host CAN message.
- #define `BOOT_COM_CAN_RX_MSG_ID` (0x667 /*| 0x80000000*/) *Configure CAN message ID host->target.*
- #define `BOOT_COM_CAN_RX_MAX_DATA` (8)
Configure number of bytes in the host->target CAN message.
- #define `BOOT_COM_CAN_CHANNEL_INDEX` (0)
Select the desired CAN peripheral as a zero based index.
- #define `BOOT_COM_RS232_ENABLE` (1)
Enable/disable UART transport layer.
- #define `BOOT_COM_RS232_BAUDRATE` (57600)
Configure the desired communication speed.
- #define `BOOT_COM_RS232_TX_MAX_DATA` (64)
Configure number of bytes in the target->host data packet.
- #define `BOOT_COM_RS232_RX_MAX_DATA` (64)
Configure number of bytes in the host->target data packet.
- #define `BOOT_COM_RS232_CHANNEL_INDEX` (1)
Select the desired UART peripheral as a zero based index.
- #define `BOOT_BACKDOOR_HOOKS_ENABLE` (0)
Enable/disable the backdoor override hook functions.
- #define `BOOT_NVM_HOOKS_ENABLE` (0)
Enable/disable the NVM hook function for supporting additional memory devices.
- #define `BOOT_NVM_SIZE_KB` (64)
Configure the size of the default memory device (typically flash EEPROM).
- #define `BOOT_NVM_CHECKSUM_HOOKS_ENABLE` (0)
Enable/disable hooks functions to override the user program checksum handling.
- #define `BOOT_COP_HOOKS_ENABLE` (1)
Enable/disable the hook functions for controlling the watchdog.

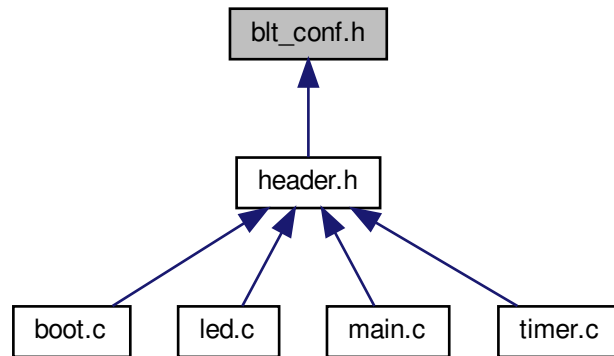
7.124.1 Detailed Description

Bootloader configuration header file.

7.125 blt_conf.h File Reference

Bootloader configuration header file.

This graph shows which files directly or indirectly include this file:



Macros

- `#define BOOT_CPU_XTAL_SPEED_KHZ (8000)`
Frequency of the external crystal oscillator.
- `#define BOOT_CPU_SYSTEM_SPEED_KHZ (64000)`
Desired system speed.
- `#define BOOT_CPU_BYTE_ORDER_MOTOROLA (0)`
Motorola or Intel style byte ordering.
- `#define BOOT_CPU_USER_PROGRAM_START_HOOK (1)`
Enable/disable hook function call right before user program start.
- `#define BOOT_COM_CAN_ENABLE (1)`
Enable/disable CAN transport layer.
- `#define BOOT_COM_CAN_BAUDRATE (500000)`
Configure the desired CAN baudrate.
- `#define BOOT_COM_CAN_TX_MSG_ID (0x7E1 /*| 0x80000000*/)`
Configure CAN message ID target->host.
- `#define BOOT_COM_CAN_TX_MAX_DATA (8)`
Configure number of bytes in the target->host CAN message.
- `#define BOOT_COM_CAN_RX_MSG_ID (0x667 /*| 0x80000000*/)`
Configure CAN message ID host->target.
- `#define BOOT_COM_CAN_RX_MAX_DATA (8)`
Configure number of bytes in the host->target CAN message.
- `#define BOOT_COM_CAN_CHANNEL_INDEX (0)`
Select the desired CAN peripheral as a zero based index.
- `#define BOOT_COM_RS232_ENABLE (1)`
Enable/disable UART transport layer.
- `#define BOOT_COM_RS232_BAUDRATE (57600)`

- *Configure the desired communication speed.*
 • #define `BOOT_COM_RS232_TX_MAX_DATA` (64)
Configure number of bytes in the target->host data packet.
- #define `BOOT_COM_RS232_RX_MAX_DATA` (64)
Configure number of bytes in the host->target data packet.
- #define `BOOT_COM_RS232_CHANNEL_INDEX` (1)
Select the desired UART peripheral as a zero based index.
- #define `BOOT_BACKDOOR_HOOKS_ENABLE` (0)
Enable/disable the backdoor override hook functions.
- #define `BOOT_NVM_HOOKS_ENABLE` (0)
Enable/disable the NVM hook function for supporting additional memory devices.
- #define `BOOT_NVM_SIZE_KB` (64)
Configure the size of the default memory device (typically flash EEPROM).
- #define `BOOT_NVM_CHECKSUM_HOOKS_ENABLE` (0)
Enable/disable hooks functions to override the user program checksum handling.
- #define `BOOT_COP_HOOKS_ENABLE` (1)
Enable/disable the hook functions for controlling the watchdog.

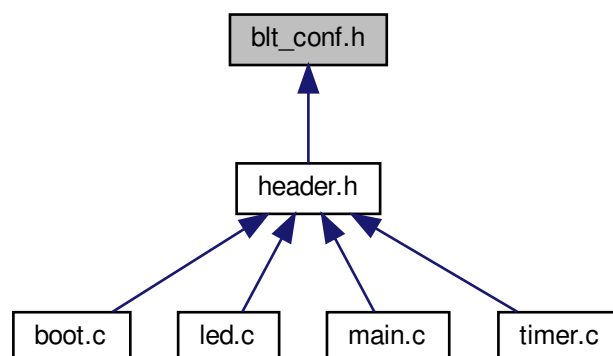
7.125.1 Detailed Description

Bootloader configuration header file.

7.126 blt_conf.h File Reference

Bootloader configuration header file.

This graph shows which files directly or indirectly include this file:



Macros

- #define `BOOT_CPU_XTAL_SPEED_KHZ` (8000)
Frequency of the external crystal oscillator.
- #define `BOOT_CPU_SYSTEM_SPEED_KHZ` (64000)
Desired system speed.
- #define `BOOT_CPU_BYTE_ORDER_MOTOROLA` (0)
Motorola or Intel style byte ordering.
- #define `BOOT_CPU_USER_PROGRAM_START_HOOK` (1)
Enable/disable hook function call right before user program start.
- #define `BOOT_COM_CAN_ENABLE` (1)
Enable/disable CAN transport layer.
- #define `BOOT_COM_CAN_BAUDRATE` (500000)
Configure the desired CAN baudrate.
- #define `BOOT_COM_CAN_TX_MSG_ID` (0x7E1 /*| 0x80000000*/) *Configure CAN message ID target->host.*
- #define `BOOT_COM_CAN_TX_MAX_DATA` (8)
Configure number of bytes in the target->host CAN message.
- #define `BOOT_COM_CAN_RX_MSG_ID` (0x667 /*| 0x80000000*/) *Configure CAN message ID host->target.*
- #define `BOOT_COM_CAN_RX_MAX_DATA` (8)
Configure number of bytes in the host->target CAN message.
- #define `BOOT_COM_CAN_CHANNEL_INDEX` (0)
Select the desired CAN peripheral as a zero based index.
- #define `BOOT_COM_RS232_ENABLE` (1)
Enable/disable UART transport layer.
- #define `BOOT_COM_RS232_BAUDRATE` (57600)
Configure the desired communication speed.
- #define `BOOT_COM_RS232_TX_MAX_DATA` (64)
Configure number of bytes in the target->host data packet.
- #define `BOOT_COM_RS232_RX_MAX_DATA` (64)
Configure number of bytes in the host->target data packet.
- #define `BOOT_COM_RS232_CHANNEL_INDEX` (1)
Select the desired UART peripheral as a zero based index.
- #define `BOOT_BACKDOOR_HOOKS_ENABLE` (0)
Enable/disable the backdoor override hook functions.
- #define `BOOT_NVM_HOOKS_ENABLE` (0)
Enable/disable the NVM hook function for supporting additional memory devices.
- #define `BOOT_NVM_SIZE_KB` (64)
Configure the size of the default memory device (typically flash EEPROM).
- #define `BOOT_NVM_CHECKSUM_HOOKS_ENABLE` (0)
Enable/disable hooks functions to override the user program checksum handling.
- #define `BOOT_COP_HOOKS_ENABLE` (1)
Enable/disable the hook functions for controlling the watchdog.

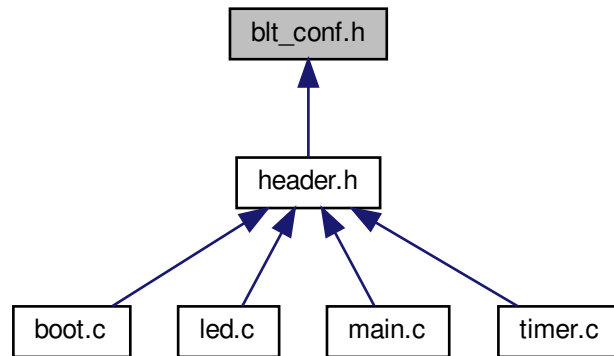
7.126.1 Detailed Description

Bootloader configuration header file.

7.127 blt_conf.h File Reference

Bootloader configuration header file.

This graph shows which files directly or indirectly include this file:



Macros

- #define `BOOT_CPU_XTAL_SPEED_KHZ` (8000)
Frequency of the external crystal oscillator.
- #define `BOOT_CPU_SYSTEM_SPEED_KHZ` (64000)
Desired system speed.
- #define `BOOT_CPU_BYTE_ORDER_MOTOROLA` (0)
Motorola or Intel style byte ordering.
- #define `BOOT_CPU_USER_PROGRAM_START_HOOK` (1)
Enable/disable hook function call right before user program start.
- #define `BOOT_COM_CAN_ENABLE` (1)
Enable/disable CAN transport layer.
- #define `BOOT_COM_CAN_BAUDRATE` (500000)
Configure the desired CAN baudrate.
- #define `BOOT_COM_CAN_TX_MSG_ID` (0x7E1 /*| 0x80000000*/) *Configure CAN message ID target->host.*
- #define `BOOT_COM_CAN_TX_MAX_DATA` (8)
Configure number of bytes in the target->host CAN message.
- #define `BOOT_COM_CAN_RX_MSG_ID` (0x667 /*| 0x80000000*/) *Configure CAN message ID host->target.*
- #define `BOOT_COM_CAN_RX_MAX_DATA` (8)
Configure number of bytes in the host->target CAN message.
- #define `BOOT_COM_CAN_CHANNEL_INDEX` (0)
Select the desired CAN peripheral as a zero based index.
- #define `BOOT_COM_RS232_ENABLE` (1)
Enable/disable UART transport layer.
- #define `BOOT_COM_RS232_BAUDRATE` (57600)

- Configure the desired communication speed.*

 - #define `BOOT_COM_RS232_TX_MAX_DATA` (64)

Configure number of bytes in the target->host data packet.
- #define `BOOT_COM_RS232_RX_MAX_DATA` (64)

Configure number of bytes in the host->target data packet.
- #define `BOOT_COM_RS232_CHANNEL_INDEX` (1)

Select the desired UART peripheral as a zero based index.
- #define `BOOT_BACKDOOR_HOOKS_ENABLE` (0)

Enable/disable the backdoor override hook functions.
- #define `BOOT_NVM_HOOKS_ENABLE` (0)

Enable/disable the NVM hook function for supporting additional memory devices.
- #define `BOOT_NVM_SIZE_KB` (64)

Configure the size of the default memory device (typically flash EEPROM).
- #define `BOOT_NVM_CHECKSUM_HOOKS_ENABLE` (0)

Enable/disable hooks functions to override the user program checksum handling.
- #define `BOOT_COP_HOOKS_ENABLE` (1)

Enable/disable the hook functions for controlling the watchdog.

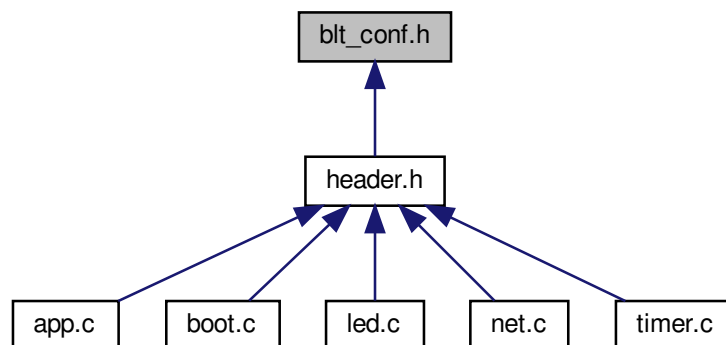
7.127.1 Detailed Description

Bootloader configuration header file.

7.128 blt_conf.h File Reference

Bootloader configuration header file.

This graph shows which files directly or indirectly include this file:



Macros

- `#define BOOT_CPU_XTAL_SPEED_KHZ (8000)`
Frequency of the external crystal oscillator.
- `#define BOOT_CPU_SYSTEM_SPEED_KHZ (168000)`
Desired system speed.
- `#define BOOT_CPU_BYTE_ORDER_MOTOROLA (0)`
Motorola or Intel style byte ordering.
- `#define BOOT_CPU_USER_PROGRAM_START_HOOK (1)`
Enable/disable hook function call right before user program start.
- `#define BOOT_COM_NET_ENABLE (1)`
Enable/disable the NET transport layer.
- `#define BOOT_COM_NET_TX_MAX_DATA (64)`
Configure number of bytes in the target->host data packet.
- `#define BOOT_COM_NET_RX_MAX_DATA (64)`
Configure number of bytes in the host->target data packet.
- `#define BOOT_COM_NET_PORT (1000)`
Configure the port that the TCP/IP server listens on.
- `#define BOOT_COM_NET_DHCP_ENABLE (1)`
Enable/disable DHCP client for automatically obtaining an IP address.
- `#define BOOT_COM_NET_IPADDR0 (192)`
Configure the 1st byte of the IP address.
- `#define BOOT_COM_NET_IPADDR1 (168)`
Configure the 2nd byte of the IP address.
- `#define BOOT_COM_NET_IPADDR2 (178)`
Configure the 3rd byte of the IP address.
- `#define BOOT_COM_NET_IPADDR3 (50)`
Configure the 4th byte of the IP address.
- `#define BOOT_COM_NET_NETMASK0 (255)`
Configure the 1st byte of the network mask.
- `#define BOOT_COM_NET_NETMASK1 (255)`
Configure the 2nd byte of the network mask.
- `#define BOOT_COM_NET_NETMASK2 (255)`
Configure the 3rd byte of the network mask.
- `#define BOOT_COM_NET_NETMASK3 (0)`
Configure the 4th byte of the network mask.
- `#define BOOT_COM_NET_GATEWAY0 (192)`
Configure the 1st byte of the gateway address.
- `#define BOOT_COM_NET_GATEWAY1 (168)`
Configure the 2nd byte of the gateway address.
- `#define BOOT_COM_NET_GATEWAY2 (178)`
Configure the 3rd byte of the gateway address.
- `#define BOOT_COM_NET_GATEWAY3 (1)`
Configure the 4th byte of the gateway address.
- `#define BOOT_COM_NET_DEFERRED_INIT_ENABLE (1)`
Enable/disable the deferred initialization mechanism. When enabled, the communication interface is only initialized when: (a) no valid user program is detected, or (b) when `CpuUserProgramStartHook()` returns `BLT_FALSE`. Your bootloader application can explicitly initialize the communication interface by calling `ComDeferredInit()`.
- `#define BOOT_COM_USB_ENABLE (1)`
Enable/disable USB transport layer.
- `#define BOOT_COM_USB_TX_MAX_DATA (63)`

- Configure number of bytes in the target->host data packet.*

 - #define `BOOT_COM_USB_RX_MAX_DATA` (63)
- Configure number of bytes in the host->target data packet.*

 - #define `BOOT_COM_CAN_ENABLE` (1)
- Enable/disable CAN transport layer.*

 - #define `BOOT_COM_CAN_BAUDRATE` (500000)
- Configure the desired CAN baudrate.*

 - #define `BOOT_COM_CAN_TX_MSG_ID` (0x7E1 /*| 0x80000000*/)
- Configure CAN message ID target->host.*

 - #define `BOOT_COM_CAN_TX_MAX_DATA` (8)
- Configure number of bytes in the target->host CAN message.*

 - #define `BOOT_COM_CAN_RX_MSG_ID` (0x667 /*| 0x80000000*/)
- Configure CAN message ID host->target.*

 - #define `BOOT_COM_CAN_RX_MAX_DATA` (8)
- Configure number of bytes in the host->target CAN message.*

 - #define `BOOT_COM_CAN_CHANNEL_INDEX` (0)
- Select the desired CAN peripheral as a zero based index.*

 - #define `BOOT_COM_RS232_ENABLE` (1)
- Enable/disable UART transport layer.*

 - #define `BOOT_COM_RS232_BAUDRATE` (57600)
- Configure the desired communication speed.*

 - #define `BOOT_COM_RS232_TX_MAX_DATA` (64)
- Configure number of bytes in the target->host data packet.*

 - #define `BOOT_COM_RS232_RX_MAX_DATA` (64)
- Configure number of bytes in the host->target data packet.*

 - #define `BOOT_COM_RS232_CHANNEL_INDEX` (2)
- Select the desired UART peripheral as a zero based index.*

 - #define `BOOT_BACKDOOR_HOOKS_ENABLE` (0)
- Enable/disable the backdoor override hook functions.*

 - #define `BOOT_NVM_HOOKS_ENABLE` (0)
- Enable/disable the NVM hook function for supporting additional memory devices.*

 - #define `BOOT_NVM_SIZE_KB` (2048)
- Configure the size of the default memory device (typically flash EEPROM).*

 - #define `BOOT_NVM_CHECKSUM_HOOKS_ENABLE` (0)
- Enable/disable hooks functions to override the user program checksum handling.*

 - #define `BOOT_FLASH_VECTOR_TABLE_CS_OFFSET` (0x1AC)
- This microcontroller has a larger vector table then the default STM32F7xx project as assumed in the bootloader's core. This means the user program has a different checksum location, because this one is added at the end of the user program's vector table.*

 - #define `BOOT_COP_HOOKS_ENABLE` (1)
- Enable/disable the hook functions for controlling the watchdog.*

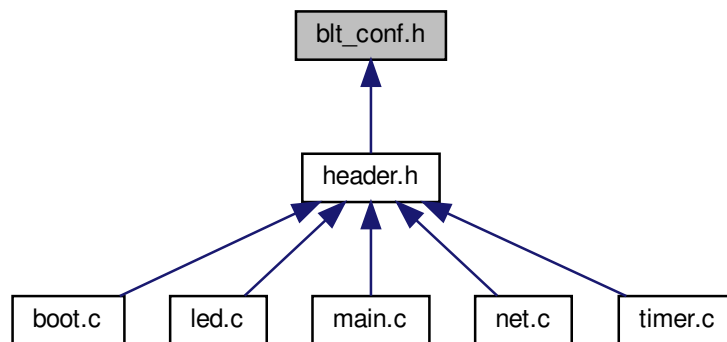
7.128.1 Detailed Description

Bootloader configuration header file.

7.129 blt_conf.h File Reference

Bootloader configuration header file.

This graph shows which files directly or indirectly include this file:



Macros

- #define `BOOT_CPU_XTAL_SPEED_KHZ` (8000)
Frequency of the external crystal oscillator.
- #define `BOOT_CPU_SYSTEM_SPEED_KHZ` (168000)
Desired system speed.
- #define `BOOT_CPU_BYTE_ORDER_MOTOROLA` (0)
Motorola or Intel style byte ordering.
- #define `BOOT_CPU_USER_PROGRAM_START_HOOK` (1)
Enable/disable hook function call right before user program start.
- #define `BOOT_COM_NET_ENABLE` (1)
Enable/disable the NET transport layer.
- #define `BOOT_COM_NET_TX_MAX_DATA` (64)
Configure number of bytes in the target->host data packet.
- #define `BOOT_COM_NET_RX_MAX_DATA` (64)
Configure number of bytes in the host->target data packet.
- #define `BOOT_COM_NET_PORT` (1000)
Configure the port that the TCP/IP server listens on.
- #define `BOOT_COM_NET_DHCP_ENABLE` (1)
Enable/disable DHCP client for automatically obtaining an IP address.
- #define `BOOT_COM_NET_IPADDR0` (192)
Configure the 1st byte of the IP address.
- #define `BOOT_COM_NET_IPADDR1` (168)
Configure the 2nd byte of the IP address.
- #define `BOOT_COM_NET_IPADDR2` (178)
Configure the 3rd byte of the IP address.
- #define `BOOT_COM_NET_IPADDR3` (50)

- Configure the 4th byte of the IP address.*

 - #define [BOOT_COM_NET_NETMASK0](#) (255)
- Configure the 1st byte of the network mask.*

 - #define [BOOT_COM_NET_NETMASK1](#) (255)
- Configure the 2nd byte of the network mask.*

 - #define [BOOT_COM_NET_NETMASK2](#) (255)
- Configure the 3rd byte of the network mask.*

 - #define [BOOT_COM_NET_NETMASK3](#) (0)
- Configure the 4th byte of the network mask.*

 - #define [BOOT_COM_NET_GATEWAY0](#) (192)
- Configure the 1st byte of the gateway address.*

 - #define [BOOT_COM_NET_GATEWAY1](#) (168)
- Configure the 2nd byte of the gateway address.*

 - #define [BOOT_COM_NET_GATEWAY2](#) (178)
- Configure the 3rd byte of the gateway address.*

 - #define [BOOT_COM_NET_GATEWAY3](#) (1)
- Configure the 4th byte of the gateway address.*

 - #define [BOOT_COM_NET_DEFERRED_INIT_ENABLE](#) (1)
- Enable/disable the deferred initialization mechanism. When enabled, the communication interface is only initialized when: (a) no valid user program is detected, or (b) when [CpuUserProgramStartHook\(\)](#) returns [BLT_FALSE](#). Your bootloader application can explicitly initialize the communication interface by calling [ComDeferredInit\(\)](#).*

 - #define [BOOT_COM_USB_ENABLE](#) (1)
- Enable/disable USB transport layer.*

 - #define [BOOT_COM_USB_TX_MAX_DATA](#) (63)
- Configure number of bytes in the target->host data packet.*

 - #define [BOOT_COM_USB_RX_MAX_DATA](#) (63)
- Configure number of bytes in the host->target data packet.*

 - #define [BOOT_COM_CAN_ENABLE](#) (1)
- Enable/disable CAN transport layer.*

 - #define [BOOT_COM_CAN_BAUDRATE](#) (500000)
- Configure the desired CAN baudrate.*

 - #define [BOOT_COM_CAN_TX_MSG_ID](#) (0x7E1 /*| 0x80000000*/)
- Configure CAN message ID target->host.*

 - #define [BOOT_COM_CAN_TX_MAX_DATA](#) (8)
- Configure number of bytes in the target->host CAN message.*

 - #define [BOOT_COM_CAN_RX_MSG_ID](#) (0x667 /*| 0x80000000*/)
- Configure CAN message ID host->target.*

 - #define [BOOT_COM_CAN_RX_MAX_DATA](#) (8)
- Configure number of bytes in the host->target CAN message.*

 - #define [BOOT_COM_CAN_CHANNEL_INDEX](#) (0)
- Select the desired CAN peripheral as a zero based index.*

 - #define [BOOT_COM_RS232_ENABLE](#) (1)
- Enable/disable UART transport layer.*

 - #define [BOOT_COM_RS232_BAUDRATE](#) (57600)
- Configure the desired communication speed.*

 - #define [BOOT_COM_RS232_TX_MAX_DATA](#) (64)
- Configure number of bytes in the target->host data packet.*

 - #define [BOOT_COM_RS232_RX_MAX_DATA](#) (64)
- Configure number of bytes in the host->target data packet.*

 - #define [BOOT_COM_RS232_CHANNEL_INDEX](#) (2)
- Select the desired UART peripheral as a zero based index.*

- #define `BOOT_BACKDOOR_HOOKS_ENABLE` (0)
Enable/disable the backdoor override hook functions.
- #define `BOOT_NVM_HOOKS_ENABLE` (0)
Enable/disable the NVM hook function for supporting additional memory devices.
- #define `BOOT_NVM_SIZE_KB` (2048)
Configure the size of the default memory device (typically flash EEPROM).
- #define `BOOT_NVM_CHECKSUM_HOOKS_ENABLE` (0)
Enable/disable hooks functions to override the user program checksum handling.
- #define `BOOT_FLASH_VECTOR_TABLE_CS_OFFSET` (0x1AC)
This microcontroller has a larger vector table then the default STM32F7xx project as assumed in the bootloader's core. This means the user program has a different checksum location, because this one is added at the end of the user program's vector table.
- #define `BOOT_COP_HOOKS_ENABLE` (1)
Enable/disable the hook functions for controlling the watchdog.

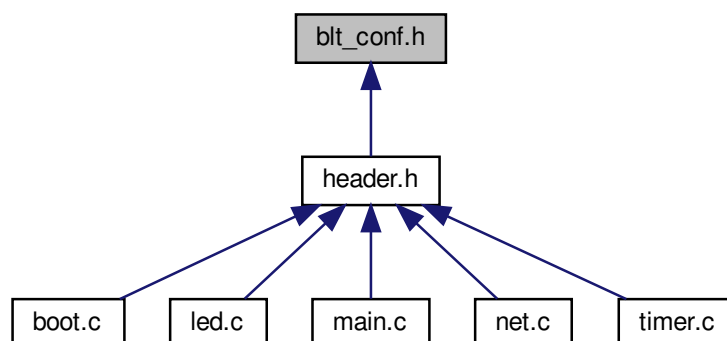
7.129.1 Detailed Description

Bootloader configuration header file.

7.130 blt_conf.h File Reference

Bootloader configuration header file.

This graph shows which files directly or indirectly include this file:



Macros

- #define `BOOT_CPU_XTAL_SPEED_KHZ` (8000)
Frequency of the external crystal oscillator.
- #define `BOOT_CPU_SYSTEM_SPEED_KHZ` (168000)
Desired system speed.
- #define `BOOT_CPU_BYTE_ORDER_MOTOROLA` (0)
Motorola or Intel style byte ordering.
- #define `BOOT_CPU_USER_PROGRAM_START_HOOK` (1)
Enable/disable hook function call right before user program start.
- #define `BOOT_COM_NET_ENABLE` (1)
Enable/disable the NET transport layer.
- #define `BOOT_COM_NET_TX_MAX_DATA` (64)
Configure number of bytes in the target->host data packet.
- #define `BOOT_COM_NET_RX_MAX_DATA` (64)
Configure number of bytes in the host->target data packet.
- #define `BOOT_COM_NET_PORT` (1000)
Configure the port that the TCP/IP server listens on.
- #define `BOOT_COM_NET_DHCP_ENABLE` (1)
Enable/disable DHCP client for automatically obtaining an IP address.
- #define `BOOT_COM_NET_IPADDR0` (192)
Configure the 1st byte of the IP address.
- #define `BOOT_COM_NET_IPADDR1` (168)
Configure the 2nd byte of the IP address.
- #define `BOOT_COM_NET_IPADDR2` (178)
Configure the 3rd byte of the IP address.
- #define `BOOT_COM_NET_IPADDR3` (50)
Configure the 4th byte of the IP address.
- #define `BOOT_COM_NET_NETMASK0` (255)
Configure the 1st byte of the network mask.
- #define `BOOT_COM_NET_NETMASK1` (255)
Configure the 2nd byte of the network mask.
- #define `BOOT_COM_NET_NETMASK2` (255)
Configure the 3rd byte of the network mask.
- #define `BOOT_COM_NET_NETMASK3` (0)
Configure the 4th byte of the network mask.
- #define `BOOT_COM_NET_GATEWAY0` (192)
Configure the 1st byte of the gateway address.
- #define `BOOT_COM_NET_GATEWAY1` (168)
Configure the 2nd byte of the gateway address.
- #define `BOOT_COM_NET_GATEWAY2` (178)
Configure the 3rd byte of the gateway address.
- #define `BOOT_COM_NET_GATEWAY3` (1)
Configure the 4th byte of the gateway address.
- #define `BOOT_COM_NET_DEFERRED_INIT_ENABLE` (1)
Enable/disable the deferred initialization mechanism. When enabled, the communication interface is only initialized when: (a) no valid user program is detected, or (b) when `CpuUserProgramStartHook()` returns `BLT_FALSE`. Your bootloader application can explicitly initialize the communication interface by calling `ComDeferredInit()`.
- #define `BOOT_COM_USB_ENABLE` (1)
Enable/disable USB transport layer.
- #define `BOOT_COM_USB_TX_MAX_DATA` (63)

- Configure number of bytes in the target->host data packet.*

 - #define `BOOT_COM_USB_RX_MAX_DATA` (63)
- Configure number of bytes in the host->target data packet.*

 - #define `BOOT_COM_CAN_ENABLE` (1)
- Enable/disable CAN transport layer.*

 - #define `BOOT_COM_CAN_BAUDRATE` (500000)
- Configure the desired CAN baudrate.*

 - #define `BOOT_COM_CAN_TX_MSG_ID` (0x7E1 /*| 0x80000000*/)
- Configure CAN message ID target->host.*

 - #define `BOOT_COM_CAN_TX_MAX_DATA` (8)
- Configure number of bytes in the target->host CAN message.*

 - #define `BOOT_COM_CAN_RX_MSG_ID` (0x667 /*| 0x80000000*/)
- Configure CAN message ID host->target.*

 - #define `BOOT_COM_CAN_RX_MAX_DATA` (8)
- Configure number of bytes in the host->target CAN message.*

 - #define `BOOT_COM_CAN_CHANNEL_INDEX` (0)
- Select the desired CAN peripheral as a zero based index.*

 - #define `BOOT_COM_RS232_ENABLE` (1)
- Enable/disable UART transport layer.*

 - #define `BOOT_COM_RS232_BAUDRATE` (57600)
- Configure the desired communication speed.*

 - #define `BOOT_COM_RS232_TX_MAX_DATA` (64)
- Configure number of bytes in the target->host data packet.*

 - #define `BOOT_COM_RS232_RX_MAX_DATA` (64)
- Configure number of bytes in the host->target data packet.*

 - #define `BOOT_COM_RS232_CHANNEL_INDEX` (2)
- Select the desired UART peripheral as a zero based index.*

 - #define `BOOT_BACKDOOR_HOOKS_ENABLE` (0)
- Enable/disable the backdoor override hook functions.*

 - #define `BOOT_NVM_HOOKS_ENABLE` (0)
- Enable/disable the NVM hook function for supporting additional memory devices.*

 - #define `BOOT_NVM_SIZE_KB` (2048)
- Configure the size of the default memory device (typically flash EEPROM).*

 - #define `BOOT_NVM_CHECKSUM_HOOKS_ENABLE` (0)
- Enable/disable hooks functions to override the user program checksum handling.*

 - #define `BOOT_FLASH_VECTOR_TABLE_CS_OFFSET` (0x1AC)
- This microcontroller has a larger vector table then the default STM32F7xx project as assumed in the bootloader's core. This means the user program has a different checksum location, because this one is added at the end of the user program's vector table.*

 - #define `BOOT_COP_HOOKS_ENABLE` (1)
- Enable/disable the hook functions for controlling the watchdog.*

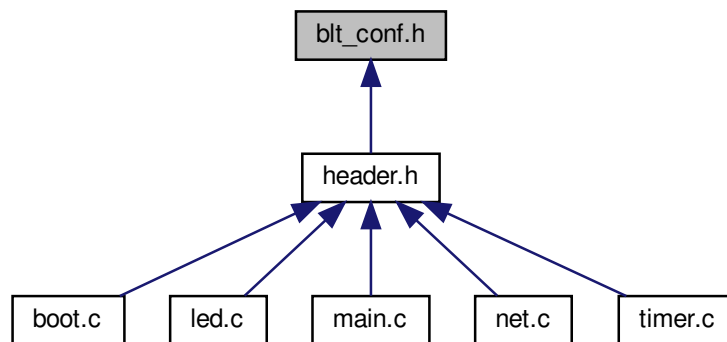
7.130.1 Detailed Description

Bootloader configuration header file.

7.131 blt_conf.h File Reference

Bootloader configuration header file.

This graph shows which files directly or indirectly include this file:



Macros

- #define [BOOT_CPU_XTAL_SPEED_KHZ](#) (8000)
Frequency of the external crystal oscillator.
- #define [BOOT_CPU_SYSTEM_SPEED_KHZ](#) (168000)
Desired system speed.
- #define [BOOT_CPU_BYTE_ORDER_MOTOROLA](#) (0)
Motorola or Intel style byte ordering.
- #define [BOOT_CPU_USER_PROGRAM_START_HOOK](#) (1)
Enable/disable hook function call right before user program start.
- #define [BOOT_COM_NET_ENABLE](#) (1)
Enable/disable the NET transport layer.
- #define [BOOT_COM_NET_TX_MAX_DATA](#) (64)
Configure number of bytes in the target->host data packet.
- #define [BOOT_COM_NET_RX_MAX_DATA](#) (64)
Configure number of bytes in the host->target data packet.
- #define [BOOT_COM_NET_PORT](#) (1000)
Configure the port that the TCP/IP server listens on.
- #define [BOOT_COM_NET_DHCP_ENABLE](#) (1)
Enable/disable DHCP client for automatically obtaining an IP address.
- #define [BOOT_COM_NET_IPADDR0](#) (192)
Configure the 1st byte of the IP address.
- #define [BOOT_COM_NET_IPADDR1](#) (168)
Configure the 2nd byte of the IP address.
- #define [BOOT_COM_NET_IPADDR2](#) (178)
Configure the 3rd byte of the IP address.
- #define [BOOT_COM_NET_IPADDR3](#) (50)

- Configure the 4th byte of the IP address.*

 - #define `BOOT_COM_NET_NETMASK0` (255)
- Configure the 1st byte of the network mask.*

 - #define `BOOT_COM_NET_NETMASK1` (255)
- Configure the 2nd byte of the network mask.*

 - #define `BOOT_COM_NET_NETMASK2` (255)
- Configure the 3rd byte of the network mask.*

 - #define `BOOT_COM_NET_NETMASK3` (0)
- Configure the 4th byte of the network mask.*

 - #define `BOOT_COM_NET_GATEWAY0` (192)
- Configure the 1st byte of the gateway address.*

 - #define `BOOT_COM_NET_GATEWAY1` (168)
- Configure the 2nd byte of the gateway address.*

 - #define `BOOT_COM_NET_GATEWAY2` (178)
- Configure the 3rd byte of the gateway address.*

 - #define `BOOT_COM_NET_GATEWAY3` (1)
- Configure the 4th byte of the gateway address.*

 - #define `BOOT_COM_NET_DEFERRED_INIT_ENABLE` (1)

Enable/disable the deferred initialization mechanism. When enabled, the communication interface is only initialized when: (a) no valid user program is detected, or (b) when `CpuUserProgramStartHook()` returns `BLT_FALSE`. Your bootloader application can explicitly initialize the communication interface by calling `ComDeferredInit()`.
- Enable/disable USB transport layer.*

 - #define `BOOT_COM_USB_ENABLE` (1)
- Enable/disable USB transport layer.*

 - #define `BOOT_COM_USB_TX_MAX_DATA` (63)

Configure number of bytes in the target->host data packet.
- Configure number of bytes in the host->target data packet.*

 - #define `BOOT_COM_USB_RX_MAX_DATA` (63)
- Enable/disable CAN transport layer.*

 - #define `BOOT_COM_CAN_ENABLE` (1)
- Enable/disable CAN transport layer.*

 - #define `BOOT_COM_CAN_BAUDRATE` (500000)

Configure the desired CAN baudrate.
- Configure CAN message ID target->host.*

 - #define `BOOT_COM_CAN_TX_MSG_ID` (0x7E1 /*| 0x80000000*/)
- Configure number of bytes in the target->host CAN message.*

 - #define `BOOT_COM_CAN_TX_MAX_DATA` (8)
- Configure CAN message ID host->target.*

 - #define `BOOT_COM_CAN_RX_MSG_ID` (0x667 /*| 0x80000000*/)
- Configure number of bytes in the host->target CAN message.*

 - #define `BOOT_COM_CAN_RX_MAX_DATA` (8)
- Select the desired CAN peripheral as a zero based index.*

 - #define `BOOT_COM_CAN_CHANNEL_INDEX` (0)
- Enable/disable UART transport layer.*

 - #define `BOOT_COM_RS232_ENABLE` (1)
- Configure the desired communication speed.*

 - #define `BOOT_COM_RS232_BAUDRATE` (57600)
- Configure number of bytes in the target->host data packet.*

 - #define `BOOT_COM_RS232_TX_MAX_DATA` (64)
- Configure number of bytes in the host->target data packet.*

 - #define `BOOT_COM_RS232_RX_MAX_DATA` (64)
- Select the desired UART peripheral as a zero based index.*

 - #define `BOOT_COM_RS232_CHANNEL_INDEX` (2)

- #define `BOOT_BACKDOOR_HOOKS_ENABLE` (0)
Enable/disable the backdoor override hook functions.
- #define `BOOT_NVM_HOOKS_ENABLE` (0)
Enable/disable the NVM hook function for supporting additional memory devices.
- #define `BOOT_NVM_SIZE_KB` (2048)
Configure the size of the default memory device (typically flash EEPROM).
- #define `BOOT_NVM_CHECKSUM_HOOKS_ENABLE` (0)
Enable/disable hooks functions to override the user program checksum handling.
- #define `BOOT_FLASH_VECTOR_TABLE_CS_OFFSET` (0x1AC)
This microcontroller has a larger vector table then the default STM32F7xx project as assumed in the bootloader's core. This means the user program has a different checksum location, because this one is added at the end of the user program's vector table.
- #define `BOOT_COP_HOOKS_ENABLE` (1)
Enable/disable the hook functions for controlling the watchdog.

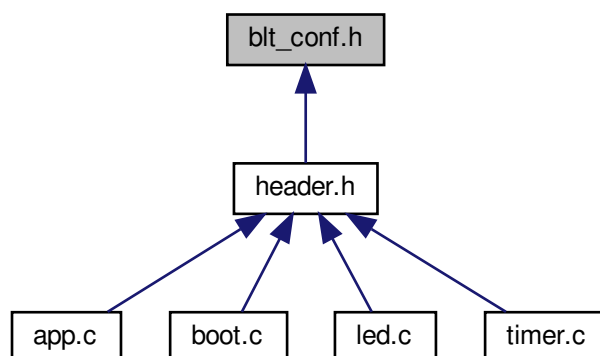
7.131.1 Detailed Description

Bootloader configuration header file.

7.132 blt_conf.h File Reference

Bootloader configuration header file.

This graph shows which files directly or indirectly include this file:



Macros

- #define `BOOT_CPU_XTAL_SPEED_KHZ` (8000)
Frequency of the external crystal oscillator.
- #define `BOOT_CPU_SYSTEM_SPEED_KHZ` (168000)
Desired system speed.
- #define `BOOT_CPU_BYTE_ORDER_MOTOROLA` (0)
Motorola or Intel style byte ordering.
- #define `BOOT_CPU_USER_PROGRAM_START_HOOK` (1)
Enable/disable hook function call right before user program start.
- #define `BOOT_COM_CAN_ENABLE` (1)
Enable/disable CAN transport layer.
- #define `BOOT_COM_CAN_BAUDRATE` (500000)
Configure the desired CAN baudrate.
- #define `BOOT_COM_CAN_TX_MSG_ID` (0x7E1 /*| 0x80000000*/) *Configure CAN message ID target->host.*
- #define `BOOT_COM_CAN_TX_MAX_DATA` (8)
Configure number of bytes in the target->host CAN message.
- #define `BOOT_COM_CAN_RX_MSG_ID` (0x667 /*| 0x80000000*/) *Configure CAN message ID host->target.*
- #define `BOOT_COM_CAN_RX_MAX_DATA` (8)
Configure number of bytes in the host->target CAN message.
- #define `BOOT_COM_CAN_CHANNEL_INDEX` (0)
Select the desired CAN peripheral as a zero based index.
- #define `BOOT_COM_RS232_ENABLE` (1)
Enable/disable UART transport layer.
- #define `BOOT_COM_RS232_BAUDRATE` (57600)
Configure the desired communication speed.
- #define `BOOT_COM_RS232_TX_MAX_DATA` (64)
Configure number of bytes in the target->host data packet.
- #define `BOOT_COM_RS232_RX_MAX_DATA` (64)
Configure number of bytes in the host->target data packet.
- #define `BOOT_COM_RS232_CHANNEL_INDEX` (1)
Select the desired UART peripheral as a zero based index.
- #define `BOOT_COM_USB_ENABLE` (1)
Enable/disable USB transport layer.
- #define `BOOT_COM_USB_TX_MAX_DATA` (63)
Configure number of bytes in the target->host data packet.
- #define `BOOT_COM_USB_RX_MAX_DATA` (63)
Configure number of bytes in the host->target data packet.
- #define `BOOT_FILE_SYS_ENABLE` (1)
Enable/disable support for firmware updates from a locally attached storage.
- #define `BOOT_FILE_LOGGING_ENABLE` (1)
Enable/disable logging messages during firmware updates.
- #define `BOOT_FILE_ERROR_HOOK_ENABLE` (1)
Enable/disable a hook function that is called upon detection of an error.
- #define `BOOT_FILE_STARTED_HOOK_ENABLE` (1)
Enable/disable a hook function that is called at the start of the update.
- #define `BOOT_FILE_COMPLETED_HOOK_ENABLE` (1)
Enable/disable a hook function that is called at the end of the update.
- #define `BOOT_BACKDOOR_HOOKS_ENABLE` (0)

- *Enable/disable the backdoor override hook functions.*
 • #define `BOOT_NVM_HOOKS_ENABLE` (0)
Enable/disable the NVM hook function for supporting additional memory devices.
- #define `BOOT_NVM_SIZE_KB` (1024)
Configure the size of the default memory device (typically flash EEPROM).
- #define `BOOT_NVM_CHECKSUM_HOOKS_ENABLE` (0)
Enable/disable hooks functions to override the user program checksum handling.
- #define `BOOT_COP_HOOKS_ENABLE` (1)
Enable/disable the hook functions for controlling the watchdog.

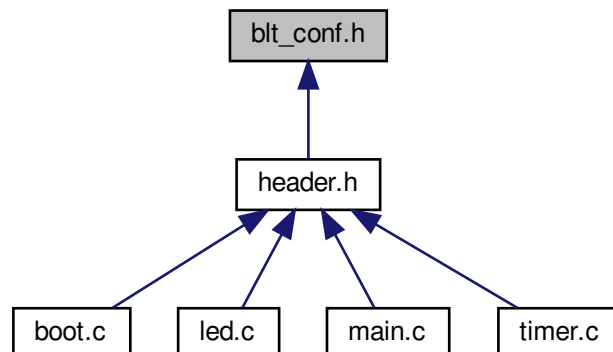
7.132.1 Detailed Description

Bootloader configuration header file.

7.133 blt_conf.h File Reference

Bootloader configuration header file.

This graph shows which files directly or indirectly include this file:



Macros

- #define `BOOT_CPU_XTAL_SPEED_KHZ` (8000)
Frequency of the external crystal oscillator.
- #define `BOOT_CPU_SYSTEM_SPEED_KHZ` (168000)
Desired system speed.
- #define `BOOT_CPU_BYTE_ORDER_MOTOROLA` (0)
Motorola or Intel style byte ordering.
- #define `BOOT_CPU_USER_PROGRAM_START_HOOK` (1)
Enable/disable hook function call right before user program start.

- `#define BOOT_COM_CAN_ENABLE (1)`
Enable/disable CAN transport layer.
- `#define BOOT_COM_CAN_BAUDRATE (500000)`
Configure the desired CAN baudrate.
- `#define BOOT_COM_CAN_TX_MSG_ID (0x7E1 /*| 0x80000000*/)`
Configure CAN message ID target->host.
- `#define BOOT_COM_CAN_TX_MAX_DATA (8)`
Configure number of bytes in the target->host CAN message.
- `#define BOOT_COM_CAN_RX_MSG_ID (0x667 /*| 0x80000000*/)`
Configure CAN message ID host->target.
- `#define BOOT_COM_CAN_RX_MAX_DATA (8)`
Configure number of bytes in the host->target CAN message.
- `#define BOOT_COM_CAN_CHANNEL_INDEX (0)`
Select the desired CAN peripheral as a zero based index.
- `#define BOOT_COM_RS232_ENABLE (1)`
Enable/disable UART transport layer.
- `#define BOOT_COM_RS232_BAUDRATE (57600)`
Configure the desired communication speed.
- `#define BOOT_COM_RS232_TX_MAX_DATA (64)`
Configure number of bytes in the target->host data packet.
- `#define BOOT_COM_RS232_RX_MAX_DATA (64)`
Configure number of bytes in the host->target data packet.
- `#define BOOT_COM_RS232_CHANNEL_INDEX (1)`
Select the desired UART peripheral as a zero based index.
- `#define BOOT_COM_USB_ENABLE (1)`
Enable/disable USB transport layer.
- `#define BOOT_COM_USB_TX_MAX_DATA (63)`
Configure number of bytes in the target->host data packet.
- `#define BOOT_COM_USB_RX_MAX_DATA (63)`
Configure number of bytes in the host->target data packet.
- `#define BOOT_FILE_SYS_ENABLE (1)`
Enable/disable support for firmware updates from a locally attached storage.
- `#define BOOT_FILE_LOGGING_ENABLE (1)`
Enable/disable logging messages during firmware updates.
- `#define BOOT_FILE_ERROR_HOOK_ENABLE (1)`
Enable/disable a hook function that is called upon detection of an error.
- `#define BOOT_FILE_STARTED_HOOK_ENABLE (1)`
Enable/disable a hook function that is called at the start of the update.
- `#define BOOT_FILE_COMPLETED_HOOK_ENABLE (1)`
Enable/disable a hook function that is called at the end of the update.
- `#define BOOT_BACKDOOR_HOOKS_ENABLE (0)`
Enable/disable the backdoor override hook functions.
- `#define BOOT_NVM_HOOKS_ENABLE (0)`
Enable/disable the NVM hook function for supporting additional memory devices.
- `#define BOOT_NVM_SIZE_KB (1024)`
Configure the size of the default memory device (typically flash EEPROM).
- `#define BOOT_NVM_CHECKSUM_HOOKS_ENABLE (0)`
Enable/disable hooks functions to override the user program checksum handling.
- `#define BOOT_COP_HOOKS_ENABLE (1)`
Enable/disable the hook functions for controlling the watchdog.

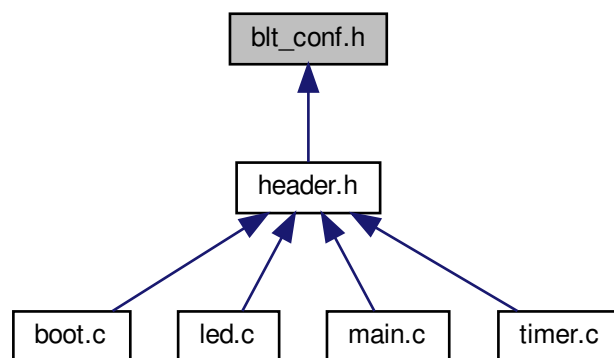
7.133.1 Detailed Description

Bootloader configuration header file.

7.134 blt_conf.h File Reference

Bootloader configuration header file.

This graph shows which files directly or indirectly include this file:



Macros

- `#define BOOT_CPU_XTAL_SPEED_KHZ (8000)`
Frequency of the external crystal oscillator.
- `#define BOOT_CPU_SYSTEM_SPEED_KHZ (168000)`
Desired system speed.
- `#define BOOT_CPU_BYTE_ORDER_MOTOROLA (0)`
Motorola or Intel style byte ordering.
- `#define BOOT_CPU_USER_PROGRAM_START_HOOK (1)`
Enable/disable hook function call right before user program start.
- `#define BOOT_COM_CAN_ENABLE (1)`
Enable/disable CAN transport layer.
- `#define BOOT_COM_CAN_BAUDRATE (500000)`
Configure the desired CAN baudrate.
- `#define BOOT_COM_CAN_TX_MSG_ID (0x7E1 /*| 0x80000000*/)`
Configure CAN message ID target->host.
- `#define BOOT_COM_CAN_TX_MAX_DATA (8)`
Configure number of bytes in the target->host CAN message.
- `#define BOOT_COM_CAN_RX_MSG_ID (0x667 /*| 0x80000000*/)`
Configure CAN message ID host->target.
- `#define BOOT_COM_CAN_RX_MAX_DATA (8)`

- Configure number of bytes in the host->target CAN message.*

 - #define `BOOT_COM_CAN_CHANNEL_INDEX` (0)
 - Select the desired CAN peripheral as a zero based index.*
- #define `BOOT_COM_RS232_ENABLE` (1)
 - Enable/disable UART transport layer.*
- #define `BOOT_COM_RS232_BAUDRATE` (57600)
 - Configure the desired communication speed.*
- #define `BOOT_COM_RS232_TX_MAX_DATA` (64)
 - Configure number of bytes in the target->host data packet.*
- #define `BOOT_COM_RS232_RX_MAX_DATA` (64)
 - Configure number of bytes in the host->target data packet.*
- #define `BOOT_COM_RS232_CHANNEL_INDEX` (1)
 - Select the desired UART peripheral as a zero based index.*
- #define `BOOT_COM_USB_ENABLE` (1)
 - Enable/disable USB transport layer.*
- #define `BOOT_COM_USB_TX_MAX_DATA` (63)
 - Configure number of bytes in the target->host data packet.*
- #define `BOOT_COM_USB_RX_MAX_DATA` (63)
 - Configure number of bytes in the host->target data packet.*
- #define `BOOT_FILE_SYS_ENABLE` (1)
 - Enable/disable support for firmware updates from a locally attached storage.*
- #define `BOOT_FILE_LOGGING_ENABLE` (1)
 - Enable/disable logging messages during firmware updates.*
- #define `BOOT_FILE_ERROR_HOOK_ENABLE` (1)
 - Enable/disable a hook function that is called upon detection of an error.*
- #define `BOOT_FILE_STARTED_HOOK_ENABLE` (1)
 - Enable/disable a hook function that is called at the start of the update.*
- #define `BOOT_FILE_COMPLETED_HOOK_ENABLE` (1)
 - Enable/disable a hook function that is called at the end of the update.*
- #define `BOOT_BACKDOOR_HOOKS_ENABLE` (0)
 - Enable/disable the backdoor override hook functions.*
- #define `BOOT_NVM_HOOKS_ENABLE` (0)
 - Enable/disable the NVM hook function for supporting additional memory devices.*
- #define `BOOT_NVM_SIZE_KB` (1024)
 - Configure the size of the default memory device (typically flash EEPROM).*
- #define `BOOT_NVM_CHECKSUM_HOOKS_ENABLE` (0)
 - Enable/disable hooks functions to override the user program checksum handling.*
- #define `BOOT_COP_HOOKS_ENABLE` (1)
 - Enable/disable the hook functions for controlling the watchdog.*

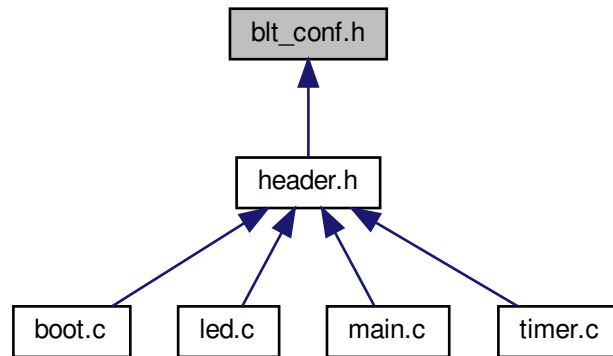
7.134.1 Detailed Description

Bootloader configuration header file.

7.135 blt_conf.h File Reference

Bootloader configuration header file.

This graph shows which files directly or indirectly include this file:



Macros

- #define `BOOT_CPU_XTAL_SPEED_KHZ` (8000)
Frequency of the external crystal oscillator.
- #define `BOOT_CPU_SYSTEM_SPEED_KHZ` (168000)
Desired system speed.
- #define `BOOT_CPU_BYTE_ORDER_MOTOROLA` (0)
Motorola or Intel style byte ordering.
- #define `BOOT_CPU_USER_PROGRAM_START_HOOK` (1)
Enable/disable hook function call right before user program start.
- #define `BOOT_COM_CAN_ENABLE` (1)
Enable/disable CAN transport layer.
- #define `BOOT_COM_CAN_BAUDRATE` (500000)
Configure the desired CAN baudrate.
- #define `BOOT_COM_CAN_TX_MSG_ID` (0x7E1 /*| 0x80000000*/) *Configure CAN message ID target->host.*
- #define `BOOT_COM_CAN_TX_MAX_DATA` (8)
Configure number of bytes in the target->host CAN message.
- #define `BOOT_COM_CAN_RX_MSG_ID` (0x667 /*| 0x80000000*/) *Configure CAN message ID host->target.*
- #define `BOOT_COM_CAN_RX_MAX_DATA` (8)
Configure number of bytes in the host->target CAN message.
- #define `BOOT_COM_CAN_CHANNEL_INDEX` (0)
Select the desired CAN peripheral as a zero based index.
- #define `BOOT_COM_RS232_ENABLE` (1)
Enable/disable UART transport layer.
- #define `BOOT_COM_RS232_BAUDRATE` (57600)

- Configure the desired communication speed.*

 - #define `BOOT_COM_RS232_TX_MAX_DATA` (64)

Configure number of bytes in the target->host data packet.
- #define `BOOT_COM_RS232_RX_MAX_DATA` (64)

Configure number of bytes in the host->target data packet.
- #define `BOOT_COM_RS232_CHANNEL_INDEX` (1)

Select the desired UART peripheral as a zero based index.
- #define `BOOT_COM_USB_ENABLE` (1)

Enable/disable USB transport layer.
- #define `BOOT_COM_USB_TX_MAX_DATA` (63)

Configure number of bytes in the target->host data packet.
- #define `BOOT_COM_USB_RX_MAX_DATA` (63)

Configure number of bytes in the host->target data packet.
- #define `BOOT_FILE_SYS_ENABLE` (1)

Enable/disable support for firmware updates from a locally attached storage.
- #define `BOOT_FILE_LOGGING_ENABLE` (1)

Enable/disable logging messages during firmware updates.
- #define `BOOT_FILE_ERROR_HOOK_ENABLE` (1)

Enable/disable a hook function that is called upon detection of an error.
- #define `BOOT_FILE_STARTED_HOOK_ENABLE` (1)

Enable/disable a hook function that is called at the start of the update.
- #define `BOOT_FILE_COMPLETED_HOOK_ENABLE` (1)

Enable/disable a hook function that is called at the end of the update.
- #define `BOOT_BACKDOOR_HOOKS_ENABLE` (0)

Enable/disable the backdoor override hook functions.
- #define `BOOT_NVM_HOOKS_ENABLE` (0)

Enable/disable the NVM hook function for supporting additional memory devices.
- #define `BOOT_NVM_SIZE_KB` (1024)

Configure the size of the default memory device (typically flash EEPROM).
- #define `BOOT_NVM_CHECKSUM_HOOKS_ENABLE` (0)

Enable/disable hooks functions to override the user program checksum handling.
- #define `BOOT_COP_HOOKS_ENABLE` (1)

Enable/disable the hook functions for controlling the watchdog.

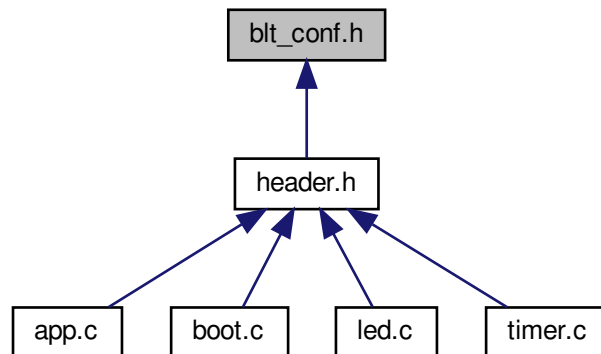
7.135.1 Detailed Description

Bootloader configuration header file.

7.136 blt_conf.h File Reference

Bootloader configuration header file.

This graph shows which files directly or indirectly include this file:



Macros

- `#define BOOT_CPU_XTAL_SPEED_KHZ (8000)`
Frequency of the external crystal oscillator.
- `#define BOOT_CPU_SYSTEM_SPEED_KHZ (80000)`
Desired system speed.
- `#define BOOT_CPU_BYTE_ORDER_MOTOROLA (0)`
Motorola or Intel style byte ordering.
- `#define BOOT_CPU_USER_PROGRAM_START_HOOK (1)`
Enable/disable hook function call right before user program start.
- `#define BOOT_COM_CAN_ENABLE (1)`
Enable/disable CAN transport layer.
- `#define BOOT_COM_CAN_BAUDRATE (500000)`
Configure the desired CAN baudrate.
- `#define BOOT_COM_CAN_TX_MSG_ID (0x7E1 /*| 0x80000000*/)`
Configure CAN message ID target->host.
- `#define BOOT_COM_CAN_TX_MAX_DATA (8)`
Configure number of bytes in the target->host CAN message.
- `#define BOOT_COM_CAN_RX_MSG_ID (0x667 /*| 0x80000000*/)`
Configure CAN message ID host->target.
- `#define BOOT_COM_CAN_RX_MAX_DATA (8)`
Configure number of bytes in the host->target CAN message.
- `#define BOOT_COM_CAN_CHANNEL_INDEX (0)`
Select the desired CAN peripheral as a zero based index.
- `#define BOOT_COM_RS232_ENABLE (1)`
Enable/disable UART transport layer.
- `#define BOOT_COM_RS232_BAUDRATE (57600)`
Configure the desired communication speed.
- `#define BOOT_COM_RS232_TX_MAX_DATA (64)`
Configure number of bytes in the target->host data packet.
- `#define BOOT_COM_RS232_RX_MAX_DATA (64)`

- Configure number of bytes in the host->target data packet.*

 - #define `BOOT_COM_RS232_CHANNEL_INDEX` (1)

Select the desired UART peripheral as a zero based index.
- #define `BOOT_BACKDOOR_HOOKS_ENABLE` (0)

Enable/disable the backdoor override hook functions.
- #define `BOOT_NVM_HOOKS_ENABLE` (0)

Enable/disable the NVM hook function for supporting additional memory devices.
- #define `BOOT_NVM_SIZE_KB` (1024)

Configure the size of the default memory device (typically flash EEPROM).
- #define `BOOT_NVM_CHECKSUM_HOOKS_ENABLE` (0)

Enable/disable hooks functions to override the user program checksum handling.
- #define `BOOT_COP_HOOKS_ENABLE` (1)

Enable/disable the hook functions for controlling the watchdog.

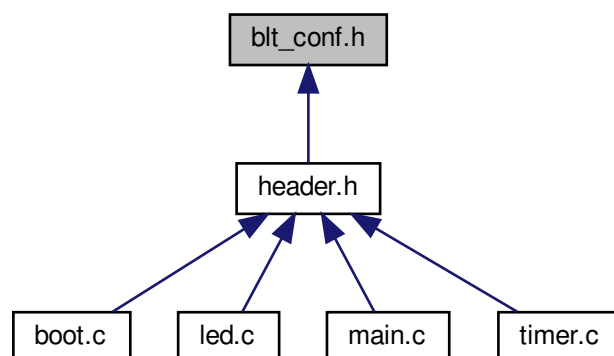
7.136.1 Detailed Description

Bootloader configuration header file.

7.137 blt_conf.h File Reference

Bootloader configuration header file.

This graph shows which files directly or indirectly include this file:



Macros

- #define `BOOT_CPU_XTAL_SPEED_KHZ` (8000)
Frequency of the external crystal oscillator.
- #define `BOOT_CPU_SYSTEM_SPEED_KHZ` (80000)
Desired system speed.
- #define `BOOT_CPU_BYTE_ORDER_MOTOROLA` (0)
Motorola or Intel style byte ordering.
- #define `BOOT_CPU_USER_PROGRAM_START_HOOK` (1)
Enable/disable hook function call right before user program start.
- #define `BOOT_COM_CAN_ENABLE` (1)
Enable/disable CAN transport layer.
- #define `BOOT_COM_CAN_BAUDRATE` (500000)
Configure the desired CAN baudrate.
- #define `BOOT_COM_CAN_TX_MSG_ID` (0x7E1 /*| 0x80000000*/) *Configure CAN message ID target->host.*
- #define `BOOT_COM_CAN_TX_MAX_DATA` (8)
Configure number of bytes in the target->host CAN message.
- #define `BOOT_COM_CAN_RX_MSG_ID` (0x667 /*| 0x80000000*/) *Configure CAN message ID host->target.*
- #define `BOOT_COM_CAN_RX_MAX_DATA` (8)
Configure number of bytes in the host->target CAN message.
- #define `BOOT_COM_CAN_CHANNEL_INDEX` (0)
Select the desired CAN peripheral as a zero based index.
- #define `BOOT_COM_RS232_ENABLE` (1)
Enable/disable UART transport layer.
- #define `BOOT_COM_RS232_BAUDRATE` (57600)
Configure the desired communication speed.
- #define `BOOT_COM_RS232_TX_MAX_DATA` (64)
Configure number of bytes in the target->host data packet.
- #define `BOOT_COM_RS232_RX_MAX_DATA` (64)
Configure number of bytes in the host->target data packet.
- #define `BOOT_COM_RS232_CHANNEL_INDEX` (1)
Select the desired UART peripheral as a zero based index.
- #define `BOOT_BACKDOOR_HOOKS_ENABLE` (0)
Enable/disable the backdoor override hook functions.
- #define `BOOT_NVM_HOOKS_ENABLE` (0)
Enable/disable the NVM hook function for supporting additional memory devices.
- #define `BOOT_NVM_SIZE_KB` (1024)
Configure the size of the default memory device (typically flash EEPROM).
- #define `BOOT_NVM_CHECKSUM_HOOKS_ENABLE` (0)
Enable/disable hooks functions to override the user program checksum handling.
- #define `BOOT_COP_HOOKS_ENABLE` (1)
Enable/disable the hook functions for controlling the watchdog.

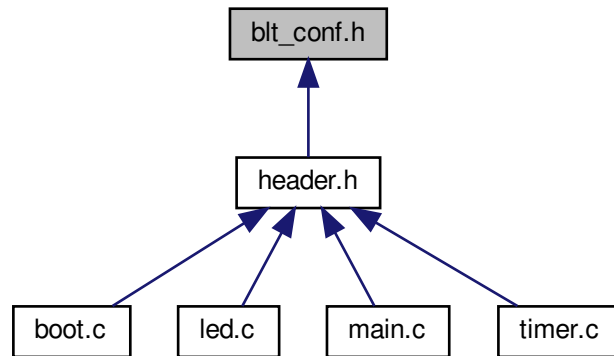
7.137.1 Detailed Description

Bootloader configuration header file.

7.138 blt_conf.h File Reference

Bootloader configuration header file.

This graph shows which files directly or indirectly include this file:



Macros

- `#define BOOT_CPU_XTAL_SPEED_KHZ (8000)`
Frequency of the external crystal oscillator.
- `#define BOOT_CPU_SYSTEM_SPEED_KHZ (80000)`
Desired system speed.
- `#define BOOT_CPU_BYTE_ORDER_MOTOROLA (0)`
Motorola or Intel style byte ordering.
- `#define BOOT_CPU_USER_PROGRAM_START_HOOK (1)`
Enable/disable hook function call right before user program start.
- `#define BOOT_COM_CAN_ENABLE (1)`
Enable/disable CAN transport layer.
- `#define BOOT_COM_CAN_BAUDRATE (500000)`
Configure the desired CAN baudrate.
- `#define BOOT_COM_CAN_TX_MSG_ID (0x7E1 /*| 0x80000000*/)`
Configure CAN message ID target->host.
- `#define BOOT_COM_CAN_TX_MAX_DATA (8)`
Configure number of bytes in the target->host CAN message.
- `#define BOOT_COM_CAN_RX_MSG_ID (0x667 /*| 0x80000000*/)`
Configure CAN message ID host->target.
- `#define BOOT_COM_CAN_RX_MAX_DATA (8)`
Configure number of bytes in the host->target CAN message.
- `#define BOOT_COM_CAN_CHANNEL_INDEX (0)`
Select the desired CAN peripheral as a zero based index.
- `#define BOOT_COM_RS232_ENABLE (1)`
Enable/disable UART transport layer.
- `#define BOOT_COM_RS232_BAUDRATE (57600)`

- Configure the desired communication speed.*

 - #define `BOOT_COM_RS232_TX_MAX_DATA` (64)

Configure number of bytes in the target->host data packet.
- #define `BOOT_COM_RS232_RX_MAX_DATA` (64)

Configure number of bytes in the host->target data packet.
- #define `BOOT_COM_RS232_CHANNEL_INDEX` (1)

Select the desired UART peripheral as a zero based index.
- #define `BOOT_BACKDOOR_HOOKS_ENABLE` (0)

Enable/disable the backdoor override hook functions.
- #define `BOOT_NVM_HOOKS_ENABLE` (0)

Enable/disable the NVM hook function for supporting additional memory devices.
- #define `BOOT_NVM_SIZE_KB` (1024)

Configure the size of the default memory device (typically flash EEPROM).
- #define `BOOT_NVM_CHECKSUM_HOOKS_ENABLE` (0)

Enable/disable hooks functions to override the user program checksum handling.
- #define `BOOT_COP_HOOKS_ENABLE` (1)

Enable/disable the hook functions for controlling the watchdog.

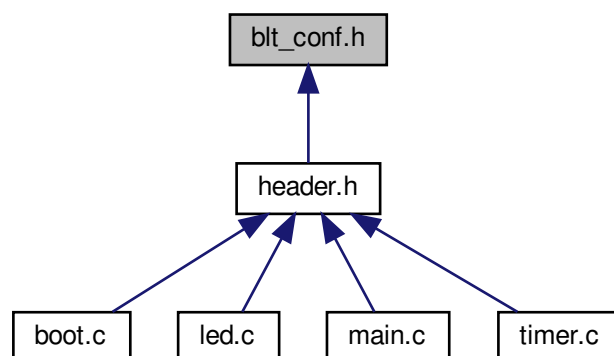
7.138.1 Detailed Description

Bootloader configuration header file.

7.139 blt_conf.h File Reference

Bootloader configuration header file.

This graph shows which files directly or indirectly include this file:



Macros

- #define `BOOT_CPU_XTAL_SPEED_KHZ` (8000)
Frequency of the external crystal oscillator.
- #define `BOOT_CPU_SYSTEM_SPEED_KHZ` (80000)
Desired system speed.
- #define `BOOT_CPU_BYTE_ORDER_MOTOROLA` (0)
Motorola or Intel style byte ordering.
- #define `BOOT_CPU_USER_PROGRAM_START_HOOK` (1)
Enable/disable hook function call right before user program start.
- #define `BOOT_COM_CAN_ENABLE` (1)
Enable/disable CAN transport layer.
- #define `BOOT_COM_CAN_BAUDRATE` (500000)
Configure the desired CAN baudrate.
- #define `BOOT_COM_CAN_TX_MSG_ID` (0x7E1 /*| 0x80000000*/) *Configure CAN message ID target->host.*
- #define `BOOT_COM_CAN_TX_MAX_DATA` (8)
Configure number of bytes in the target->host CAN message.
- #define `BOOT_COM_CAN_RX_MSG_ID` (0x667 /*| 0x80000000*/) *Configure CAN message ID host->target.*
- #define `BOOT_COM_CAN_RX_MAX_DATA` (8)
Configure number of bytes in the host->target CAN message.
- #define `BOOT_COM_CAN_CHANNEL_INDEX` (0)
Select the desired CAN peripheral as a zero based index.
- #define `BOOT_COM_RS232_ENABLE` (1)
Enable/disable UART transport layer.
- #define `BOOT_COM_RS232_BAUDRATE` (57600)
Configure the desired communication speed.
- #define `BOOT_COM_RS232_TX_MAX_DATA` (64)
Configure number of bytes in the target->host data packet.
- #define `BOOT_COM_RS232_RX_MAX_DATA` (64)
Configure number of bytes in the host->target data packet.
- #define `BOOT_COM_RS232_CHANNEL_INDEX` (1)
Select the desired UART peripheral as a zero based index.
- #define `BOOT_BACKDOOR_HOOKS_ENABLE` (0)
Enable/disable the backdoor override hook functions.
- #define `BOOT_NVM_HOOKS_ENABLE` (0)
Enable/disable the NVM hook function for supporting additional memory devices.
- #define `BOOT_NVM_SIZE_KB` (1024)
Configure the size of the default memory device (typically flash EEPROM).
- #define `BOOT_NVM_CHECKSUM_HOOKS_ENABLE` (0)
Enable/disable hooks functions to override the user program checksum handling.
- #define `BOOT_COP_HOOKS_ENABLE` (1)
Enable/disable the hook functions for controlling the watchdog.

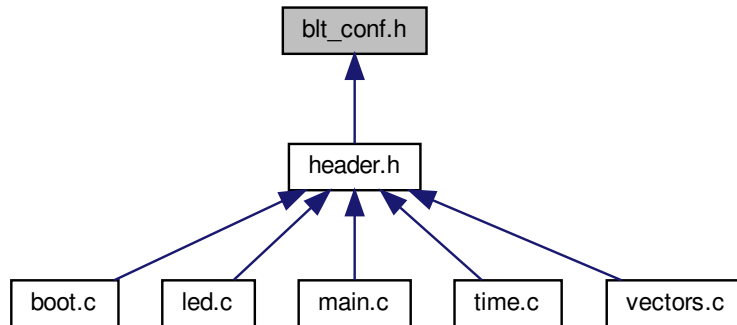
7.139.1 Detailed Description

Bootloader configuration header file.

7.140 blt_conf.h File Reference

Bootloader configuration header file.

This graph shows which files directly or indirectly include this file:



Macros

- `#define BOOT_CPU_XTAL_SPEED_KHZ (16000)`
Frequency of the external crystal oscillator.
- `#define BOOT_CPU_SYSTEM_SPEED_KHZ (50000)`
Desired system speed.
- `#define BOOT_CPU_BYTE_ORDER_MOTOROLA (0)`
Motorola or Intel style byte ordering.
- `#define BOOT_CPU_USER_PROGRAM_START_HOOK (1)`
Enable/disable hook function call right before user program start.
- `#define BOOT_COM_RS232_ENABLE (1)`
Enable/disable UART transport layer.
- `#define BOOT_COM_RS232_BAUDRATE (57600)`
Configure the desired communication speed.
- `#define BOOT_COM_RS232_TX_MAX_DATA (64)`
Configure number of bytes in the target->host data packet.
- `#define BOOT_COM_RS232_RX_MAX_DATA (64)`
Configure number of bytes in the host->target data packet.
- `#define BOOT_COM_RS232_CHANNEL_INDEX (0)`
Select the desired UART peripheral as a zero based index.
- `#define BOOT_COM_USB_ENABLE (1)`
Enable/disable USB transport layer.
- `#define BOOT_COM_USB_TX_MAX_DATA (63)`
Configure number of bytes in the target->host data packet.
- `#define BOOT_COM_USB_RX_MAX_DATA (63)`
Configure number of bytes in the host->target data packet.
- `#define BOOT_FILE_SYS_ENABLE (1)`
Enable/disable support for firmware updates from a locally attached storage.

- `#define BOOT_FILE_LOGGING_ENABLE (1)`
Enable/disable logging messages during firmware updates.
- `#define BOOT_FILE_ERROR_HOOK_ENABLE (1)`
Enable/disable a hook function that is called upon detection of an error.
- `#define BOOT_FILE_STARTED_HOOK_ENABLE (1)`
Enable/disable a hook function that is called at the start of the update.
- `#define BOOT_FILE_COMPLETED_HOOK_ENABLE (1)`
Enable/disable a hook function that is called at the end of the update.
- `#define BOOT_BACKDOOR_HOOKS_ENABLE (0)`
Enable/disable the backdoor override hook functions.
- `#define BOOT_NVM_HOOKS_ENABLE (0)`
Enable/disable the NVM hook function for supporting additional memory devices.
- `#define BOOT_NVM_SIZE_KB (256)`
Configure the size of the default memory device (typically flash EEPROM).
- `#define BOOT_NVM_CHECKSUM_HOOKS_ENABLE (0)`
Enable/disable hooks functions to override the user program checksum handling.
- `#define BOOT_COP_HOOKS_ENABLE (0)`
Enable/disable the hook functions for controlling the watchdog.

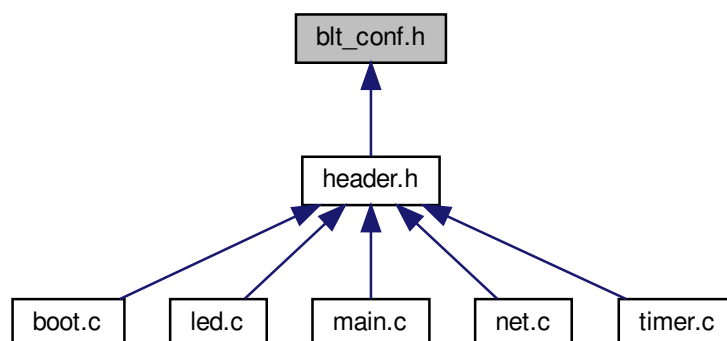
7.140.1 Detailed Description

Bootloader configuration header file.

7.141 blt_conf.h File Reference

Bootloader configuration header file.

This graph shows which files directly or indirectly include this file:



Macros

- #define `BOOT_CPU_XTAL_SPEED_KHZ` (12000)
Frequency of the external crystal oscillator.
- #define `BOOT_CPU_SYSTEM_SPEED_KHZ` (144000)
Desired system speed.
- #define `BOOT_CPU_BYTE_ORDER_MOTOROLA` (0)
Motorola or Intel style byte ordering.
- #define `BOOT_CPU_USER_PROGRAM_START_HOOK` (1)
Enable/disable hook function call right before user program start.
- #define `BOOT_COM_CAN_ENABLE` (1)
Enable/disable CAN transport layer.
- #define `BOOT_COM_CAN_BAUDRATE` (500000)
Configure the desired CAN baudrate.
- #define `BOOT_COM_CAN_TX_MSG_ID` (0x7E1 /*| 0x80000000*/) *Configure CAN message ID target->host.*
- #define `BOOT_COM_CAN_TX_MAX_DATA` (8)
Configure number of bytes in the target->host CAN message.
- #define `BOOT_COM_CAN_RX_MSG_ID` (0x667 /*| 0x80000000*/) *Configure CAN message ID host->target.*
- #define `BOOT_COM_CAN_RX_MAX_DATA` (8)
Configure number of bytes in the host->target CAN message.
- #define `BOOT_COM_CAN_CHANNEL_INDEX` (1)
Select the desired CAN peripheral as a zero based index.
- #define `BOOT_COM_RS232_ENABLE` (1)
Enable/disable UART transport layer.
- #define `BOOT_COM_RS232_BAUDRATE` (57600)
Configure the desired communication speed.
- #define `BOOT_COM_RS232_TX_MAX_DATA` (64)
Configure number of bytes in the target->host data packet.
- #define `BOOT_COM_RS232_RX_MAX_DATA` (64)
Configure number of bytes in the host->target data packet.
- #define `BOOT_COM_RS232_CHANNEL_INDEX` (0)
Select the desired UART peripheral as a zero based index.
- #define `BOOT_COM_NET_ENABLE` (1)
Enable/disable the NET transport layer.
- #define `BOOT_COM_NET_TX_MAX_DATA` (64)
Configure number of bytes in the target->host data packet.
- #define `BOOT_COM_NET_RX_MAX_DATA` (64)
Configure number of bytes in the host->target data packet.
- #define `BOOT_COM_NET_PORT` (1000)
Configure the port that the TCP/IP server listens on.
- #define `BOOT_COM_NET_DHCP_ENABLE` (1)
Enable/disable DHCP client for automatically obtaining an IP address.
- #define `BOOT_COM_NET_IPADDR0` (192)
Configure the 1st byte of the IP address.
- #define `BOOT_COM_NET_IPADDR1` (168)
Configure the 2nd byte of the IP address.
- #define `BOOT_COM_NET_IPADDR2` (178)
Configure the 3rd byte of the IP address.
- #define `BOOT_COM_NET_IPADDR3` (50)

- Configure the 4th byte of the IP address.*

 - #define `BOOT_COM_NET_NETMASK0` (255)

Configure the 1st byte of the network mask.

 - #define `BOOT_COM_NET_NETMASK1` (255)

Configure the 2nd byte of the network mask.

 - #define `BOOT_COM_NET_NETMASK2` (255)

Configure the 3rd byte of the network mask.

 - #define `BOOT_COM_NET_NETMASK3` (0)

Configure the 4th byte of the network mask.

 - #define `BOOT_COM_NET_GATEWAY0` (192)

Configure the 1st byte of the gateway address.

 - #define `BOOT_COM_NET_GATEWAY1` (168)

Configure the 2nd byte of the gateway address.

 - #define `BOOT_COM_NET_GATEWAY2` (178)

Configure the 3rd byte of the gateway address.

 - #define `BOOT_COM_NET_GATEWAY3` (1)

Configure the 4th byte of the gateway address.

 - #define `BOOT_COM_NET_DEFERRED_INIT_ENABLE` (1)

Enable/disable the deferred initialization mechanism. When enabled, the communication interface is only initialized when: (a) no valid user program is detected, or (b) when `CpuUserProgramStartHook()` returns `BLT_FALSE`. Your bootloader application can explicitly initialize the communication interface by calling `ComDeferredInit()`.

 - #define `BOOT_FILE_SYS_ENABLE` (1)

Enable/disable support for firmware updates from a locally attached storage.

 - #define `BOOT_FILE_LOGGING_ENABLE` (1)

Enable/disable logging messages during firmware updates.

 - #define `BOOT_FILE_ERROR_HOOK_ENABLE` (1)

Enable/disable a hook function that is called upon detection of an error.

 - #define `BOOT_FILE_STARTED_HOOK_ENABLE` (1)

Enable/disable a hook function that is called at the start of the update.

 - #define `BOOT_FILE_COMPLETED_HOOK_ENABLE` (1)

Enable/disable a hook function that is called at the end of the update.

 - #define `BOOT_BACKDOOR_HOOKS_ENABLE` (0)

Enable/disable the backdoor override hook functions.

 - #define `BOOT_NVM_HOOKS_ENABLE` (0)

Enable/disable the NVM hook function for supporting additional memory devices.

 - #define `BOOT_NVM_SIZE_KB` (2048)

Configure the size of the default memory device (typically flash EEPROM).

 - #define `BOOT_NVM_CHECKSUM_HOOKS_ENABLE` (0)

Enable/disable hooks functions to override the user program checksum handling.

 - #define `BOOT_COP_HOOKS_ENABLE` (1)

Enable/disable the hook functions for controlling the watchdog.

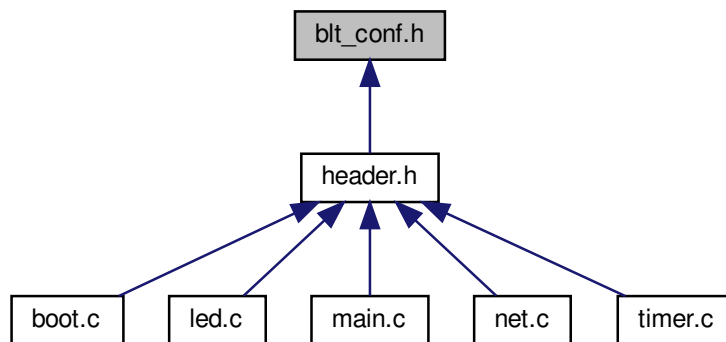
7.141.1 Detailed Description

Bootloader configuration header file.

7.142 blt_conf.h File Reference

Bootloader configuration header file.

This graph shows which files directly or indirectly include this file:



Macros

- #define `BOOT_CPU_XTAL_SPEED_KHZ` (12000)
Frequency of the external crystal oscillator.
- #define `BOOT_CPU_SYSTEM_SPEED_KHZ` (144000)
Desired system speed.
- #define `BOOT_CPU_BYTE_ORDER_MOTOROLA` (0)
Motorola or Intel style byte ordering.
- #define `BOOT_CPU_USER_PROGRAM_START_HOOK` (1)
Enable/disable hook function call right before user program start.
- #define `BOOT_COM_CAN_ENABLE` (1)
Enable/disable CAN transport layer.
- #define `BOOT_COM_CAN_BAUDRATE` (500000)
Configure the desired CAN baudrate.
- #define `BOOT_COM_CAN_TX_MSG_ID` (0x7E1 /*| 0x80000000*/) *Configure CAN message ID target->host.*
- #define `BOOT_COM_CAN_TX_MAX_DATA` (8)
Configure number of bytes in the target->host CAN message.
- #define `BOOT_COM_CAN_RX_MSG_ID` (0x667 /*| 0x80000000*/) *Configure CAN message ID host->target.*
- #define `BOOT_COM_CAN_RX_MAX_DATA` (8)
Configure number of bytes in the host->target CAN message.
- #define `BOOT_COM_CAN_CHANNEL_INDEX` (1)
Select the desired CAN peripheral as a zero based index.
- #define `BOOT_COM_RS232_ENABLE` (1)
Enable/disable UART transport layer.
- #define `BOOT_COM_RS232_BAUDRATE` (57600)

- Configure the desired communication speed.*

 - #define `BOOT_COM_RS232_TX_MAX_DATA` (64)

Configure number of bytes in the target->host data packet.

 - #define `BOOT_COM_RS232_RX_MAX_DATA` (64)
- Configure number of bytes in the host->target data packet.*
- #define `BOOT_COM_RS232_CHANNEL_INDEX` (0)
- Select the desired UART peripheral as a zero based index.*
- #define `BOOT_COM_NET_ENABLE` (1)
- Enable/disable the NET transport layer.*
- #define `BOOT_COM_NET_TX_MAX_DATA` (64)
- Configure number of bytes in the target->host data packet.*
- #define `BOOT_COM_NET_RX_MAX_DATA` (64)
- Configure number of bytes in the host->target data packet.*
- #define `BOOT_COM_NET_PORT` (1000)
- Configure the port that the TCP/IP server listens on.*
- #define `BOOT_COM_NET_DHCP_ENABLE` (1)
- Enable/disable DHCP client for automatically obtaining an IP address.*
- #define `BOOT_COM_NET_IPADDR0` (192)
- Configure the 1st byte of the IP address.*
- #define `BOOT_COM_NET_IPADDR1` (168)
- Configure the 2nd byte of the IP address.*
- #define `BOOT_COM_NET_IPADDR2` (178)
- Configure the 3rd byte of the IP address.*
- #define `BOOT_COM_NET_IPADDR3` (50)
- Configure the 4th byte of the IP address.*
- #define `BOOT_COM_NET_NETMASK0` (255)
- Configure the 1st byte of the network mask.*
- #define `BOOT_COM_NET_NETMASK1` (255)
- Configure the 2nd byte of the network mask.*
- #define `BOOT_COM_NET_NETMASK2` (255)
- Configure the 3rd byte of the network mask.*
- #define `BOOT_COM_NET_NETMASK3` (0)
- Configure the 4th byte of the network mask.*
- #define `BOOT_COM_NET_GATEWAY0` (192)
- Configure the 1st byte of the gateway address.*
- #define `BOOT_COM_NET_GATEWAY1` (168)
- Configure the 2nd byte of the gateway address.*
- #define `BOOT_COM_NET_GATEWAY2` (178)
- Configure the 3rd byte of the gateway address.*
- #define `BOOT_COM_NET_GATEWAY3` (1)
- Configure the 4th byte of the gateway address.*
- #define `BOOT_COM_NET_DEFERRED_INIT_ENABLE` (1)
- Enable/disable the deferred initialization mechanism. When enabled, the communication interface is only initialized when: (a) no valid user program is detected, or (b) when `CpuUserProgramStartHook()` returns `BLT_FALSE`. Your bootloader application can explicitly initialize the communication interface by calling `ComDeferredInit()`.*
- #define `BOOT_FILE_SYS_ENABLE` (1)
- Enable/disable support for firmware updates from a locally attached storage.*
- #define `BOOT_FILE_LOGGING_ENABLE` (1)
- Enable/disable logging messages during firmware updates.*
- #define `BOOT_FILE_ERROR_HOOK_ENABLE` (1)
- Enable/disable a hook function that is called upon detection of an error.*

- #define `BOOT_FILE_STARTED_HOOK_ENABLE` (1)
Enable/disable a hook function that is called at the start of the update.
- #define `BOOT_FILE_COMPLETED_HOOK_ENABLE` (1)
Enable/disable a hook function that is called at the end of the update.
- #define `BOOT_BACKDOOR_HOOKS_ENABLE` (0)
Enable/disable the backdoor override hook functions.
- #define `BOOT_NVM_HOOKS_ENABLE` (0)
Enable/disable the NVM hook function for supporting additional memory devices.
- #define `BOOT_NVM_SIZE_KB` (2048)
Configure the size of the default memory device (typically flash EEPROM).
- #define `BOOT_NVM_CHECKSUM_HOOKS_ENABLE` (0)
Enable/disable hooks functions to override the user program checksum handling.
- #define `BOOT_COP_HOOKS_ENABLE` (1)
Enable/disable the hook functions for controlling the watchdog.

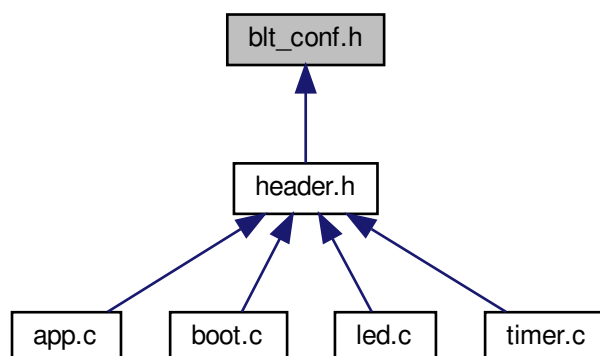
7.142.1 Detailed Description

Bootloader configuration header file.

7.143 blt_conf.h File Reference

Bootloader configuration header file.

This graph shows which files directly or indirectly include this file:



Macros

- #define [BOOT_CPU_XTAL_SPEED_KHZ](#) (8000)
Frequency of the external crystal oscillator.
- #define [BOOT_CPU_SYSTEM_SPEED_KHZ](#) (216000)
Desired system speed.
- #define [BOOT_CPU_BYTE_ORDER_MOTOROLA](#) (0)
Motorola or Intel style byte ordering.
- #define [BOOT_CPU_USER_PROGRAM_START_HOOK](#) (1)
Enable/disable hook function call right before user program start.
- #define [BOOT_COM_USB_ENABLE](#) (1)
Enable/disable USB transport layer.
- #define [BOOT_COM_USB_TX_MAX_DATA](#) (63)
Configure number of bytes in the target->host data packet.
- #define [BOOT_COM_USB_RX_MAX_DATA](#) (63)
Configure number of bytes in the host->target data packet.
- #define [BOOT_COM_CAN_ENABLE](#) (1)
Enable/disable CAN transport layer.
- #define [BOOT_COM_CAN_BAUDRATE](#) (500000)
Configure the desired CAN baudrate.
- #define [BOOT_COM_CAN_TX_MSG_ID](#) (0x7E1 /*| 0x80000000*/) *Configure CAN message ID target->host.*
- #define [BOOT_COM_CAN_TX_MAX_DATA](#) (8)
Configure number of bytes in the target->host CAN message.
- #define [BOOT_COM_CAN_RX_MSG_ID](#) (0x667 /*| 0x80000000*/) *Configure CAN message ID host->target.*
- #define [BOOT_COM_CAN_RX_MAX_DATA](#) (8)
Configure number of bytes in the host->target CAN message.
- #define [BOOT_COM_CAN_CHANNEL_INDEX](#) (0)
Select the desired CAN peripheral as a zero based index.
- #define [BOOT_COM_RS232_ENABLE](#) (1)
Enable/disable UART transport layer.
- #define [BOOT_COM_RS232_BAUDRATE](#) (57600)
Configure the desired communication speed.
- #define [BOOT_COM_RS232_TX_MAX_DATA](#) (64)
Configure number of bytes in the target->host data packet.
- #define [BOOT_COM_RS232_RX_MAX_DATA](#) (64)
Configure number of bytes in the host->target data packet.
- #define [BOOT_COM_RS232_CHANNEL_INDEX](#) (2)
Select the desired UART peripheral as a zero based index.
- #define [BOOT_BACKDOOR_HOOKS_ENABLE](#) (0)
Enable/disable the backdoor override hook functions.
- #define [BOOT_NVM_HOOKS_ENABLE](#) (0)
Enable/disable the NVM hook function for supporting additional memory devices.
- #define [BOOT_NVM_SIZE_KB](#) (1024)
Configure the size of the default memory device (typically flash EEPROM).
- #define [BOOT_NVM_CHECKSUM_HOOKS_ENABLE](#) (0)
Enable/disable hooks functions to override the user program checksum handling.
- #define [BOOT_COP_HOOKS_ENABLE](#) (1)
Enable/disable the hook functions for controlling the watchdog.

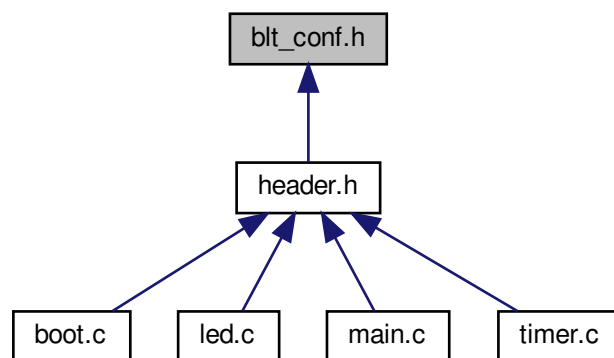
7.143.1 Detailed Description

Bootloader configuration header file.

7.144 blt_conf.h File Reference

Bootloader configuration header file.

This graph shows which files directly or indirectly include this file:



Macros

- `#define BOOT_CPU_XTAL_SPEED_KHZ` (8000)
Frequency of the external crystal oscillator.
- `#define BOOT_CPU_SYSTEM_SPEED_KHZ` (216000)
Desired system speed.
- `#define BOOT_CPU_BYTE_ORDER_MOTOROLA` (0)
Motorola or Intel style byte ordering.
- `#define BOOT_CPU_USER_PROGRAM_START_HOOK` (1)
Enable/disable hook function call right before user program start.
- `#define BOOT_COM_USB_ENABLE` (1)
Enable/disable USB transport layer.
- `#define BOOT_COM_USB_TX_MAX_DATA` (63)
Configure number of bytes in the target->host data packet.
- `#define BOOT_COM_USB_RX_MAX_DATA` (63)
Configure number of bytes in the host->target data packet.
- `#define BOOT_COM_CAN_ENABLE` (1)
Enable/disable CAN transport layer.
- `#define BOOT_COM_CAN_BAUDRATE` (500000)
Configure the desired CAN baudrate.
- `#define BOOT_COM_CAN_TX_MSG_ID` (0x7E1 /*| 0x80000000*/)

- Configure CAN message ID target->host.*

 - #define `BOOT_COM_CAN_TX_MAX_DATA` (8)

Configure number of bytes in the target->host CAN message.
- #define `BOOT_COM_CAN_RX_MSG_ID` (0x667 /*| 0x80000000*/)

Configure CAN message ID host->target.
- #define `BOOT_COM_CAN_RX_MAX_DATA` (8)

Configure number of bytes in the host->target CAN message.
- #define `BOOT_COM_CAN_CHANNEL_INDEX` (0)

Select the desired CAN peripheral as a zero based index.
- #define `BOOT_COM_RS232_ENABLE` (1)

Enable/disable UART transport layer.
- #define `BOOT_COM_RS232_BAUDRATE` (57600)

Configure the desired communication speed.
- #define `BOOT_COM_RS232_TX_MAX_DATA` (64)

Configure number of bytes in the target->host data packet.
- #define `BOOT_COM_RS232_RX_MAX_DATA` (64)

Configure number of bytes in the host->target data packet.
- #define `BOOT_COM_RS232_CHANNEL_INDEX` (2)

Select the desired UART peripheral as a zero based index.
- #define `BOOT_BACKDOOR_HOOKS_ENABLE` (0)

Enable/disable the backdoor override hook functions.
- #define `BOOT_NVM_HOOKS_ENABLE` (0)

Enable/disable the NVM hook function for supporting additional memory devices.
- #define `BOOT_NVM_SIZE_KB` (1024)

Configure the size of the default memory device (typically flash EEPROM).
- #define `BOOT_NVM_CHECKSUM_HOOKS_ENABLE` (0)

Enable/disable hooks functions to override the user program checksum handling.
- #define `BOOT_COP_HOOKS_ENABLE` (1)

Enable/disable the hook functions for controlling the watchdog.

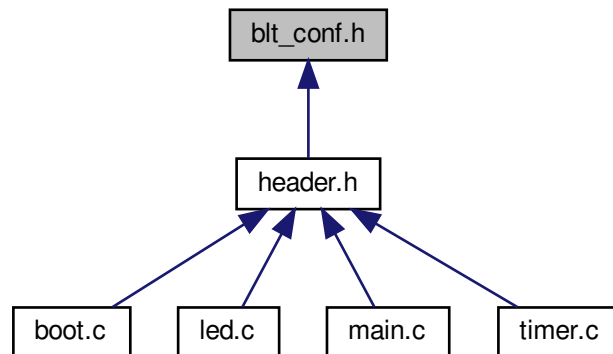
7.144.1 Detailed Description

Bootloader configuration header file.

7.145 blt_conf.h File Reference

Bootloader configuration header file.

This graph shows which files directly or indirectly include this file:



Macros

- `#define BOOT_CPU_XTAL_SPEED_KHZ (8000)`
Frequency of the external crystal oscillator.
- `#define BOOT_CPU_SYSTEM_SPEED_KHZ (216000)`
Desired system speed.
- `#define BOOT_CPU_BYTE_ORDER_MOTOROLA (0)`
Motorola or Intel style byte ordering.
- `#define BOOT_CPU_USER_PROGRAM_START_HOOK (1)`
Enable/disable hook function call right before user program start.
- `#define BOOT_COM_USB_ENABLE (1)`
Enable/disable USB transport layer.
- `#define BOOT_COM_USB_TX_MAX_DATA (63)`
Configure number of bytes in the target->host data packet.
- `#define BOOT_COM_USB_RX_MAX_DATA (63)`
Configure number of bytes in the host->target data packet.
- `#define BOOT_COM_CAN_ENABLE (1)`
Enable/disable CAN transport layer.
- `#define BOOT_COM_CAN_BAUDRATE (500000)`
Configure the desired CAN baudrate.
- `#define BOOT_COM_CAN_TX_MSG_ID (0x7E1 /*| 0x80000000*/)`
Configure CAN message ID target->host.
- `#define BOOT_COM_CAN_TX_MAX_DATA (8)`
Configure number of bytes in the target->host CAN message.
- `#define BOOT_COM_CAN_RX_MSG_ID (0x667 /*| 0x80000000*/)`
Configure CAN message ID host->target.
- `#define BOOT_COM_CAN_RX_MAX_DATA (8)`
Configure number of bytes in the host->target CAN message.
- `#define BOOT_COM_CAN_CHANNEL_INDEX (0)`
Select the desired CAN peripheral as a zero based index.
- `#define BOOT_COM_RS232_ENABLE (1)`

- *Enable/disable UART transport layer.*
• #define [BOOT_COM_RS232_BAUDRATE](#) (57600)
Configure the desired communication speed.
- #define [BOOT_COM_RS232_TX_MAX_DATA](#) (64)
Configure number of bytes in the target->host data packet.
- #define [BOOT_COM_RS232_RX_MAX_DATA](#) (64)
Configure number of bytes in the host->target data packet.
- #define [BOOT_COM_RS232_CHANNEL_INDEX](#) (2)
Select the desired UART peripheral as a zero based index.
- #define [BOOT_BACKDOOR_HOOKS_ENABLE](#) (0)
Enable/disable the backdoor override hook functions.
- #define [BOOT_NVM_HOOKS_ENABLE](#) (0)
Enable/disable the NVM hook function for supporting additional memory devices.
- #define [BOOT_NVM_SIZE_KB](#) (1024)
Configure the size of the default memory device (typically flash EEPROM).
- #define [BOOT_NVM_CHECKSUM_HOOKS_ENABLE](#) (0)
Enable/disable hooks functions to override the user program checksum handling.
- #define [BOOT_COP_HOOKS_ENABLE](#) (1)
Enable/disable the hook functions for controlling the watchdog.

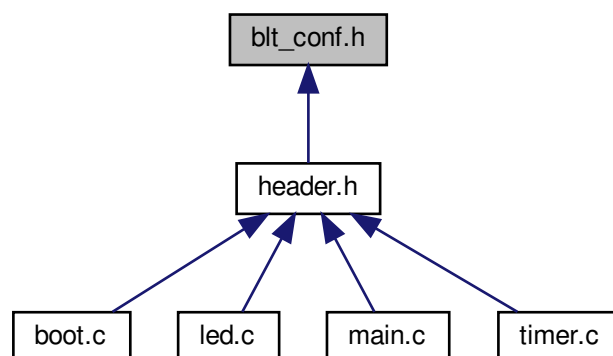
7.145.1 Detailed Description

Bootloader configuration header file.

7.146 blt_conf.h File Reference

Bootloader configuration header file.

This graph shows which files directly or indirectly include this file:



Macros

- #define [BOOT_CPU_XTAL_SPEED_KHZ](#) (8000)
Frequency of the external crystal oscillator.
- #define [BOOT_CPU_SYSTEM_SPEED_KHZ](#) (216000)
Desired system speed.
- #define [BOOT_CPU_BYTE_ORDER_MOTOROLA](#) (0)
Motorola or Intel style byte ordering.
- #define [BOOT_CPU_USER_PROGRAM_START_HOOK](#) (1)
Enable/disable hook function call right before user program start.
- #define [BOOT_COM_USB_ENABLE](#) (1)
Enable/disable USB transport layer.
- #define [BOOT_COM_USB_TX_MAX_DATA](#) (63)
Configure number of bytes in the target->host data packet.
- #define [BOOT_COM_USB_RX_MAX_DATA](#) (63)
Configure number of bytes in the host->target data packet.
- #define [BOOT_COM_CAN_ENABLE](#) (1)
Enable/disable CAN transport layer.
- #define [BOOT_COM_CAN_BAUDRATE](#) (500000)
Configure the desired CAN baudrate.
- #define [BOOT_COM_CAN_TX_MSG_ID](#) (0x7E1 /*| 0x80000000*/) *Configure CAN message ID target->host.*
- #define [BOOT_COM_CAN_TX_MAX_DATA](#) (8)
Configure number of bytes in the target->host CAN message.
- #define [BOOT_COM_CAN_RX_MSG_ID](#) (0x667 /*| 0x80000000*/) *Configure CAN message ID host->target.*
- #define [BOOT_COM_CAN_RX_MAX_DATA](#) (8)
Configure number of bytes in the host->target CAN message.
- #define [BOOT_COM_CAN_CHANNEL_INDEX](#) (0)
Select the desired CAN peripheral as a zero based index.
- #define [BOOT_COM_RS232_ENABLE](#) (1)
Enable/disable UART transport layer.
- #define [BOOT_COM_RS232_BAUDRATE](#) (57600)
Configure the desired communication speed.
- #define [BOOT_COM_RS232_TX_MAX_DATA](#) (64)
Configure number of bytes in the target->host data packet.
- #define [BOOT_COM_RS232_RX_MAX_DATA](#) (64)
Configure number of bytes in the host->target data packet.
- #define [BOOT_COM_RS232_CHANNEL_INDEX](#) (2)
Select the desired UART peripheral as a zero based index.
- #define [BOOT_BACKDOOR_HOOKS_ENABLE](#) (0)
Enable/disable the backdoor override hook functions.
- #define [BOOT_NVM_HOOKS_ENABLE](#) (0)
Enable/disable the NVM hook function for supporting additional memory devices.
- #define [BOOT_NVM_SIZE_KB](#) (1024)
Configure the size of the default memory device (typically flash EEPROM).
- #define [BOOT_NVM_CHECKSUM_HOOKS_ENABLE](#) (0)
Enable/disable hooks functions to override the user program checksum handling.
- #define [BOOT_COP_HOOKS_ENABLE](#) (1)
Enable/disable the hook functions for controlling the watchdog.

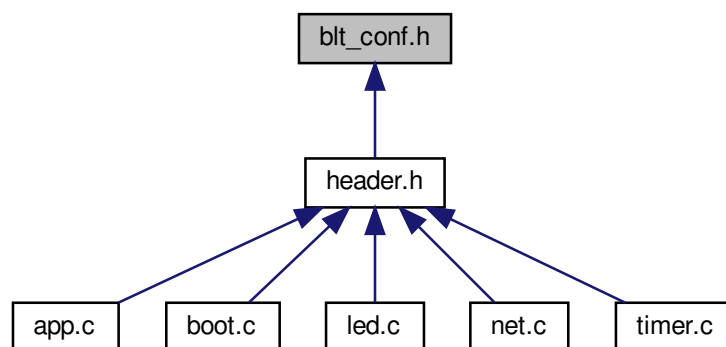
7.146.1 Detailed Description

Bootloader configuration header file.

7.147 blt_conf.h File Reference

Bootloader configuration header file.

This graph shows which files directly or indirectly include this file:



Macros

- `#define BOOT_CPU_XTAL_SPEED_KHZ (8000)`
Frequency of the external crystal oscillator.
- `#define BOOT_CPU_SYSTEM_SPEED_KHZ (216000)`
Desired system speed.
- `#define BOOT_CPU_BYTE_ORDER_MOTOROLA (0)`
Motorola or Intel style byte ordering.
- `#define BOOT_CPU_USER_PROGRAM_START_HOOK (1)`
Enable/disable hook function call right before user program start.
- `#define BOOT_COM_NET_ENABLE (1)`
Enable/disable the NET transport layer.
- `#define BOOT_COM_NET_TX_MAX_DATA (64)`
Configure number of bytes in the target->host data packet.
- `#define BOOT_COM_NET_RX_MAX_DATA (64)`
Configure number of bytes in the host->target data packet.
- `#define BOOT_COM_NET_PORT (1000)`
Configure the port that the TCP/IP server listens on.
- `#define BOOT_COM_NET_DHCP_ENABLE (1)`
Enable/disable DHCP client for automatically obtaining an IP address.
- `#define BOOT_COM_NET_IPADDR0 (192)`
Configure the 1st byte of the IP address.

- #define [BOOT_COM_NET_IPADDR1](#) (168)
Configure the 2nd byte of the IP address.
- #define [BOOT_COM_NET_IPADDR2](#) (178)
Configure the 3rd byte of the IP address.
- #define [BOOT_COM_NET_IPADDR3](#) (50)
Configure the 4th byte of the IP address.
- #define [BOOT_COM_NET_NETMASK0](#) (255)
Configure the 1st byte of the network mask.
- #define [BOOT_COM_NET_NETMASK1](#) (255)
Configure the 2nd byte of the network mask.
- #define [BOOT_COM_NET_NETMASK2](#) (255)
Configure the 3rd byte of the network mask.
- #define [BOOT_COM_NET_NETMASK3](#) (0)
Configure the 4th byte of the network mask.
- #define [BOOT_COM_NET_GATEWAY0](#) (192)
Configure the 1st byte of the gateway address.
- #define [BOOT_COM_NET_GATEWAY1](#) (168)
Configure the 2nd byte of the gateway address.
- #define [BOOT_COM_NET_GATEWAY2](#) (178)
Configure the 3rd byte of the gateway address.
- #define [BOOT_COM_NET_GATEWAY3](#) (1)
Configure the 4th byte of the gateway address.
- #define [BOOT_COM_NET_DEFERRED_INIT_ENABLE](#) (1)
Enable/disable the deferred initialization mechanism. When enabled, the communication interface is only initialized when: (a) no valid user program is detected, or (b) when [CpuUserProgramStartHook\(\)](#) returns BLT_FALSE. Your bootloader application can explicitly initialize the communication interface by calling [ComDeferredInit\(\)](#).
- #define [BOOT_COM_RS232_ENABLE](#) (1)
Enable/disable UART transport layer.
- #define [BOOT_COM_RS232_BAUDRATE](#) (57600)
Configure the desired communication speed.
- #define [BOOT_COM_RS232_TX_MAX_DATA](#) (64)
Configure number of bytes in the target->host data packet.
- #define [BOOT_COM_RS232_RX_MAX_DATA](#) (64)
Configure number of bytes in the host->target data packet.
- #define [BOOT_COM_RS232_CHANNEL_INDEX](#) (2)
Select the desired UART peripheral as a zero based index.
- #define [BOOT_BACKDOOR_HOOKS_ENABLE](#) (0)
Enable/disable the backdoor override hook functions.
- #define [BOOT_NVM_HOOKS_ENABLE](#) (0)
Enable/disable the NVM hook function for supporting additional memory devices.
- #define [BOOT_NVM_SIZE_KB](#) (2048)
Configure the size of the default memory device (typically flash EEPROM).
- #define [BOOT_NVM_CHECKSUM_HOOKS_ENABLE](#) (0)
Enable/disable hooks functions to override the user program checksum handling.
- #define [BOOT_FLASH_VECTOR_TABLE_CS_OFFSET](#) (0x1F8)
This microcontroller has a larger vector table then the default STM32F7xx project as assumed in the bootloader's core. This means the user program has a different checksum location, because this one is added at the end of the user program's vector table.
- #define [BOOT_COP_HOOKS_ENABLE](#) (1)
Enable/disable the hook functions for controlling the watchdog.

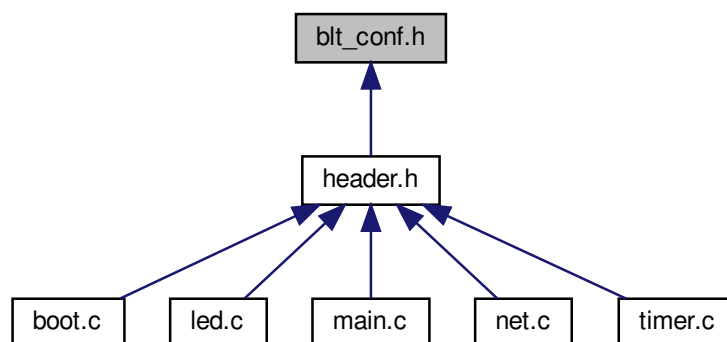
7.147.1 Detailed Description

Bootloader configuration header file.

7.148 blt_conf.h File Reference

Bootloader configuration header file.

This graph shows which files directly or indirectly include this file:



Macros

- `#define BOOT_CPU_XTAL_SPEED_KHZ (8000)`
Frequency of the external crystal oscillator.
- `#define BOOT_CPU_SYSTEM_SPEED_KHZ (216000)`
Desired system speed.
- `#define BOOT_CPU_BYTE_ORDER_MOTOROLA (0)`
Motorola or Intel style byte ordering.
- `#define BOOT_CPU_USER_PROGRAM_START_HOOK (1)`
Enable/disable hook function call right before user program start.
- `#define BOOT_COM_NET_ENABLE (1)`
Enable/disable the NET transport layer.
- `#define BOOT_COM_NET_TX_MAX_DATA (64)`
Configure number of bytes in the target->host data packet.
- `#define BOOT_COM_NET_RX_MAX_DATA (64)`
Configure number of bytes in the host->target data packet.
- `#define BOOT_COM_NET_PORT (1000)`
Configure the port that the TCP/IP server listens on.
- `#define BOOT_COM_NET_DHCP_ENABLE (1)`
Enable/disable DHCP client for automatically obtaining an IP address.
- `#define BOOT_COM_NET_IPADDR0 (192)`
Configure the 1st byte of the IP address.

- #define [BOOT_COM_NET_IPADDR1](#) (168)
Configure the 2nd byte of the IP address.
- #define [BOOT_COM_NET_IPADDR2](#) (178)
Configure the 3rd byte of the IP address.
- #define [BOOT_COM_NET_IPADDR3](#) (50)
Configure the 4th byte of the IP address.
- #define [BOOT_COM_NET_NETMASK0](#) (255)
Configure the 1st byte of the network mask.
- #define [BOOT_COM_NET_NETMASK1](#) (255)
Configure the 2nd byte of the network mask.
- #define [BOOT_COM_NET_NETMASK2](#) (255)
Configure the 3rd byte of the network mask.
- #define [BOOT_COM_NET_NETMASK3](#) (0)
Configure the 4th byte of the network mask.
- #define [BOOT_COM_NET_GATEWAY0](#) (192)
Configure the 1st byte of the gateway address.
- #define [BOOT_COM_NET_GATEWAY1](#) (168)
Configure the 2nd byte of the gateway address.
- #define [BOOT_COM_NET_GATEWAY2](#) (178)
Configure the 3rd byte of the gateway address.
- #define [BOOT_COM_NET_GATEWAY3](#) (1)
Configure the 4th byte of the gateway address.
- #define [BOOT_COM_NET_DEFERRED_INIT_ENABLE](#) (1)
Enable/disable the deferred initialization mechanism. When enabled, the communication interface is only initialized when: (a) no valid user program is detected, or (b) when [CpuUserProgramStartHook\(\)](#) returns BLT_FALSE. Your bootloader application can explicitly initialize the communication interface by calling [ComDeferredInit\(\)](#).
- #define [BOOT_COM_RS232_ENABLE](#) (1)
Enable/disable UART transport layer.
- #define [BOOT_COM_RS232_BAUDRATE](#) (57600)
Configure the desired communication speed.
- #define [BOOT_COM_RS232_TX_MAX_DATA](#) (64)
Configure number of bytes in the target->host data packet.
- #define [BOOT_COM_RS232_RX_MAX_DATA](#) (64)
Configure number of bytes in the host->target data packet.
- #define [BOOT_COM_RS232_CHANNEL_INDEX](#) (2)
Select the desired UART peripheral as a zero based index.
- #define [BOOT_BACKDOOR_HOOKS_ENABLE](#) (0)
Enable/disable the backdoor override hook functions.
- #define [BOOT_NVM_HOOKS_ENABLE](#) (0)
Enable/disable the NVM hook function for supporting additional memory devices.
- #define [BOOT_NVM_SIZE_KB](#) (2048)
Configure the size of the default memory device (typically flash EEPROM).
- #define [BOOT_NVM_CHECKSUM_HOOKS_ENABLE](#) (0)
Enable/disable hooks functions to override the user program checksum handling.
- #define [BOOT_FLASH_VECTOR_TABLE_CS_OFFSET](#) (0x1F8)
This microcontroller has a larger vector table then the default STM32F7xx project as assumed in the bootloader's core. This means the user program has a different checksum location, because this one is added at the end of the user program's vector table.
- #define [BOOT_COP_HOOKS_ENABLE](#) (1)
Enable/disable the hook functions for controlling the watchdog.

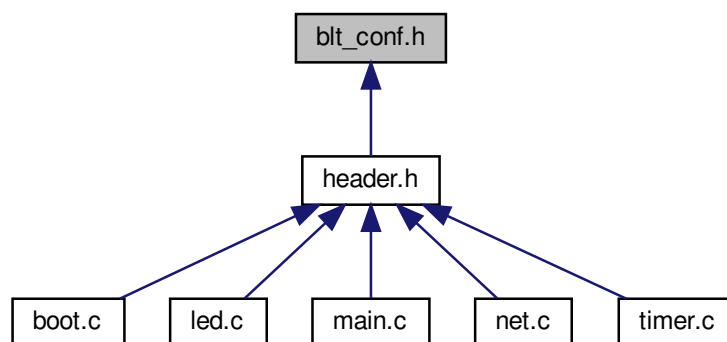
7.148.1 Detailed Description

Bootloader configuration header file.

7.149 blt_conf.h File Reference

Bootloader configuration header file.

This graph shows which files directly or indirectly include this file:



Macros

- `#define BOOT_CPU_XTAL_SPEED_KHZ (8000)`
Frequency of the external crystal oscillator.
- `#define BOOT_CPU_SYSTEM_SPEED_KHZ (216000)`
Desired system speed.
- `#define BOOT_CPU_BYTE_ORDER_MOTOROLA (0)`
Motorola or Intel style byte ordering.
- `#define BOOT_CPU_USER_PROGRAM_START_HOOK (1)`
Enable/disable hook function call right before user program start.
- `#define BOOT_COM_NET_ENABLE (1)`
Enable/disable the NET transport layer.
- `#define BOOT_COM_NET_TX_MAX_DATA (64)`
Configure number of bytes in the target->host data packet.
- `#define BOOT_COM_NET_RX_MAX_DATA (64)`
Configure number of bytes in the host->target data packet.
- `#define BOOT_COM_NET_PORT (1000)`
Configure the port that the TCP/IP server listens on.
- `#define BOOT_COM_NET_DHCP_ENABLE (1)`
Enable/disable DHCP client for automatically obtaining an IP address.
- `#define BOOT_COM_NET_IPADDR0 (192)`
Configure the 1st byte of the IP address.

- #define [BOOT_COM_NET_IPADDR1](#) (168)
Configure the 2nd byte of the IP address.
- #define [BOOT_COM_NET_IPADDR2](#) (178)
Configure the 3rd byte of the IP address.
- #define [BOOT_COM_NET_IPADDR3](#) (50)
Configure the 4th byte of the IP address.
- #define [BOOT_COM_NET_NETMASK0](#) (255)
Configure the 1st byte of the network mask.
- #define [BOOT_COM_NET_NETMASK1](#) (255)
Configure the 2nd byte of the network mask.
- #define [BOOT_COM_NET_NETMASK2](#) (255)
Configure the 3rd byte of the network mask.
- #define [BOOT_COM_NET_NETMASK3](#) (0)
Configure the 4th byte of the network mask.
- #define [BOOT_COM_NET_GATEWAY0](#) (192)
Configure the 1st byte of the gateway address.
- #define [BOOT_COM_NET_GATEWAY1](#) (168)
Configure the 2nd byte of the gateway address.
- #define [BOOT_COM_NET_GATEWAY2](#) (178)
Configure the 3rd byte of the gateway address.
- #define [BOOT_COM_NET_GATEWAY3](#) (1)
Configure the 4th byte of the gateway address.
- #define [BOOT_COM_NET_DEFERRED_INIT_ENABLE](#) (1)
Enable/disable the deferred initialization mechanism. When enabled, the communication interface is only initialized when: (a) no valid user program is detected, or (b) when [CpuUserProgramStartHook\(\)](#) returns BLT_FALSE. Your bootloader application can explicitly initialize the communication interface by calling [ComDeferredInit\(\)](#).
- #define [BOOT_COM_RS232_ENABLE](#) (1)
Enable/disable UART transport layer.
- #define [BOOT_COM_RS232_BAUDRATE](#) (57600)
Configure the desired communication speed.
- #define [BOOT_COM_RS232_TX_MAX_DATA](#) (64)
Configure number of bytes in the target->host data packet.
- #define [BOOT_COM_RS232_RX_MAX_DATA](#) (64)
Configure number of bytes in the host->target data packet.
- #define [BOOT_COM_RS232_CHANNEL_INDEX](#) (2)
Select the desired UART peripheral as a zero based index.
- #define [BOOT_BACKDOOR_HOOKS_ENABLE](#) (0)
Enable/disable the backdoor override hook functions.
- #define [BOOT_NVM_HOOKS_ENABLE](#) (0)
Enable/disable the NVM hook function for supporting additional memory devices.
- #define [BOOT_NVM_SIZE_KB](#) (2048)
Configure the size of the default memory device (typically flash EEPROM).
- #define [BOOT_NVM_CHECKSUM_HOOKS_ENABLE](#) (0)
Enable/disable hooks functions to override the user program checksum handling.
- #define [BOOT_FLASH_VECTOR_TABLE_CS_OFFSET](#) (0x1F8)
This microcontroller has a larger vector table then the default STM32F7xx project as assumed in the bootloader's core. This means the user program has a different checksum location, because this one is added at the end of the user program's vector table.
- #define [BOOT_COP_HOOKS_ENABLE](#) (1)
Enable/disable the hook functions for controlling the watchdog.

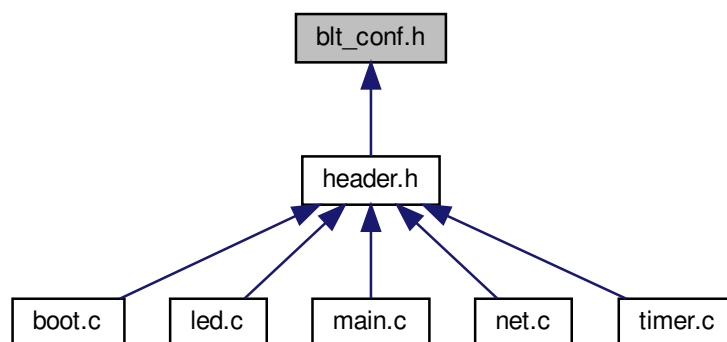
7.149.1 Detailed Description

Bootloader configuration header file.

7.150 blt_conf.h File Reference

Bootloader configuration header file.

This graph shows which files directly or indirectly include this file:



Macros

- `#define BOOT_CPU_XTAL_SPEED_KHZ (8000)`
Frequency of the external crystal oscillator.
- `#define BOOT_CPU_SYSTEM_SPEED_KHZ (216000)`
Desired system speed.
- `#define BOOT_CPU_BYTE_ORDER_MOTOROLA (0)`
Motorola or Intel style byte ordering.
- `#define BOOT_CPU_USER_PROGRAM_START_HOOK (1)`
Enable/disable hook function call right before user program start.
- `#define BOOT_COM_NET_ENABLE (1)`
Enable/disable the NET transport layer.
- `#define BOOT_COM_NET_TX_MAX_DATA (64)`
Configure number of bytes in the target->host data packet.
- `#define BOOT_COM_NET_RX_MAX_DATA (64)`
Configure number of bytes in the host->target data packet.
- `#define BOOT_COM_NET_PORT (1000)`
Configure the port that the TCP/IP server listens on.
- `#define BOOT_COM_NET_DHCP_ENABLE (1)`
Enable/disable DHCP client for automatically obtaining an IP address.
- `#define BOOT_COM_NET_IPADDR0 (192)`
Configure the 1st byte of the IP address.

- #define [BOOT_COM_NET_IPADDR1](#) (168)
Configure the 2nd byte of the IP address.
- #define [BOOT_COM_NET_IPADDR2](#) (178)
Configure the 3rd byte of the IP address.
- #define [BOOT_COM_NET_IPADDR3](#) (50)
Configure the 4th byte of the IP address.
- #define [BOOT_COM_NET_NETMASK0](#) (255)
Configure the 1st byte of the network mask.
- #define [BOOT_COM_NET_NETMASK1](#) (255)
Configure the 2nd byte of the network mask.
- #define [BOOT_COM_NET_NETMASK2](#) (255)
Configure the 3rd byte of the network mask.
- #define [BOOT_COM_NET_NETMASK3](#) (0)
Configure the 4th byte of the network mask.
- #define [BOOT_COM_NET_GATEWAY0](#) (192)
Configure the 1st byte of the gateway address.
- #define [BOOT_COM_NET_GATEWAY1](#) (168)
Configure the 2nd byte of the gateway address.
- #define [BOOT_COM_NET_GATEWAY2](#) (178)
Configure the 3rd byte of the gateway address.
- #define [BOOT_COM_NET_GATEWAY3](#) (1)
Configure the 4th byte of the gateway address.
- #define [BOOT_COM_NET_DEFERRED_INIT_ENABLE](#) (1)
Enable/disable the deferred initialization mechanism. When enabled, the communication interface is only initialized when: (a) no valid user program is detected, or (b) when [CpuUserProgramStartHook\(\)](#) returns BLT_FALSE. Your bootloader application can explicitly initialize the communication interface by calling [ComDeferredInit\(\)](#).
- #define [BOOT_COM_RS232_ENABLE](#) (1)
Enable/disable UART transport layer.
- #define [BOOT_COM_RS232_BAUDRATE](#) (57600)
Configure the desired communication speed.
- #define [BOOT_COM_RS232_TX_MAX_DATA](#) (64)
Configure number of bytes in the target->host data packet.
- #define [BOOT_COM_RS232_RX_MAX_DATA](#) (64)
Configure number of bytes in the host->target data packet.
- #define [BOOT_COM_RS232_CHANNEL_INDEX](#) (2)
Select the desired UART peripheral as a zero based index.
- #define [BOOT_BACKDOOR_HOOKS_ENABLE](#) (0)
Enable/disable the backdoor override hook functions.
- #define [BOOT_NVM_HOOKS_ENABLE](#) (0)
Enable/disable the NVM hook function for supporting additional memory devices.
- #define [BOOT_NVM_SIZE_KB](#) (2048)
Configure the size of the default memory device (typically flash EEPROM).
- #define [BOOT_NVM_CHECKSUM_HOOKS_ENABLE](#) (0)
Enable/disable hooks functions to override the user program checksum handling.
- #define [BOOT_FLASH_VECTOR_TABLE_CS_OFFSET](#) (0x1F8)
This microcontroller has a larger vector table then the default STM32F7xx project as assumed in the bootloader's core. This means the user program has a different checksum location, because this one is added at the end of the user program's vector table.
- #define [BOOT_COP_HOOKS_ENABLE](#) (1)
Enable/disable the hook functions for controlling the watchdog.

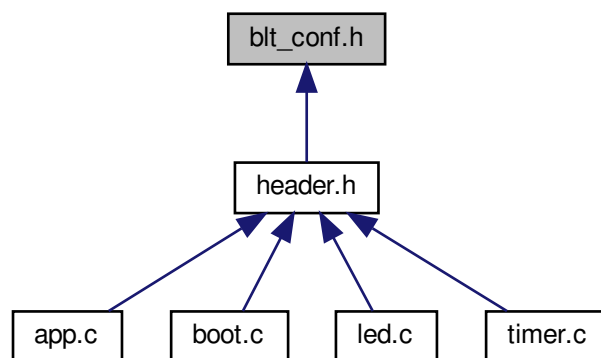
7.150.1 Detailed Description

Bootloader configuration header file.

7.151 blt_conf.h File Reference

Bootloader configuration header file.

This graph shows which files directly or indirectly include this file:



Macros

- `#define BOOT_CPU_XTAL_SPEED_KHZ (8000)`
Frequency of the external crystal oscillator.
- `#define BOOT_CPU_SYSTEM_SPEED_KHZ (480000)`
Desired system speed.
- `#define BOOT_CPU_BYTE_ORDER_MOTOROLA (0)`
Motorola or Intel style byte ordering.
- `#define BOOT_CPU_USER_PROGRAM_START_HOOK (1)`
Enable/disable hook function call right before user program start.
- `#define BOOT_COM_USB_ENABLE (1)`
Enable/disable USB transport layer.
- `#define BOOT_COM_USB_TX_MAX_DATA (63)`
Configure number of bytes in the target->host data packet.
- `#define BOOT_COM_USB_RX_MAX_DATA (63)`
Configure number of bytes in the host->target data packet.
- `#define BOOT_COM_CAN_ENABLE (1)`
Enable/disable CAN transport layer.
- `#define BOOT_COM_CAN_BAUDRATE (500000)`
Configure the desired CAN baudrate.
- `#define BOOT_COM_CAN_TX_MSG_ID (0x7E1 /*| 0x80000000*/)`

- Configure CAN message ID target->host.*

 - #define `BOOT_COM_CAN_TX_MAX_DATA` (8)

Configure number of bytes in the target->host CAN message.
- #define `BOOT_COM_CAN_RX_MSG_ID` (0x667 /*| 0x80000000*/)

Configure CAN message ID host->target.
- #define `BOOT_COM_CAN_RX_MAX_DATA` (8)

Configure number of bytes in the host->target CAN message.
- #define `BOOT_COM_CAN_CHANNEL_INDEX` (0)

Select the desired CAN peripheral as a zero based index.
- #define `BOOT_COM_RS232_ENABLE` (1)

Enable/disable UART transport layer.
- #define `BOOT_COM_RS232_BAUDRATE` (57600)

Configure the desired communication speed.
- #define `BOOT_COM_RS232_TX_MAX_DATA` (64)

Configure number of bytes in the target->host data packet.
- #define `BOOT_COM_RS232_RX_MAX_DATA` (64)

Configure number of bytes in the host->target data packet.
- #define `BOOT_COM_RS232_CHANNEL_INDEX` (2)

Select the desired UART peripheral as a zero based index.
- #define `BOOT_BACKDOOR_HOOKS_ENABLE` (0)

Enable/disable the backdoor override hook functions.
- #define `BOOT_NVM_HOOKS_ENABLE` (0)

Enable/disable the NVM hook function for supporting additional memory devices.
- #define `BOOT_NVM_SIZE_KB` (2048)

Configure the size of the default memory device (typically flash EEPROM).
- #define `BOOT_NVM_CHECKSUM_HOOKS_ENABLE` (0)

Enable/disable hooks functions to override the user program checksum handling.
- #define `BOOT_COP_HOOKS_ENABLE` (1)

Enable/disable the hook functions for controlling the watchdog.

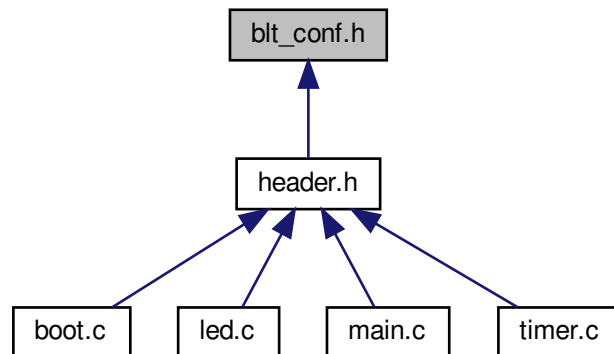
7.151.1 Detailed Description

Bootloader configuration header file.

7.152 blt_conf.h File Reference

Bootloader configuration header file.

This graph shows which files directly or indirectly include this file:



Macros

- `#define BOOT_CPU_XTAL_SPEED_KHZ (8000)`
Frequency of the external crystal oscillator.
- `#define BOOT_CPU_SYSTEM_SPEED_KHZ (480000)`
Desired system speed.
- `#define BOOT_CPU_BYTE_ORDER_MOTOROLA (0)`
Motorola or Intel style byte ordering.
- `#define BOOT_CPU_USER_PROGRAM_START_HOOK (1)`
Enable/disable hook function call right before user program start.
- `#define BOOT_COM_USB_ENABLE (1)`
Enable/disable USB transport layer.
- `#define BOOT_COM_USB_TX_MAX_DATA (63)`
Configure number of bytes in the target->host data packet.
- `#define BOOT_COM_USB_RX_MAX_DATA (63)`
Configure number of bytes in the host->target data packet.
- `#define BOOT_COM_CAN_ENABLE (1)`
Enable/disable CAN transport layer.
- `#define BOOT_COM_CAN_BAUDRATE (500000)`
Configure the desired CAN baudrate.
- `#define BOOT_COM_CAN_TX_MSG_ID (0x7E1 /*| 0x80000000*/)`
Configure CAN message ID target->host.
- `#define BOOT_COM_CAN_TX_MAX_DATA (8)`
Configure number of bytes in the target->host CAN message.
- `#define BOOT_COM_CAN_RX_MSG_ID (0x667 /*| 0x80000000*/)`
Configure CAN message ID host->target.
- `#define BOOT_COM_CAN_RX_MAX_DATA (8)`
Configure number of bytes in the host->target CAN message.
- `#define BOOT_COM_CAN_CHANNEL_INDEX (0)`
Select the desired CAN peripheral as a zero based index.
- `#define BOOT_COM_RS232_ENABLE (1)`

- *Enable/disable UART transport layer.*
- #define [BOOT_COM_RS232_BAUDRATE](#) (57600)
- *Configure the desired communication speed.*
- #define [BOOT_COM_RS232_TX_MAX_DATA](#) (64)
- *Configure number of bytes in the target->host data packet.*
- #define [BOOT_COM_RS232_RX_MAX_DATA](#) (64)
- *Configure number of bytes in the host->target data packet.*
- #define [BOOT_COM_RS232_CHANNEL_INDEX](#) (2)
- *Select the desired UART peripheral as a zero based index.*
- #define [BOOT_BACKDOOR_HOOKS_ENABLE](#) (0)
- *Enable/disable the backdoor override hook functions.*
- #define [BOOT_NVM_HOOKS_ENABLE](#) (0)
- *Enable/disable the NVM hook function for supporting additional memory devices.*
- #define [BOOT_NVM_SIZE_KB](#) (2048)
- *Configure the size of the default memory device (typically flash EEPROM).*
- #define [BOOT_NVM_CHECKSUM_HOOKS_ENABLE](#) (0)
- *Enable/disable hooks functions to override the user program checksum handling.*
- #define [BOOT_COP_HOOKS_ENABLE](#) (1)
- *Enable/disable the hook functions for controlling the watchdog.*

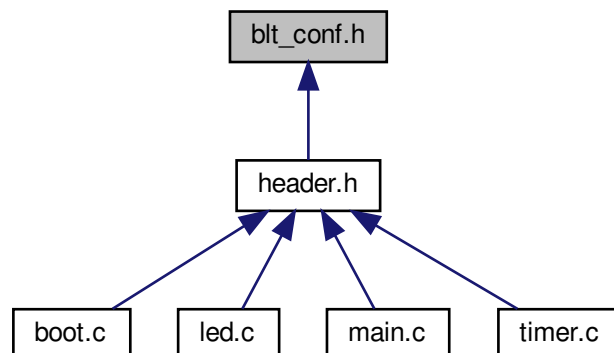
7.152.1 Detailed Description

Bootloader configuration header file.

7.153 blt_conf.h File Reference

Bootloader configuration header file.

This graph shows which files directly or indirectly include this file:



Macros

- #define [BOOT_CPU_XTAL_SPEED_KHZ](#) (8000)
Frequency of the external crystal oscillator.
- #define [BOOT_CPU_SYSTEM_SPEED_KHZ](#) (480000)
Desired system speed.
- #define [BOOT_CPU_BYTE_ORDER_MOTOROLA](#) (0)
Motorola or Intel style byte ordering.
- #define [BOOT_CPU_USER_PROGRAM_START_HOOK](#) (1)
Enable/disable hook function call right before user program start.
- #define [BOOT_COM_USB_ENABLE](#) (1)
Enable/disable USB transport layer.
- #define [BOOT_COM_USB_TX_MAX_DATA](#) (63)
Configure number of bytes in the target->host data packet.
- #define [BOOT_COM_USB_RX_MAX_DATA](#) (63)
Configure number of bytes in the host->target data packet.
- #define [BOOT_COM_CAN_ENABLE](#) (1)
Enable/disable CAN transport layer.
- #define [BOOT_COM_CAN_BAUDRATE](#) (500000)
Configure the desired CAN baudrate.
- #define [BOOT_COM_CAN_TX_MSG_ID](#) (0x7E1 /*| 0x80000000*/) *Configure CAN message ID target->host.*
- #define [BOOT_COM_CAN_TX_MAX_DATA](#) (8)
Configure number of bytes in the target->host CAN message.
- #define [BOOT_COM_CAN_RX_MSG_ID](#) (0x667 /*| 0x80000000*/) *Configure CAN message ID host->target.*
- #define [BOOT_COM_CAN_RX_MAX_DATA](#) (8)
Configure number of bytes in the host->target CAN message.
- #define [BOOT_COM_CAN_CHANNEL_INDEX](#) (0)
Select the desired CAN peripheral as a zero based index.
- #define [BOOT_COM_RS232_ENABLE](#) (1)
Enable/disable UART transport layer.
- #define [BOOT_COM_RS232_BAUDRATE](#) (57600)
Configure the desired communication speed.
- #define [BOOT_COM_RS232_TX_MAX_DATA](#) (64)
Configure number of bytes in the target->host data packet.
- #define [BOOT_COM_RS232_RX_MAX_DATA](#) (64)
Configure number of bytes in the host->target data packet.
- #define [BOOT_COM_RS232_CHANNEL_INDEX](#) (2)
Select the desired UART peripheral as a zero based index.
- #define [BOOT_BACKDOOR_HOOKS_ENABLE](#) (0)
Enable/disable the backdoor override hook functions.
- #define [BOOT_NVM_HOOKS_ENABLE](#) (0)
Enable/disable the NVM hook function for supporting additional memory devices.
- #define [BOOT_NVM_SIZE_KB](#) (2048)
Configure the size of the default memory device (typically flash EEPROM).
- #define [BOOT_NVM_CHECKSUM_HOOKS_ENABLE](#) (0)
Enable/disable hooks functions to override the user program checksum handling.
- #define [BOOT_COP_HOOKS_ENABLE](#) (1)
Enable/disable the hook functions for controlling the watchdog.

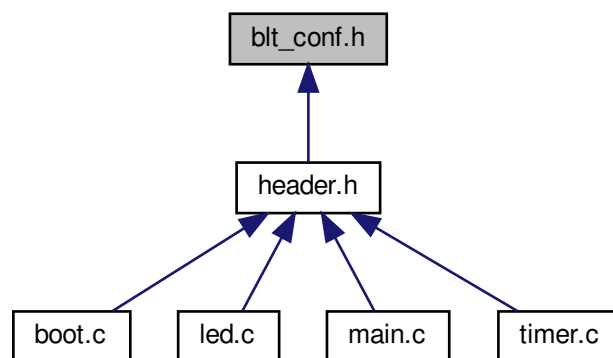
7.153.1 Detailed Description

Bootloader configuration header file.

7.154 blt_conf.h File Reference

Bootloader configuration header file.

This graph shows which files directly or indirectly include this file:



Macros

- `#define BOOT_CPU_XTAL_SPEED_KHZ (8000)`
Frequency of the external crystal oscillator.
- `#define BOOT_CPU_SYSTEM_SPEED_KHZ (480000)`
Desired system speed.
- `#define BOOT_CPU_BYTE_ORDER_MOTOROLA (0)`
Motorola or Intel style byte ordering.
- `#define BOOT_CPU_USER_PROGRAM_START_HOOK (1)`
Enable/disable hook function call right before user program start.
- `#define BOOT_COM_USB_ENABLE (1)`
Enable/disable USB transport layer.
- `#define BOOT_COM_USB_TX_MAX_DATA (63)`
Configure number of bytes in the target->host data packet.
- `#define BOOT_COM_USB_RX_MAX_DATA (63)`
Configure number of bytes in the host->target data packet.
- `#define BOOT_COM_CAN_ENABLE (1)`
Enable/disable CAN transport layer.
- `#define BOOT_COM_CAN_BAUDRATE (500000)`
Configure the desired CAN baudrate.
- `#define BOOT_COM_CAN_TX_MSG_ID (0x7E1 /*| 0x80000000*/)`

- Configure CAN message ID target->host.*

 - #define `BOOT_COM_CAN_TX_MAX_DATA` (8)

Configure number of bytes in the target->host CAN message.
- #define `BOOT_COM_CAN_RX_MSG_ID` (0x667 /*| 0x80000000*/)

Configure CAN message ID host->target.
- #define `BOOT_COM_CAN_RX_MAX_DATA` (8)

Configure number of bytes in the host->target CAN message.
- #define `BOOT_COM_CAN_CHANNEL_INDEX` (0)

Select the desired CAN peripheral as a zero based index.
- #define `BOOT_COM_RS232_ENABLE` (1)

Enable/disable UART transport layer.
- #define `BOOT_COM_RS232_BAUDRATE` (57600)

Configure the desired communication speed.
- #define `BOOT_COM_RS232_TX_MAX_DATA` (64)

Configure number of bytes in the target->host data packet.
- #define `BOOT_COM_RS232_RX_MAX_DATA` (64)

Configure number of bytes in the host->target data packet.
- #define `BOOT_COM_RS232_CHANNEL_INDEX` (2)

Select the desired UART peripheral as a zero based index.
- #define `BOOT_BACKDOOR_HOOKS_ENABLE` (0)

Enable/disable the backdoor override hook functions.
- #define `BOOT_NVM_HOOKS_ENABLE` (0)

Enable/disable the NVM hook function for supporting additional memory devices.
- #define `BOOT_NVM_SIZE_KB` (2048)

Configure the size of the default memory device (typically flash EEPROM).
- #define `BOOT_NVM_CHECKSUM_HOOKS_ENABLE` (0)

Enable/disable hooks functions to override the user program checksum handling.
- #define `BOOT_COP_HOOKS_ENABLE` (1)

Enable/disable the hook functions for controlling the watchdog.

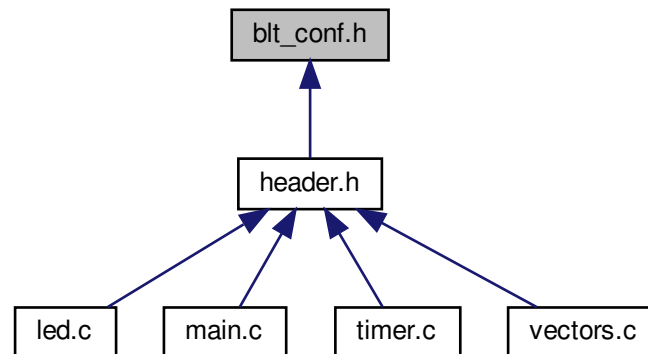
7.154.1 Detailed Description

Bootloader configuration header file.

7.155 blt_conf.h File Reference

Bootloader configuration header file.

This graph shows which files directly or indirectly include this file:



Macros

- `#define BOOT_CPU_XTAL_SPEED_KHZ (8000)`
Frequency of the external crystal oscillator.
- `#define BOOT_CPU_SYSTEM_SPEED_KHZ (24000)`
Desired system speed.
- `#define BOOT_CPU_BYTE_ORDER_MOTOROLA (1)`
Motorola or Intel style byte ordering.
- `#define BOOT_CPU_USER_PROGRAM_START_HOOK (1)`
Enable/disable hook function call right before user program start.
- `#define BOOT_COM_CAN_ENABLE (1)`
Enable/disable CAN transport layer.
- `#define BOOT_COM_CAN_BAUDRATE (500000)`
Configure the desired CAN baudrate.
- `#define BOOT_COM_CAN_TX_MSG_ID (0x7E1 /*| 0x80000000*/)`
Configure CAN message ID target->host.
- `#define BOOT_COM_CAN_TX_MAX_DATA (8)`
Configure number of bytes in the target->host CAN message.
- `#define BOOT_COM_CAN_RX_MSG_ID (0x667 /*| 0x80000000*/)`
Configure CAN message ID host->target.
- `#define BOOT_COM_CAN_RX_MAX_DATA (8)`
Configure number of bytes in the host->target CAN message.
- `#define BOOT_COM_CAN_CHANNEL_INDEX (0)`
Select the desired CAN peripheral as a zero based index.
- `#define BOOT_COM_RS232_ENABLE (0)`
Enable/disable UART transport layer.
- `#define BOOT_COM_RS232_BAUDRATE (57600)`
Configure the desired communication speed.
- `#define BOOT_COM_RS232_TX_MAX_DATA (64)`
Configure number of bytes in the target->host data packet.
- `#define BOOT_COM_RS232_RX_MAX_DATA (64)`

- *Configure number of bytes in the host->target data packet.*
- `#define BOOT_COM_RS232_CHANNEL_INDEX (0)`
Select the desired UART peripheral as a zero based index.
- `#define BOOT_BACKDOOR_HOOKS_ENABLE (0)`
Enable/disable the backdoor override hook functions.
- `#define BOOT_NVM_HOOKS_ENABLE (0)`
Enable/disable the NVM hook function for supporting additional memory devices.
- `#define BOOT_NVM_SIZE_KB (128)`
Configure the size of the default memory device (typically flash EEPROM).
- `#define BOOT_NVM_CHECKSUM_HOOKS_ENABLE (0)`
Enable/disable hooks functions to override the user program checksum handling.
- `#define BOOT_COP_HOOKS_ENABLE (1)`
Enable/disable the hook functions for controlling the watchdog.

7.155.1 Detailed Description

Bootloader configuration header file.

7.156 blt_conf.h File Reference

Bootloader configuration header file.

This graph shows which files directly or indirectly include this file:



Macros

- `#define BOOT_CPU_XTAL_SPEED_KHZ (8000)`
Frequency of the external crystal oscillator.
- `#define BOOT_CPU_SYSTEM_SPEED_KHZ (24000)`
Desired system speed.
- `#define BOOT_CPU_BYTE_ORDER_MOTOROLA (1)`
Motorola or Intel style byte ordering.
- `#define BOOT_CPU_USER_PROGRAM_START_HOOK (0)`
Enable/disable hook function call right before user program start.
- `#define BOOT_COM_CAN_ENABLE (1)`
Enable/disable CAN transport layer.
- `#define BOOT_COM_CAN_BAUDRATE (500000)`
Configure the desired CAN baudrate.
- `#define BOOT_COM_CAN_TX_MSG_ID (0x7E1 /*| 0x80000000*/)`
Configure CAN message ID target->host.
- `#define BOOT_COM_CAN_TX_MAX_DATA (8)`
Configure number of bytes in the target->host CAN message.
- `#define BOOT_COM_CAN_RX_MSG_ID (0x667 /*| 0x80000000*/)`
Configure CAN message ID host->target.

- #define `BOOT_COM_CAN_RX_MAX_DATA` (8)
Configure number of bytes in the host->target CAN message.
- #define `BOOT_COM_CAN_CHANNEL_INDEX` (0)
Select the desired CAN peripheral as a zero based index.
- #define `BOOT_COM_RS232_ENABLE` (1)
Enable/disable UART transport layer.
- #define `BOOT_COM_RS232_BAUDRATE` (57600)
Configure the desired communication speed.
- #define `BOOT_COM_RS232_TX_MAX_DATA` (64)
Configure number of bytes in the target->host data packet.
- #define `BOOT_COM_RS232_RX_MAX_DATA` (64)
Configure number of bytes in the host->target data packet.
- #define `BOOT_COM_RS232_CHANNEL_INDEX` (0)
Select the desired UART peripheral as a zero based index.
- #define `BOOT_FILE_SYS_ENABLE` (0)
Enable/disable support for firmware updates from a locally attached storage.
- #define `BOOT_FILE_LOGGING_ENABLE` (0)
Enable/disable logging messages during firmware updates.
- #define `BOOT_FILE_ERROR_HOOK_ENABLE` (0)
Enable/disable a hook function that is called upon detection of an error.
- #define `BOOT_FILE_STARTED_HOOK_ENABLE` (0)
Enable/disable a hook function that is called at the start of the update.
- #define `BOOT_FILE_COMPLETED_HOOK_ENABLE` (0)
Enable/disable a hook function that is called at the end of the update.
- #define `BOOT_BACKDOOR_HOOKS_ENABLE` (0)
Enable/disable the backdoor override hook functions.
- #define `BOOT_NVM_HOOKS_ENABLE` (0)
Enable/disable the NVM hook function for supporting additional memory devices.
- #define `BOOT_NVM_SIZE_KB` (256)
Configure the size of the default memory device (typically flash EEPROM).
- #define `BOOT_NVM_CHECKSUM_HOOKS_ENABLE` (0)
Enable/disable hooks functions to override the user program checksum handling.
- #define `BOOT_COP_HOOKS_ENABLE` (0)
Enable/disable the hook functions for controlling the watchdog.

7.156.1 Detailed Description

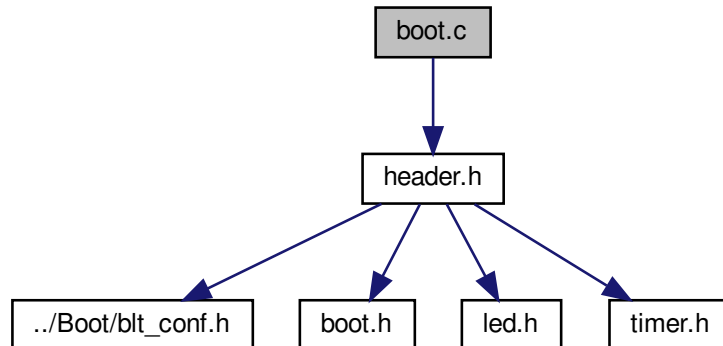
Bootloader configuration header file.

7.157 boot.c File Reference

Demo program bootloader interface source file.

```
#include "header.h"
```

Include dependency graph for Demo/_template/Prog/boot.c:



Macros

- `#define RS232_CTO_RX_PACKET_TIMEOUT_MS (100u)`
Timeout time for the reception of a CTO packet. The timer is started upon reception of the first packet byte.

Functions

- static void `BootComRs232Init` (void)
Initializes the UART communication interface.
- static void `BootComRs232CheckActivationRequest` (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void `BootComInit` (void)
Initializes the communication interface.
- void `BootComCheckActivationRequest` (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void `BootActivate` (void)
Bootloader activation function.
- static unsigned char `Rs232ReceiveByte` (unsigned char *data)
Receives a communication interface byte if one is present.

7.157.1 Detailed Description

Demo program bootloader interface source file.

7.157.2 Function Documentation

7.157.2.1 BootActivate()

```
void BootActivate (  
    void )
```

Bootloader activation function.

Returns

none.

Referenced by BootComRs232CheckActivationRequest(), and NetApp().

7.157.2.2 BootComCheckActivationRequest()

```
void BootComCheckActivationRequest (  
    void )
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

Referenced by AppTask(), and main().

7.157.2.3 BootComInit()

```
void BootComInit (  
    void )
```

Initializes the communication interface.

Returns

none.

Referenced by AppInit(), and main().

7.157.2.4 BootComRs232CheckActivationRequest()

```
static void BootComRs232CheckActivationRequest (
    void ) [static]
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

Referenced by BootComCheckActivationRequest().

7.157.2.5 BootComRs232Init()

```
static void BootComRs232Init (
    void ) [static]
```

Initializes the UART communication interface.

Returns

none.

Referenced by BootComInit().

7.157.2.6 Rs232ReceiveByte()

```
static unsigned char Rs232ReceiveByte (
    unsigned char * data ) [static]
```

Receives a communication interface byte if one is present.

Parameters

<i>data</i>	Pointer to byte where the data is to be stored.
-------------	---

Returns

1 if a byte was received, 0 otherwise.

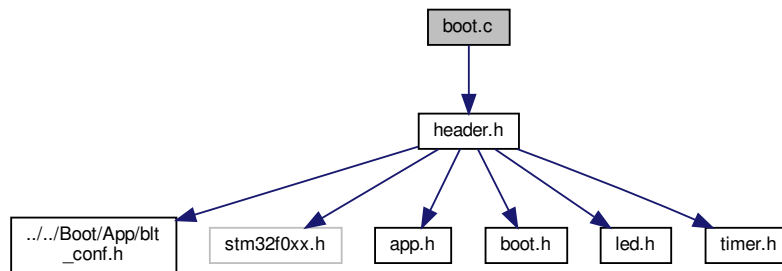
Referenced by BootComRs232CheckActivationRequest().

7.158 boot.c File Reference

Demo program bootloader interface source file.

```
#include "header.h"
```

Include dependency graph for Demo/ARMCM0_STM32F0_Discovery_STM32F051_CubeIDE/Prog/App/boot.c:



Macros

- `#define RS232_CTO_RX_PACKET_TIMEOUT_MS (100u)`
Timeout time for the reception of a CTO packet. The timer is started upon reception of the first packet byte.

Functions

- static void `BootComRs232Init` (void)
Initializes the UART communication interface.
- static void `BootComRs232CheckActivationRequest` (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void `BootComInit` (void)
Initializes the communication interface.
- void `BootComCheckActivationRequest` (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void `BootActivate` (void)
Bootloader activation function.
- static unsigned char `Rs232ReceiveByte` (unsigned char *data)
Receives a communication interface byte if one is present.

Variables

- static UART_HandleTypeDef `rs232Handle`
UART handle to be used in API calls.

7.158.1 Detailed Description

Demo program bootloader interface source file.

7.158.2 Function Documentation

7.158.2.1 BootActivate()

```
void BootActivate (
    void )
```

Bootloader activation function.

Returns

none.

Referenced by BootComRs232CheckActivationRequest().

7.158.2.2 BootComCheckActivationRequest()

```
void BootComCheckActivationRequest (
    void )
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.158.2.3 BootComInit()

```
void BootComInit (
    void )
```

Initializes the communication interface.

Returns

none.

7.158.2.4 BootComRs232CheckActivationRequest()

```
static void BootComRs232CheckActivationRequest (  
    void ) [static]
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

Referenced by BootComCheckActivationRequest().

7.158.2.5 BootComRs232Init()

```
static void BootComRs232Init (  
    void ) [static]
```

Initializes the UART communication interface.

Returns

none.

Referenced by BootComInit().

7.158.2.6 Rs232ReceiveByte()

```
static unsigned char Rs232ReceiveByte (  
    unsigned char * data ) [static]
```

Receives a communication interface byte if one is present.

Parameters

<i>data</i>	Pointer to byte where the data is to be stored.
-------------	---

Returns

1 if a byte was received, 0 otherwise.

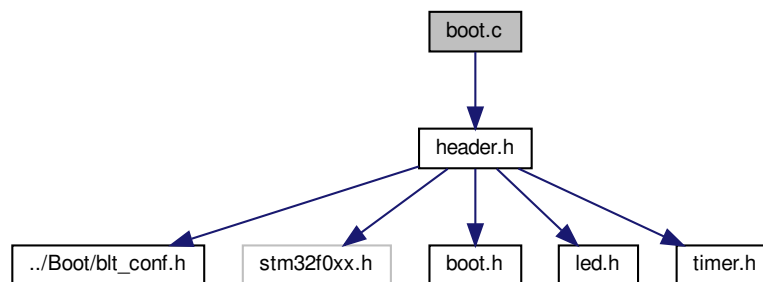
Referenced by BootComRs232CheckActivationRequest().

7.159 boot.c File Reference

Demo program bootloader interface source file.

```
#include "header.h"
```

Include dependency graph for Demo/ARMCM0_STM32F0_Discovery_STM32F051_GCC/Prog/boot.c:



Macros

- `#define RS232_CTO_RX_PACKET_TIMEOUT_MS (100u)`
Timeout time for the reception of a CTO packet. The timer is started upon reception of the first packet byte.

Functions

- static void `BootComRs232Init` (void)
Initializes the UART communication interface.
- static void `BootComRs232CheckActivationRequest` (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void `BootComInit` (void)
Initializes the communication interface.
- void `BootComCheckActivationRequest` (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void `BootActivate` (void)
Bootloader activation function.
- static unsigned char `Rs232ReceiveByte` (unsigned char *data)
Receives a communication interface byte if one is present.

Variables

- static UART_HandleTypeDef `rs232Handle`
UART handle to be used in API calls.

7.159.1 Detailed Description

Demo program bootloader interface source file.

7.159.2 Function Documentation

7.159.2.1 BootActivate()

```
void BootActivate (
    void )
```

Bootloader activation function.

Returns

none.

Referenced by BootComRs232CheckActivationRequest().

7.159.2.2 BootComCheckActivationRequest()

```
void BootComCheckActivationRequest (
    void )
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.159.2.3 BootComInit()

```
void BootComInit (
    void )
```

Initializes the communication interface.

Returns

none.

7.159.2.4 BootComRs232CheckActivationRequest()

```
static void BootComRs232CheckActivationRequest (
    void ) [static]
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

Referenced by BootComCheckActivationRequest().

7.159.2.5 BootComRs232Init()

```
static void BootComRs232Init (
    void ) [static]
```

Initializes the UART communication interface.

Returns

none.

Referenced by BootComInit().

7.159.2.6 Rs232ReceiveByte()

```
static unsigned char Rs232ReceiveByte (
    unsigned char * data ) [static]
```

Receives a communication interface byte if one is present.

Parameters

<i>data</i>	Pointer to byte where the data is to be stored.
-------------	---

Returns

1 if a byte was received, 0 otherwise.

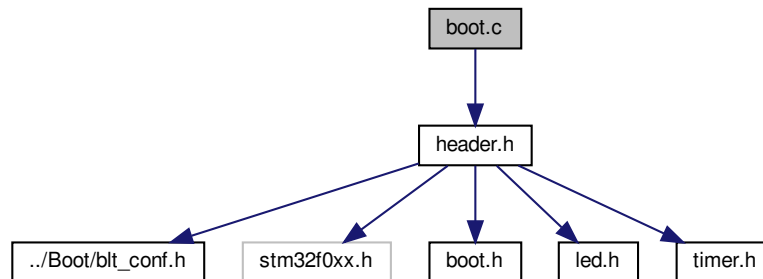
Referenced by BootComRs232CheckActivationRequest().

7.160 boot.c File Reference

Demo program bootloader interface source file.

```
#include "header.h"
```

Include dependency graph for Demo/ARMCM0_STM32F0_Discovery_STM32F051_IAR/Prog/boot.c:



Macros

- `#define RS232_CTO_RX_PACKET_TIMEOUT_MS (100u)`
Timeout time for the reception of a CTO packet. The timer is started upon reception of the first packet byte.

Functions

- static void `BootComRs232Init` (void)
Initializes the UART communication interface.
- static void `BootComRs232CheckActivationRequest` (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void `BootComInit` (void)
Initializes the communication interface.
- void `BootComCheckActivationRequest` (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void `BootActivate` (void)
Bootloader activation function.
- static unsigned char `Rs232ReceiveByte` (unsigned char *data)
Receives a communication interface byte if one is present.

Variables

- static UART_HandleTypeDef `rs232Handle`
UART handle to be used in API calls.

7.160.1 Detailed Description

Demo program bootloader interface source file.

7.160.2 Function Documentation

7.160.2.1 BootActivate()

```
void BootActivate (
    void )
```

Bootloader activation function.

Returns

none.

Referenced by BootComRs232CheckActivationRequest().

7.160.2.2 BootComCheckActivationRequest()

```
void BootComCheckActivationRequest (
    void )
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.160.2.3 BootComInit()

```
void BootComInit (
    void )
```

Initializes the communication interface.

Returns

none.

7.160.2.4 BootComRs232CheckActivationRequest()

```
static void BootComRs232CheckActivationRequest (  
    void ) [static]
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

Referenced by BootComCheckActivationRequest().

7.160.2.5 BootComRs232Init()

```
static void BootComRs232Init (  
    void ) [static]
```

Initializes the UART communication interface.

Returns

none.

Referenced by BootComInit().

7.160.2.6 Rs232ReceiveByte()

```
static unsigned char Rs232ReceiveByte (  
    unsigned char * data ) [static]
```

Receives a communication interface byte if one is present.

Parameters

<i>data</i>	Pointer to byte where the data is to be stored.
-------------	---

Returns

1 if a byte was received, 0 otherwise.

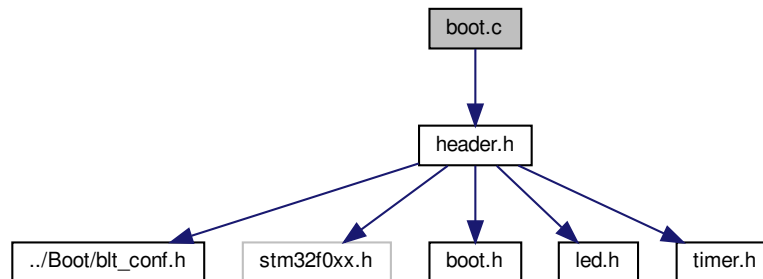
Referenced by BootComRs232CheckActivationRequest().

7.161 boot.c File Reference

Demo program bootloader interface source file.

```
#include "header.h"
```

Include dependency graph for Demo/ARMCM0_STM32F0_Discovery_STM32F051_Keil/Prog/boot.c:



Macros

- `#define RS232_CTO_RX_PACKET_TIMEOUT_MS (100u)`
Timeout time for the reception of a CTO packet. The timer is started upon reception of the first packet byte.

Functions

- static void `BootComRs232Init` (void)
Initializes the UART communication interface.
- static void `BootComRs232CheckActivationRequest` (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void `BootComInit` (void)
Initializes the communication interface.
- void `BootComCheckActivationRequest` (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void `BootActivate` (void)
Bootloader activation function.
- static unsigned char `Rs232ReceiveByte` (unsigned char *data)
Receives a communication interface byte if one is present.

Variables

- static UART_HandleTypeDef `rs232Handle`
UART handle to be used in API calls.

7.161.1 Detailed Description

Demo program bootloader interface source file.

7.161.2 Function Documentation

7.161.2.1 BootActivate()

```
void BootActivate (
    void )
```

Bootloader activation function.

Returns

none.

Referenced by BootComRs232CheckActivationRequest().

7.161.2.2 BootComCheckActivationRequest()

```
void BootComCheckActivationRequest (
    void )
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.161.2.3 BootComInit()

```
void BootComInit (
    void )
```

Initializes the communication interface.

Returns

none.

7.161.2.4 BootComRs232CheckActivationRequest()

```
static void BootComRs232CheckActivationRequest (
    void ) [static]
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

Referenced by BootComCheckActivationRequest().

7.161.2.5 BootComRs232Init()

```
static void BootComRs232Init (
    void ) [static]
```

Initializes the UART communication interface.

Returns

none.

Referenced by BootComInit().

7.161.2.6 Rs232ReceiveByte()

```
static unsigned char Rs232ReceiveByte (
    unsigned char * data ) [static]
```

Receives a communication interface byte if one is present.

Parameters

<i>data</i>	Pointer to byte where the data is to be stored.
-------------	---

Returns

1 if a byte was received, 0 otherwise.

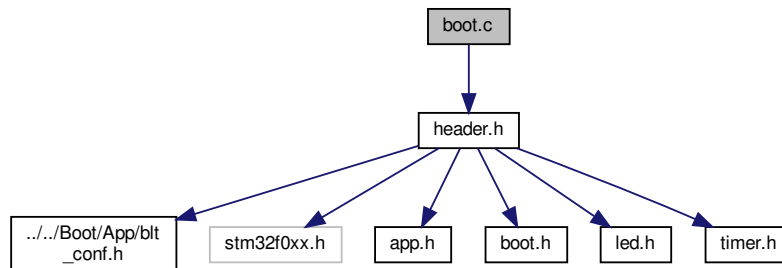
Referenced by BootComRs232CheckActivationRequest().

7.162 boot.c File Reference

Demo program bootloader interface source file.

```
#include "header.h"
```

Include dependency graph for Demo/ARMCM0_STM32F0_Nucleo_F091RC_CubeIDE/Prog/App/boot.c:



Data Structures

- struct [tCanBusTiming](#)
Structure type for grouping CAN bus timing related information.

Macros

- #define [RS232_CTO_RX_PACKET_TIMEOUT_MS](#) (100u)
Timeout time for the reception of a CTO packet. The timer is started upon reception of the first packet byte.

Functions

- static void [BootComRs232Init](#) (void)
Initializes the UART communication interface.
- static void [BootComRs232CheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- static void [BootComCanInit](#) (void)
Initializes the CAN communication interface.
- static void [BootComCanCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootComInit](#) (void)
Initializes the communication interface.
- void [BootComCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootActivate](#) (void)
Bootloader activation function.
- static unsigned char [Rs232ReceiveByte](#) (unsigned char *data)
Receives a communication interface byte if one is present.
- static unsigned char [CanGetSpeedConfig](#) (unsigned short baud, unsigned short *prescaler, unsigned char *tseg1, unsigned char *tseg2)
Search algorithm to match the desired baudrate to a possible bus timing configuration.

Variables

- static UART_HandleTypeDef [rs232Handle](#)
UART handle to be used in API calls.
- static const [tCanBusTiming](#) [canTiming](#) []
CAN bittiming table for dynamically calculating the bittiming settings.
- static CAN_HandleTypeDef [canHandle](#)
CAN handle to be used in API calls.

7.162.1 Detailed Description

Demo program bootloader interface source file.

7.162.2 Function Documentation

7.162.2.1 BootActivate()

```
void BootActivate (  
    void )
```

Bootloader activation function.

Returns

none.

Referenced by [BootComCanCheckActivationRequest\(\)](#), and [BootComRs232CheckActivationRequest\(\)](#).

7.162.2.2 BootComCanCheckActivationRequest()

```
static void BootComCanCheckActivationRequest (  
    void ) [static]
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

Referenced by [BootComCheckActivationRequest\(\)](#).

7.162.2.3 BootComCanInit()

```
static void BootComCanInit (
    void ) [static]
```

Initializes the CAN communication interface.

Returns

none.

Referenced by BootComInit().

7.162.2.4 BootComCheckActivationRequest()

```
void BootComCheckActivationRequest (
    void )
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.162.2.5 BootComInit()

```
void BootComInit (
    void )
```

Initializes the communication interface.

Returns

none.

7.162.2.6 BootComRs232CheckActivationRequest()

```
static void BootComRs232CheckActivationRequest (
    void ) [static]
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

Referenced by BootComCheckActivationRequest().

7.162.2.7 BootComRs232Init()

```
static void BootComRs232Init (  
    void ) [static]
```

Initializes the UART communication interface.

Returns

none.

Referenced by BootComInit().

7.162.2.8 CanGetSpeedConfig()

```
static unsigned char CanGetSpeedConfig (  
    unsigned short baud,  
    unsigned short * prescaler,  
    unsigned char * tseg1,  
    unsigned char * tseg2 ) [static]
```

Search algorithm to match the desired baudrate to a possible bus timing configuration.

Parameters

<i>baud</i>	The desired baudrate in kbps. Valid values are 10..1000.
<i>prescaler</i>	Pointer to where the value for the prescaler will be stored.
<i>tseg1</i>	Pointer to where the value for TSEG2 will be stored.
<i>tseg2</i>	Pointer to where the value for TSEG2 will be stored.

Returns

1 if the CAN bustiming register values were found, 0 otherwise.

Referenced by BootComCanInit().

7.162.2.9 Rs232ReceiveByte()

```
static unsigned char Rs232ReceiveByte (  
    unsigned char * data ) [static]
```

Receives a communication interface byte if one is present.

Parameters

<i>data</i>	Pointer to byte where the data is to be stored.
-------------	---

Returns

1 if a byte was received, 0 otherwise.

Referenced by `BootComRs232CheckActivationRequest()`.

7.162.3 Variable Documentation**7.162.3.1 canTiming**

```
const tCanBusTiming canTiming[] [static]
```

Initial value:

```
=
{
    { 5, 2 },
    { 6, 2 },
    { 6, 3 },
    { 7, 3 },
    { 8, 3 },
    { 9, 3 },
    { 9, 4 },
    { 10, 4 },
    { 11, 4 },
    { 12, 4 },
    { 12, 5 },
    { 13, 5 },
    { 14, 5 },
    { 15, 5 },
    { 15, 6 },
    { 16, 6 },
    { 16, 7 },
    { 16, 8 }
}
```

CAN bittiming table for dynamically calculating the bittiming settings.

According to the CAN protocol 1 bit-time can be made up of between 8..25 time quanta (TQ). The total TQ in a bit is SYNC + TSEG1 + TSEG2 with SYNC always being 1. The sample point is (SYNC + TSEG1) / (SYNC + TSEG1 + SEG2) * 100%. This array contains possible and valid time quanta configurations with a sample point between 68..78%.

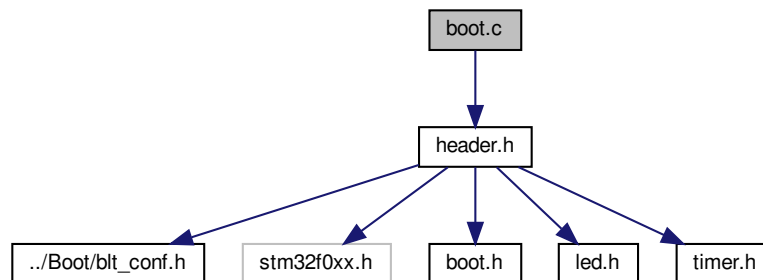
Referenced by `CanGetSpeedConfig()`.

7.163 boot.c File Reference

Demo program bootloader interface source file.

```
#include "header.h"
```

Include dependency graph for Demo/ARMCM0_STM32F0_Nucleo_F091RC_GCC/Prog/boot.c:



Data Structures

- struct [tCanBusTiming](#)
Structure type for grouping CAN bus timing related information.

Macros

- `#define RS232_CTO_RX_PACKET_TIMEOUT_MS (100u)`
Timeout time for the reception of a CTO packet. The timer is started upon reception of the first packet byte.

Functions

- static void [BootComRs232Init](#) (void)
Initializes the UART communication interface.
- static void [BootComRs232CheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- static void [BootComCanInit](#) (void)
Initializes the CAN communication interface.
- static void [BootComCanCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootComInit](#) (void)
Initializes the communication interface.
- void [BootComCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootActivate](#) (void)
Bootloader activation function.
- static unsigned char [Rs232ReceiveByte](#) (unsigned char *data)
Receives a communication interface byte if one is present.
- static unsigned char [CanGetSpeedConfig](#) (unsigned short baud, unsigned short *prescaler, unsigned char *tseg1, unsigned char *tseg2)
Search algorithm to match the desired baudrate to a possible bus timing configuration.

Variables

- static UART_HandleTypeDef [rs232Handle](#)
UART handle to be used in API calls.
- static const [tCanBusTiming](#) [canTiming](#) []
CAN bittiming table for dynamically calculating the bittiming settings.
- static CAN_HandleTypeDef [canHandle](#)
CAN handle to be used in API calls.

7.163.1 Detailed Description

Demo program bootloader interface source file.

7.163.2 Function Documentation

7.163.2.1 BootActivate()

```
void BootActivate (  
    void )
```

Bootloader activation function.

Returns

none.

Referenced by [BootComCanCheckActivationRequest\(\)](#), and [BootComRs232CheckActivationRequest\(\)](#).

7.163.2.2 BootComCanCheckActivationRequest()

```
static void BootComCanCheckActivationRequest (  
    void ) [static]
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

Referenced by [BootComCheckActivationRequest\(\)](#).

7.163.2.3 BootComCanInit()

```
static void BootComCanInit (
    void ) [static]
```

Initializes the CAN communication interface.

Returns

none.

Referenced by BootComInit().

7.163.2.4 BootComCheckActivationRequest()

```
void BootComCheckActivationRequest (
    void )
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.163.2.5 BootComInit()

```
void BootComInit (
    void )
```

Initializes the communication interface.

Returns

none.

7.163.2.6 BootComRs232CheckActivationRequest()

```
static void BootComRs232CheckActivationRequest (
    void ) [static]
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

Referenced by BootComCheckActivationRequest().

7.163.2.7 BootComRs232Init()

```
static void BootComRs232Init (  
    void ) [static]
```

Initializes the UART communication interface.

Returns

none.

Referenced by BootComInit().

7.163.2.8 CanGetSpeedConfig()

```
static unsigned char CanGetSpeedConfig (  
    unsigned short baud,  
    unsigned short * prescaler,  
    unsigned char * tseg1,  
    unsigned char * tseg2 ) [static]
```

Search algorithm to match the desired baudrate to a possible bus timing configuration.

Parameters

<i>baud</i>	The desired baudrate in kbps. Valid values are 10..1000.
<i>prescaler</i>	Pointer to where the value for the prescaler will be stored.
<i>tseg1</i>	Pointer to where the value for TSEG2 will be stored.
<i>tseg2</i>	Pointer to where the value for TSEG2 will be stored.

Returns

1 if the CAN bustiming register values were found, 0 otherwise.

Referenced by BootComCanInit().

7.163.2.9 Rs232ReceiveByte()

```
static unsigned char Rs232ReceiveByte (  
    unsigned char * data ) [static]
```

Receives a communication interface byte if one is present.

Parameters

<i>data</i>	Pointer to byte where the data is to be stored.
-------------	---

Returns

1 if a byte was received, 0 otherwise.

Referenced by `BootComRs232CheckActivationRequest()`.

7.163.3 Variable Documentation**7.163.3.1 canTiming**

```
const tCanBusTiming canTiming[] [static]
```

Initial value:

```
=
{
    { 5, 2 },
    { 6, 2 },
    { 6, 3 },
    { 7, 3 },
    { 8, 3 },
    { 9, 3 },
    { 9, 4 },
    { 10, 4 },
    { 11, 4 },
    { 12, 4 },
    { 12, 5 },
    { 13, 5 },
    { 14, 5 },
    { 15, 5 },
    { 15, 6 },
    { 16, 6 },
    { 16, 7 },
    { 16, 8 }
}
```

CAN bittiming table for dynamically calculating the bittiming settings.

According to the CAN protocol 1 bit-time can be made up of between 8..25 time quanta (TQ). The total TQ in a bit is SYNC + TSEG1 + TSEG2 with SYNC always being 1. The sample point is (SYNC + TSEG1) / (SYNC + TSEG1 + SEG2) * 100%. This array contains possible and valid time quanta configurations with a sample point between 68..78%.

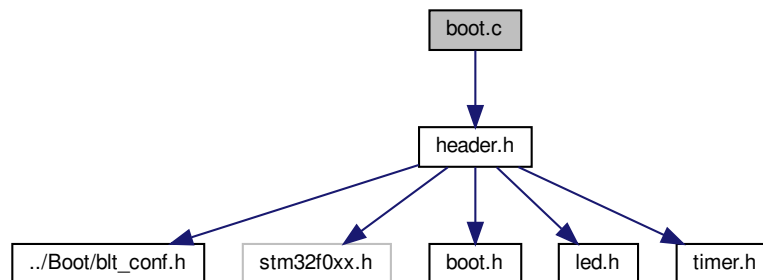
Referenced by `CanGetSpeedConfig()`.

7.164 boot.c File Reference

Demo program bootloader interface source file.

```
#include "header.h"
```

Include dependency graph for Demo/ARMCM0_STM32F0_Nucleo_F091RC_IAR/Prog/boot.c:



Data Structures

- struct [tCanBusTiming](#)
Structure type for grouping CAN bus timing related information.

Macros

- `#define RS232_CTO_RX_PACKET_TIMEOUT_MS (100u)`
Timeout time for the reception of a CTO packet. The timer is started upon reception of the first packet byte.

Functions

- static void [BootComRs232Init](#) (void)
Initializes the UART communication interface.
- static void [BootComRs232CheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- static void [BootComCanInit](#) (void)
Initializes the CAN communication interface.
- static void [BootComCanCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootComInit](#) (void)
Initializes the communication interface.
- void [BootComCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootActivate](#) (void)
Bootloader activation function.
- static unsigned char [Rs232ReceiveByte](#) (unsigned char *data)
Receives a communication interface byte if one is present.
- static unsigned char [CanGetSpeedConfig](#) (unsigned short baud, unsigned short *prescaler, unsigned char *tseg1, unsigned char *tseg2)
Search algorithm to match the desired baudrate to a possible bus timing configuration.

Variables

- static UART_HandleTypeDef [rs232Handle](#)
UART handle to be used in API calls.
- static const [tCanBusTiming](#) [canTiming](#) []
CAN bittiming table for dynamically calculating the bittiming settings.
- static CAN_HandleTypeDef [canHandle](#)
CAN handle to be used in API calls.

7.164.1 Detailed Description

Demo program bootloader interface source file.

7.164.2 Function Documentation

7.164.2.1 BootActivate()

```
void BootActivate (
    void )
```

Bootloader activation function.

Returns

none.

Referenced by [BootComCanCheckActivationRequest\(\)](#), and [BootComRs232CheckActivationRequest\(\)](#).

7.164.2.2 BootComCanCheckActivationRequest()

```
static void BootComCanCheckActivationRequest (
    void ) [static]
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

Referenced by [BootComCheckActivationRequest\(\)](#).

7.164.2.3 BootComCanInit()

```
static void BootComCanInit (
    void ) [static]
```

Initializes the CAN communication interface.

Returns

none.

Referenced by BootComInit().

7.164.2.4 BootComCheckActivationRequest()

```
void BootComCheckActivationRequest (
    void )
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.164.2.5 BootComInit()

```
void BootComInit (
    void )
```

Initializes the communication interface.

Returns

none.

7.164.2.6 BootComRs232CheckActivationRequest()

```
static void BootComRs232CheckActivationRequest (
    void ) [static]
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

Referenced by BootComCheckActivationRequest().

7.164.2.7 BootComRs232Init()

```
static void BootComRs232Init (
    void ) [static]
```

Initializes the UART communication interface.

Returns

none.

Referenced by BootComInit().

7.164.2.8 CanGetSpeedConfig()

```
static unsigned char CanGetSpeedConfig (
    unsigned short baud,
    unsigned short * prescaler,
    unsigned char * tseg1,
    unsigned char * tseg2 ) [static]
```

Search algorithm to match the desired baudrate to a possible bus timing configuration.

Parameters

<i>baud</i>	The desired baudrate in kbps. Valid values are 10..1000.
<i>prescaler</i>	Pointer to where the value for the prescaler will be stored.
<i>tseg1</i>	Pointer to where the value for TSEG2 will be stored.
<i>tseg2</i>	Pointer to where the value for TSEG2 will be stored.

Returns

1 if the CAN bustiming register values were found, 0 otherwise.

Referenced by BootComCanInit().

7.164.2.9 Rs232ReceiveByte()

```
static unsigned char Rs232ReceiveByte (
    unsigned char * data ) [static]
```

Receives a communication interface byte if one is present.

Parameters

<i>data</i>	Pointer to byte where the data is to be stored.
-------------	---

Returns

1 if a byte was received, 0 otherwise.

Referenced by `BootComRs232CheckActivationRequest()`.

7.164.3 Variable Documentation

7.164.3.1 canTiming

```
const tCanBusTiming canTiming[] [static]
```

Initial value:

```
=
{
    { 5, 2 },
    { 6, 2 },
    { 6, 3 },
    { 7, 3 },
    { 8, 3 },
    { 9, 3 },
    { 9, 4 },
    { 10, 4 },
    { 11, 4 },
    { 12, 4 },
    { 12, 5 },
    { 13, 5 },
    { 14, 5 },
    { 15, 5 },
    { 15, 6 },
    { 16, 6 },
    { 16, 7 },
    { 16, 8 }
}
```

CAN bittiming table for dynamically calculating the bittiming settings.

According to the CAN protocol 1 bit-time can be made up of between 8..25 time quanta (TQ). The total TQ in a bit is SYNC + TSEG1 + TSEG2 with SYNC always being 1. The sample point is (SYNC + TSEG1) / (SYNC + TSEG1 + SEG2) * 100%. This array contains possible and valid time quanta configurations with a sample point between 68..78%.

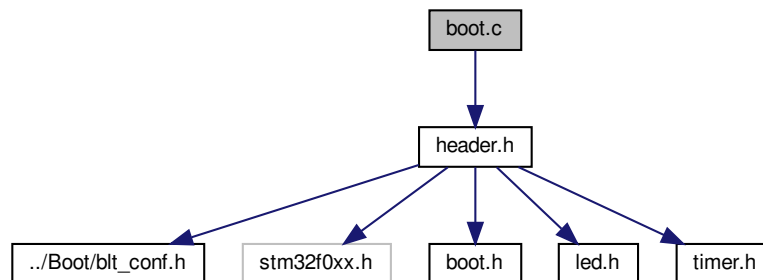
Referenced by `CanGetSpeedConfig()`.

7.165 boot.c File Reference

Demo program bootloader interface source file.

```
#include "header.h"
```

Include dependency graph for Demo/ARMCM0_STM32F0_Nucleo_F091RC_Keil/Prog/boot.c:



Data Structures

- struct [tCanBusTiming](#)
Structure type for grouping CAN bus timing related information.

Macros

- `#define RS232_CTO_RX_PACKET_TIMEOUT_MS (100u)`
Timeout time for the reception of a CTO packet. The timer is started upon reception of the first packet byte.

Functions

- static void [BootComRs232Init](#) (void)
Initializes the UART communication interface.
- static void [BootComRs232CheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- static void [BootComCanInit](#) (void)
Initializes the CAN communication interface.
- static void [BootComCanCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootComInit](#) (void)
Initializes the communication interface.
- void [BootComCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootActivate](#) (void)
Bootloader activation function.
- static unsigned char [Rs232ReceiveByte](#) (unsigned char *data)
Receives a communication interface byte if one is present.
- static unsigned char [CanGetSpeedConfig](#) (unsigned short baud, unsigned short *prescaler, unsigned char *tseg1, unsigned char *tseg2)
Search algorithm to match the desired baudrate to a possible bus timing configuration.

Variables

- static UART_HandleTypeDef [rs232Handle](#)
UART handle to be used in API calls.
- static const [tCanBusTiming](#) [canTiming](#) []
CAN bittiming table for dynamically calculating the bittiming settings.
- static CAN_HandleTypeDef [canHandle](#)
CAN handle to be used in API calls.

7.165.1 Detailed Description

Demo program bootloader interface source file.

7.165.2 Function Documentation

7.165.2.1 BootActivate()

```
void BootActivate (  
    void )
```

Bootloader activation function.

Returns

none.

Referenced by [BootComCanCheckActivationRequest\(\)](#), and [BootComRs232CheckActivationRequest\(\)](#).

7.165.2.2 BootComCanCheckActivationRequest()

```
static void BootComCanCheckActivationRequest (  
    void ) [static]
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

Referenced by [BootComCheckActivationRequest\(\)](#).

7.165.2.3 BootComCanInit()

```
static void BootComCanInit (  
    void ) [static]
```

Initializes the CAN communication interface.

Returns

none.

Referenced by BootComInit().

7.165.2.4 BootComCheckActivationRequest()

```
void BootComCheckActivationRequest (  
    void )
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.165.2.5 BootComInit()

```
void BootComInit (  
    void )
```

Initializes the communication interface.

Returns

none.

7.165.2.6 BootComRs232CheckActivationRequest()

```
static void BootComRs232CheckActivationRequest (  
    void ) [static]
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

Referenced by BootComCheckActivationRequest().

7.165.2.7 BootComRs232Init()

```
static void BootComRs232Init (  
    void ) [static]
```

Initializes the UART communication interface.

Returns

none.

Referenced by BootComInit().

7.165.2.8 CanGetSpeedConfig()

```
static unsigned char CanGetSpeedConfig (  
    unsigned short baud,  
    unsigned short * prescaler,  
    unsigned char * tseg1,  
    unsigned char * tseg2 ) [static]
```

Search algorithm to match the desired baudrate to a possible bus timing configuration.

Parameters

<i>baud</i>	The desired baudrate in kbps. Valid values are 10..1000.
<i>prescaler</i>	Pointer to where the value for the prescaler will be stored.
<i>tseg1</i>	Pointer to where the value for TSEG2 will be stored.
<i>tseg2</i>	Pointer to where the value for TSEG2 will be stored.

Returns

1 if the CAN bustiming register values were found, 0 otherwise.

Referenced by BootComCanInit().

7.165.2.9 Rs232ReceiveByte()

```
static unsigned char Rs232ReceiveByte (  
    unsigned char * data ) [static]
```

Receives a communication interface byte if one is present.

Parameters

<i>data</i>	Pointer to byte where the data is to be stored.
-------------	---

Returns

1 if a byte was received, 0 otherwise.

Referenced by `BootComRs232CheckActivationRequest()`.

7.165.3 Variable Documentation**7.165.3.1 canTiming**

```
const tCanBusTiming canTiming[] [static]
```

Initial value:

```
=
{
    { 5, 2 },
    { 6, 2 },
    { 6, 3 },
    { 7, 3 },
    { 8, 3 },
    { 9, 3 },
    { 9, 4 },
    { 10, 4 },
    { 11, 4 },
    { 12, 4 },
    { 12, 5 },
    { 13, 5 },
    { 14, 5 },
    { 15, 5 },
    { 15, 6 },
    { 16, 6 },
    { 16, 7 },
    { 16, 8 }
}
```

CAN bittiming table for dynamically calculating the bittiming settings.

According to the CAN protocol 1 bit-time can be made up of between 8..25 time quanta (TQ). The total TQ in a bit is SYNC + TSEG1 + TSEG2 with SYNC always being 1. The sample point is (SYNC + TSEG1) / (SYNC + TSEG1 + SEG2) * 100%. This array contains possible and valid time quanta configurations with a sample point between 68..78%.

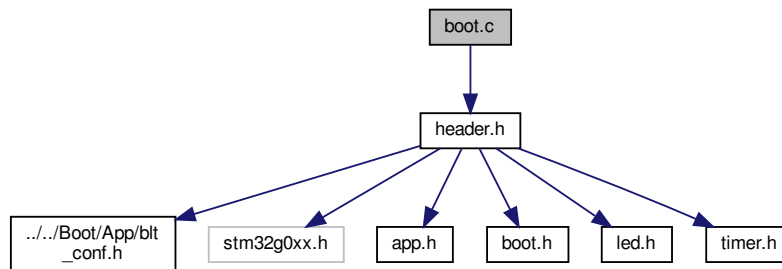
Referenced by `CanGetSpeedConfig()`.

7.166 boot.c File Reference

Demo program bootloader interface source file.

```
#include "header.h"
```

Include dependency graph for Demo/ARMCM0_STM32G0_Nucleo_G071RB_CubeIDE/Prog/App/boot.c:



Macros

- `#define RS232_CTO_RX_PACKET_TIMEOUT_MS (100u)`
Timeout time for the reception of a CTO packet. The timer is started upon reception of the first packet byte.

Functions

- static void `BootComRs232Init` (void)
Initializes the UART communication interface.
- static void `BootComRs232CheckActivationRequest` (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void `BootComInit` (void)
Initializes the communication interface.
- void `BootComCheckActivationRequest` (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void `BootActivate` (void)
Bootloader activation function.
- static unsigned char `Rs232ReceiveByte` (unsigned char *data)
Receives a communication interface byte if one is present.

Variables

- static UART_HandleTypeDef `rs232Handle`
UART handle to be used in API calls.

7.166.1 Detailed Description

Demo program bootloader interface source file.

7.166.2 Function Documentation

7.166.2.1 BootActivate()

```
void BootActivate (
    void )
```

Bootloader activation function.

Returns

none.

Referenced by BootComRs232CheckActivationRequest().

7.166.2.2 BootComCheckActivationRequest()

```
void BootComCheckActivationRequest (
    void )
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.166.2.3 BootComInit()

```
void BootComInit (
    void )
```

Initializes the communication interface.

Returns

none.

7.166.2.4 BootComRs232CheckActivationRequest()

```
static void BootComRs232CheckActivationRequest (  
    void ) [static]
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

Referenced by BootComCheckActivationRequest().

7.166.2.5 BootComRs232Init()

```
static void BootComRs232Init (  
    void ) [static]
```

Initializes the UART communication interface.

Returns

none.

Referenced by BootComInit().

7.166.2.6 Rs232ReceiveByte()

```
static unsigned char Rs232ReceiveByte (  
    unsigned char * data ) [static]
```

Receives a communication interface byte if one is present.

Parameters

<i>data</i>	Pointer to byte where the data is to be stored.
-------------	---

Returns

1 if a byte was received, 0 otherwise.

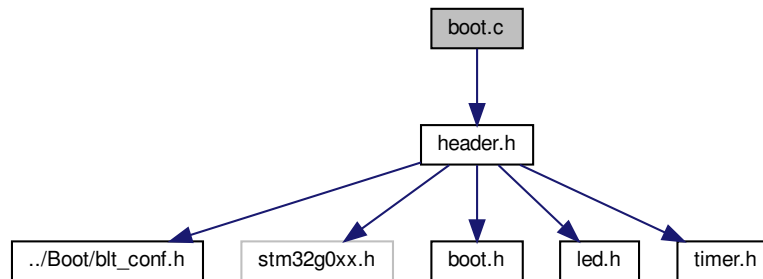
Referenced by BootComRs232CheckActivationRequest().

7.167 boot.c File Reference

Demo program bootloader interface source file.

```
#include "header.h"
```

Include dependency graph for Demo/ARMCM0_STM32G0_Nucleo_G071RB_GCC/Prog/boot.c:



Macros

- `#define RS232_CTO_RX_PACKET_TIMEOUT_MS (100u)`
Timeout time for the reception of a CTO packet. The timer is started upon reception of the first packet byte.

Functions

- static void `BootComRs232Init` (void)
Initializes the UART communication interface.
- static void `BootComRs232CheckActivationRequest` (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void `BootComInit` (void)
Initializes the communication interface.
- void `BootComCheckActivationRequest` (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void `BootActivate` (void)
Bootloader activation function.
- static unsigned char `Rs232ReceiveByte` (unsigned char *data)
Receives a communication interface byte if one is present.

Variables

- static UART_HandleTypeDef `rs232Handle`
UART handle to be used in API calls.

7.167.1 Detailed Description

Demo program bootloader interface source file.

7.167.2 Function Documentation

7.167.2.1 BootActivate()

```
void BootActivate (
    void )
```

Bootloader activation function.

Returns

none.

Referenced by BootComRs232CheckActivationRequest().

7.167.2.2 BootComCheckActivationRequest()

```
void BootComCheckActivationRequest (
    void )
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.167.2.3 BootComInit()

```
void BootComInit (
    void )
```

Initializes the communication interface.

Returns

none.

7.167.2.4 BootComRs232CheckActivationRequest()

```
static void BootComRs232CheckActivationRequest (
    void ) [static]
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

Referenced by BootComCheckActivationRequest().

7.167.2.5 BootComRs232Init()

```
static void BootComRs232Init (
    void ) [static]
```

Initializes the UART communication interface.

Returns

none.

Referenced by BootComInit().

7.167.2.6 Rs232ReceiveByte()

```
static unsigned char Rs232ReceiveByte (
    unsigned char * data ) [static]
```

Receives a communication interface byte if one is present.

Parameters

<i>data</i>	Pointer to byte where the data is to be stored.
-------------	---

Returns

1 if a byte was received, 0 otherwise.

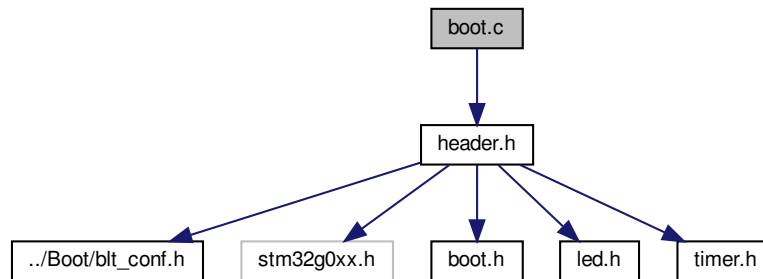
Referenced by BootComRs232CheckActivationRequest().

7.168 boot.c File Reference

Demo program bootloader interface source file.

```
#include "header.h"
```

Include dependency graph for Demo/ARMCM0_STM32G0_Nucleo_G071RB_IAR/Prog/boot.c:



Macros

- `#define RS232_CTO_RX_PACKET_TIMEOUT_MS (100u)`
Timeout time for the reception of a CTO packet. The timer is started upon reception of the first packet byte.

Functions

- static void `BootComRs232Init` (void)
Initializes the UART communication interface.
- static void `BootComRs232CheckActivationRequest` (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void `BootComInit` (void)
Initializes the communication interface.
- void `BootComCheckActivationRequest` (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void `BootActivate` (void)
Bootloader activation function.
- static unsigned char `Rs232ReceiveByte` (unsigned char *data)
Receives a communication interface byte if one is present.

Variables

- static UART_HandleTypeDef `rs232Handle`
UART handle to be used in API calls.

7.168.1 Detailed Description

Demo program bootloader interface source file.

7.168.2 Function Documentation

7.168.2.1 BootActivate()

```
void BootActivate (
    void )
```

Bootloader activation function.

Returns

none.

Referenced by BootComRs232CheckActivationRequest().

7.168.2.2 BootComCheckActivationRequest()

```
void BootComCheckActivationRequest (
    void )
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.168.2.3 BootComInit()

```
void BootComInit (
    void )
```

Initializes the communication interface.

Returns

none.

7.168.2.4 BootComRs232CheckActivationRequest()

```
static void BootComRs232CheckActivationRequest (
    void ) [static]
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

Referenced by BootComCheckActivationRequest().

7.168.2.5 BootComRs232Init()

```
static void BootComRs232Init (
    void ) [static]
```

Initializes the UART communication interface.

Returns

none.

Referenced by BootComInit().

7.168.2.6 Rs232ReceiveByte()

```
static unsigned char Rs232ReceiveByte (
    unsigned char * data ) [static]
```

Receives a communication interface byte if one is present.

Parameters

<i>data</i>	Pointer to byte where the data is to be stored.
-------------	---

Returns

1 if a byte was received, 0 otherwise.

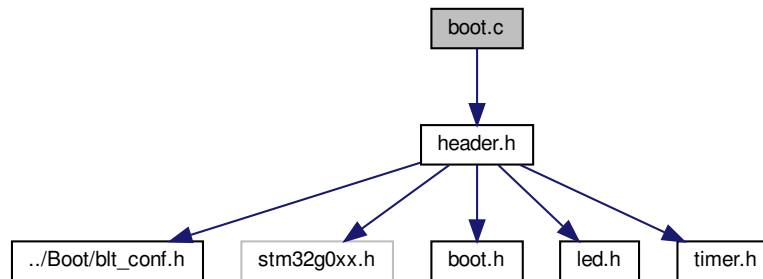
Referenced by BootComRs232CheckActivationRequest().

7.169 boot.c File Reference

Demo program bootloader interface source file.

```
#include "header.h"
```

Include dependency graph for Demo/ARMCM0_STM32G0_Nucleo_G071RB_Keil/Prog/boot.c:



Macros

- `#define RS232_CTO_RX_PACKET_TIMEOUT_MS (100u)`
Timeout time for the reception of a CTO packet. The timer is started upon reception of the first packet byte.

Functions

- static void `BootComRs232Init` (void)
Initializes the UART communication interface.
- static void `BootComRs232CheckActivationRequest` (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void `BootComInit` (void)
Initializes the communication interface.
- void `BootComCheckActivationRequest` (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void `BootActivate` (void)
Bootloader activation function.
- static unsigned char `Rs232ReceiveByte` (unsigned char *data)
Receives a communication interface byte if one is present.

Variables

- static UART_HandleTypeDef `rs232Handle`
UART handle to be used in API calls.

7.169.1 Detailed Description

Demo program bootloader interface source file.

7.169.2 Function Documentation

7.169.2.1 BootActivate()

```
void BootActivate (
    void )
```

Bootloader activation function.

Returns

none.

Referenced by BootComRs232CheckActivationRequest().

7.169.2.2 BootComCheckActivationRequest()

```
void BootComCheckActivationRequest (
    void )
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.169.2.3 BootComInit()

```
void BootComInit (
    void )
```

Initializes the communication interface.

Returns

none.

7.169.2.4 BootComRs232CheckActivationRequest()

```
static void BootComRs232CheckActivationRequest (
    void ) [static]
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

Referenced by BootComCheckActivationRequest().

7.169.2.5 BootComRs232Init()

```
static void BootComRs232Init (
    void ) [static]
```

Initializes the UART communication interface.

Returns

none.

Referenced by BootComInit().

7.169.2.6 Rs232ReceiveByte()

```
static unsigned char Rs232ReceiveByte (
    unsigned char * data ) [static]
```

Receives a communication interface byte if one is present.

Parameters

<i>data</i>	Pointer to byte where the data is to be stored.
-------------	---

Returns

1 if a byte was received, 0 otherwise.

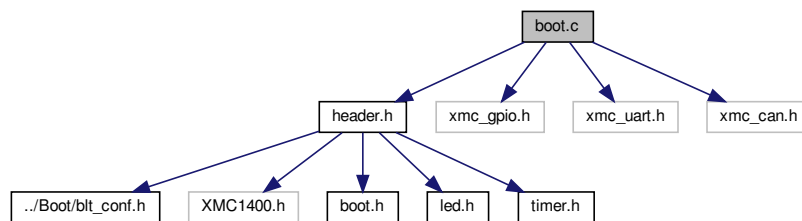
Referenced by BootComRs232CheckActivationRequest().

7.170 boot.c File Reference

Demo program bootloader interface source file.

```
#include "header.h"
#include "xmc_gpio.h"
#include "xmc_uart.h"
#include "xmc_can.h"
```

Include dependency graph for Demo/ARMCM0_XMC1_XMC1400_Boot_Kit_GCC/Prog/boot.c:



Macros

- `#define RS232_CTO_RX_PACKET_TIMEOUT_MS (100u)`
Timeout time for the reception of a CTO packet. The timer is started upon reception of the first packet byte.

Functions

- static void `BootComRs232Init` (void)
Initializes the UART communication interface.
- static void `BootComRs232CheckActivationRequest` (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- static void `BootComCanInit` (void)
Initializes the CAN communication interface.
- static void `BootComCanCheckActivationRequest` (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void `BootComInit` (void)
Initializes the communication interface.
- void `BootComCheckActivationRequest` (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void `BootActivate` (void)
Bootloader activation function.
- static unsigned char `Rs232ReceiveByte` (unsigned char *data)
Receives a communication interface byte if one is present.

Variables

- static XMC_CAN_MO_t [receiveMsgObj](#)
Receive message object data structure.

7.170.1 Detailed Description

Demo program bootloader interface source file.

7.170.2 Function Documentation

7.170.2.1 BootActivate()

```
void BootActivate (  
    void )
```

Bootloader activation function.

Returns

none.

Referenced by [BootComCanCheckActivationRequest\(\)](#), and [BootComRs232CheckActivationRequest\(\)](#).

7.170.2.2 BootComCanCheckActivationRequest()

```
static void BootComCanCheckActivationRequest (  
    void ) [static]
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

Referenced by [BootComCheckActivationRequest\(\)](#).

7.170.2.3 BootComCanInit()

```
static void BootComCanInit (  
    void ) [static]
```

Initializes the CAN communication interface.

Returns

none.

Referenced by BootComInit().

7.170.2.4 BootComCheckActivationRequest()

```
void BootComCheckActivationRequest (  
    void )
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.170.2.5 BootComInit()

```
void BootComInit (  
    void )
```

Initializes the communication interface.

Returns

none.

7.170.2.6 BootComRs232CheckActivationRequest()

```
static void BootComRs232CheckActivationRequest (  
    void ) [static]
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

Referenced by BootComCheckActivationRequest().

7.170.2.7 BootComRs232Init()

```
static void BootComRs232Init (
    void ) [static]
```

Initializes the UART communication interface.

Returns

none.

Referenced by BootComInit().

7.170.2.8 Rs232ReceiveByte()

```
static unsigned char Rs232ReceiveByte (
    unsigned char * data ) [static]
```

Receives a communication interface byte if one is present.

Parameters

<i>data</i>	Pointer to byte where the data is to be stored.
-------------	---

Returns

1 if a byte was received, 0 otherwise.

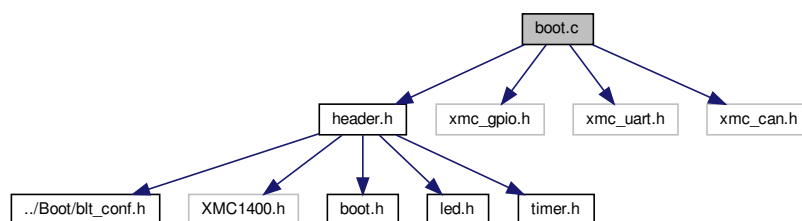
Referenced by BootComRs232CheckActivationRequest().

7.171 boot.c File Reference

Demo program bootloader interface source file.

```
#include "header.h"
#include "xmc_gpio.h"
#include "xmc_uart.h"
#include "xmc_can.h"
```

Include dependency graph for Demo/ARMCM0_XMC1_XMC1400_Boot_Kit_IAR/Prog/boot.c:



Macros

- `#define RS232_CTO_RX_PACKET_TIMEOUT_MS (100u)`
Timeout time for the reception of a CTO packet. The timer is started upon reception of the first packet byte.

Functions

- static void `BootComRs232Init` (void)
Initializes the UART communication interface.
- static void `BootComRs232CheckActivationRequest` (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- static void `BootComCanInit` (void)
Initializes the CAN communication interface.
- static void `BootComCanCheckActivationRequest` (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void `BootComInit` (void)
Initializes the communication interface.
- void `BootComCheckActivationRequest` (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void `BootActivate` (void)
Bootloader activation function.
- static unsigned char `Rs232ReceiveByte` (unsigned char *data)
Receives a communication interface byte if one is present.

Variables

- static XMC_CAN_MO_t `receiveMsgObj`
Receive message object data structure.

7.171.1 Detailed Description

Demo program bootloader interface source file.

7.171.2 Function Documentation

7.171.2.1 BootActivate()

```
void BootActivate (
    void )
```

Bootloader activation function.

Returns

none.

Referenced by `BootComCanCheckActivationRequest()`, and `BootComRs232CheckActivationRequest()`.

7.171.2.2 BootComCanCheckActivationRequest()

```
static void BootComCanCheckActivationRequest (  
    void ) [static]
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

Referenced by BootComCheckActivationRequest().

7.171.2.3 BootComCanInit()

```
static void BootComCanInit (  
    void ) [static]
```

Initializes the CAN communication interface.

Returns

none.

Referenced by BootComInit().

7.171.2.4 BootComCheckActivationRequest()

```
void BootComCheckActivationRequest (  
    void )
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.171.2.5 BootComInit()

```
void BootComInit (
    void )
```

Initializes the communication interface.

Returns

none.

7.171.2.6 BootComRs232CheckActivationRequest()

```
static void BootComRs232CheckActivationRequest (
    void ) [static]
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

Referenced by BootComCheckActivationRequest().

7.171.2.7 BootComRs232Init()

```
static void BootComRs232Init (
    void ) [static]
```

Initializes the UART communication interface.

Returns

none.

Referenced by BootComInit().

7.171.2.8 Rs232ReceiveByte()

```
static unsigned char Rs232ReceiveByte (
    unsigned char * data ) [static]
```

Receives a communication interface byte if one is present.

Parameters

<i>data</i>	Pointer to byte where the data is to be stored.
-------------	---

Returns

1 if a byte was received, 0 otherwise.

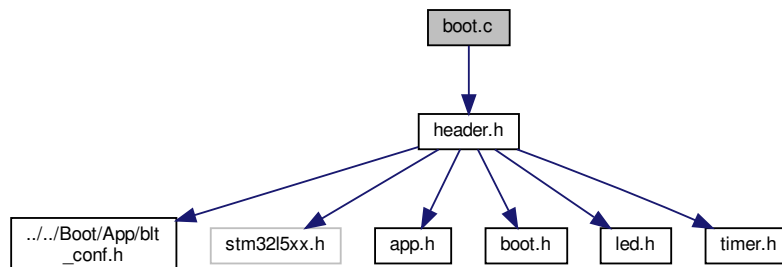
Referenced by `BootComRs232CheckActivationRequest()`.

7.172 boot.c File Reference

Demo program bootloader interface source file.

```
#include "header.h"
```

Include dependency graph for Demo/ARMCM33_STM32L5_Nucleo_L552ZE_CubeIDE/Prog/App/boot.c:

**Data Structures**

- struct [tCanBusTiming](#)

Structure type for grouping CAN bus timing related information.

Macros

- #define [RS232_CTO_RX_PACKET_TIMEOUT_MS](#) (100u)

Timeout time for the reception of a CTO packet. The timer is started upon reception of the first packet byte.

Functions

- static void [BootComRs232Init](#) (void)
Initializes the UART communication interface.
- static void [BootComRs232CheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- static void [BootComCanInit](#) (void)
Initializes the CAN communication interface.
- static void [BootComCanCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootComInit](#) (void)
Initializes the communication interface.
- void [BootComCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootActivate](#) (void)
Bootloader activation function.
- static unsigned char [Rs232ReceiveByte](#) (unsigned char *data)
Receives a communication interface byte if one is present.
- static unsigned char [CanGetSpeedConfig](#) (unsigned short baud, unsigned short *prescaler, unsigned char *tseg1, unsigned char *tseg2)
Search algorithm to match the desired baudrate to a possible bus timing configuration.

Variables

- static UART_HandleTypeDef [rs232Handle](#)
UART handle to be used in API calls.
- static const [tCanBusTiming](#) [canTiming](#) []
CAN bittiming table for dynamically calculating the bittiming settings.
- static FDCAN_HandleTypeDef [canHandle](#)
CAN handle to be used in API calls.

7.172.1 Detailed Description

Demo program bootloader interface source file.

7.172.2 Function Documentation

7.172.2.1 [BootActivate\(\)](#)

```
void BootActivate (
    void )
```

Bootloader activation function.

Returns

none.

Referenced by [BootComCanCheckActivationRequest\(\)](#), and [BootComRs232CheckActivationRequest\(\)](#).

7.172.2.2 BootComCanCheckActivationRequest()

```
static void BootComCanCheckActivationRequest (  
    void ) [static]
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

Referenced by BootComCheckActivationRequest().

7.172.2.3 BootComCanInit()

```
static void BootComCanInit (  
    void ) [static]
```

Initializes the CAN communication interface.

Returns

none.

Referenced by BootComInit().

7.172.2.4 BootComCheckActivationRequest()

```
void BootComCheckActivationRequest (  
    void )
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.172.2.5 BootComInit()

```
void BootComInit (
    void )
```

Initializes the communication interface.

Returns

none.

7.172.2.6 BootComRs232CheckActivationRequest()

```
static void BootComRs232CheckActivationRequest (
    void ) [static]
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

Referenced by BootComCheckActivationRequest().

7.172.2.7 BootComRs232Init()

```
static void BootComRs232Init (
    void ) [static]
```

Initializes the UART communication interface.

Returns

none.

Referenced by BootComInit().

7.172.2.8 CanGetSpeedConfig()

```
static unsigned char CanGetSpeedConfig (
    unsigned short baud,
    unsigned short * prescaler,
    unsigned char * tseg1,
    unsigned char * tseg2 ) [static]
```

Search algorithm to match the desired baudrate to a possible bus timing configuration.

Parameters

<i>baud</i>	The desired baudrate in kbps. Valid values are 10..1000.
<i>prescaler</i>	Pointer to where the value for the prescaler will be stored.
<i>tseg1</i>	Pointer to where the value for TSEG2 will be stored.
<i>tseg2</i>	Pointer to where the value for TSEG2 will be stored.

Returns

1 if the CAN bustiming register values were found, 0 otherwise.

Referenced by BootComCanInit().

7.172.2.9 Rs232ReceiveByte()

```
static unsigned char Rs232ReceiveByte (
    unsigned char * data ) [static]
```

Receives a communication interface byte if one is present.

Parameters

<i>data</i>	Pointer to byte where the data is to be stored.
-------------	---

Returns

1 if a byte was received, 0 otherwise.

Referenced by BootComRs232CheckActivationRequest().

7.172.3 Variable Documentation**7.172.3.1 canTiming**

```
const tCanBusTiming canTiming[] [static]
```

Initial value:

```
=
{
    { 5, 2 },
    { 6, 2 },
    { 6, 3 },
    { 7, 3 },
    { 8, 3 },
```

```

    { 9, 3 },
    { 9, 4 },
    { 10, 4 },
    { 11, 4 },
    { 12, 4 },
    { 12, 5 },
    { 13, 5 },
    { 14, 5 },
    { 15, 5 },
    { 15, 6 },
    { 16, 6 },
    { 16, 7 },
    { 16, 8 }
}

```

CAN bittiming table for dynamically calculating the bittiming settings.

According to the CAN protocol 1 bit-time can be made up of between 8..25 time quanta (TQ). The total TQ in a bit is SYNC + TSEG1 + TSEG2 with SYNC always being 1. The sample point is (SYNC + TSEG1) / (SYNC + TSEG1 + SEG2) * 100%. This array contains possible and valid time quanta configurations with a sample point between 68..78%.

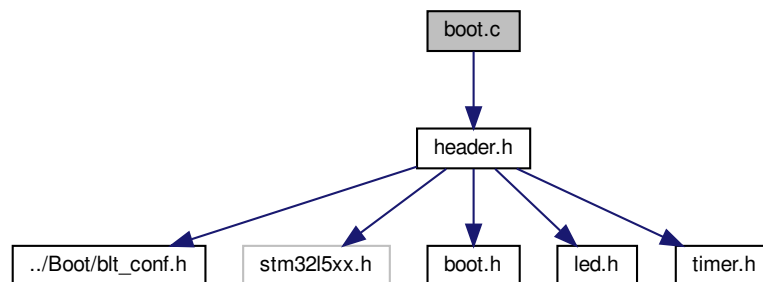
Referenced by CanGetSpeedConfig().

7.173 boot.c File Reference

Demo program bootloader interface source file.

```
#include "header.h"
```

Include dependency graph for Demo/ARMCM33_STM32L5_Nucleo_L552ZE_GCC/Prog/boot.c:



Data Structures

- struct [tCanBusTiming](#)
Structure type for grouping CAN bus timing related information.

Macros

- #define [RS232_CTO_RX_PACKET_TIMEOUT_MS](#) (100u)
Timeout time for the reception of a CTO packet. The timer is started upon reception of the first packet byte.

Functions

- static void [BootComRs232Init](#) (void)
Initializes the UART communication interface.
- static void [BootComRs232CheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- static void [BootComCanInit](#) (void)
Initializes the CAN communication interface.
- static void [BootComCanCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootComInit](#) (void)
Initializes the communication interface.
- void [BootComCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootActivate](#) (void)
Bootloader activation function.
- static unsigned char [Rs232ReceiveByte](#) (unsigned char *data)
Receives a communication interface byte if one is present.
- static unsigned char [CanGetSpeedConfig](#) (unsigned short baud, unsigned short *prescaler, unsigned char *tseg1, unsigned char *tseg2)
Search algorithm to match the desired baudrate to a possible bus timing configuration.

Variables

- static UART_HandleTypeDef [rs232Handle](#)
UART handle to be used in API calls.
- static const [tCanBusTiming](#) [canTiming](#) []
CAN bittiming table for dynamically calculating the bittiming settings.
- static FDCAN_HandleTypeDef [canHandle](#)
CAN handle to be used in API calls.

7.173.1 Detailed Description

Demo program bootloader interface source file.

7.173.2 Function Documentation

7.173.2.1 BootActivate()

```
void BootActivate (
    void )
```

Bootloader activation function.

Returns

none.

Referenced by [BootComCanCheckActivationRequest\(\)](#), and [BootComRs232CheckActivationRequest\(\)](#).

7.173.2.2 BootComCanCheckActivationRequest()

```
static void BootComCanCheckActivationRequest (  
    void ) [static]
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

Referenced by BootComCheckActivationRequest().

7.173.2.3 BootComCanInit()

```
static void BootComCanInit (  
    void ) [static]
```

Initializes the CAN communication interface.

Returns

none.

Referenced by BootComInit().

7.173.2.4 BootComCheckActivationRequest()

```
void BootComCheckActivationRequest (  
    void )
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.173.2.5 BootComInit()

```
void BootComInit (
    void )
```

Initializes the communication interface.

Returns

none.

7.173.2.6 BootComRs232CheckActivationRequest()

```
static void BootComRs232CheckActivationRequest (
    void ) [static]
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

Referenced by BootComCheckActivationRequest().

7.173.2.7 BootComRs232Init()

```
static void BootComRs232Init (
    void ) [static]
```

Initializes the UART communication interface.

Returns

none.

Referenced by BootComInit().

7.173.2.8 CanGetSpeedConfig()

```
static unsigned char CanGetSpeedConfig (
    unsigned short baud,
    unsigned short * prescaler,
    unsigned char * tseg1,
    unsigned char * tseg2 ) [static]
```

Search algorithm to match the desired baudrate to a possible bus timing configuration.

Parameters

<i>baud</i>	The desired baudrate in kbps. Valid values are 10..1000.
<i>prescaler</i>	Pointer to where the value for the prescaler will be stored.
<i>tseg1</i>	Pointer to where the value for TSEG2 will be stored.
<i>tseg2</i>	Pointer to where the value for TSEG2 will be stored.

Returns

1 if the CAN bustiming register values were found, 0 otherwise.

Referenced by BootComCanInit().

7.173.2.9 Rs232ReceiveByte()

```
static unsigned char Rs232ReceiveByte (  
    unsigned char * data ) [static]
```

Receives a communication interface byte if one is present.

Parameters

<i>data</i>	Pointer to byte where the data is to be stored.
-------------	---

Returns

1 if a byte was received, 0 otherwise.

Referenced by BootComRs232CheckActivationRequest().

7.173.3 Variable Documentation**7.173.3.1 canTiming**

```
const tCanBusTiming canTiming[] [static]
```

Initial value:

```
=  
{  
    { 5, 2 },  
    { 6, 2 },  
    { 6, 3 },  
    { 7, 3 },  
    { 8, 3 },  
}
```

```

{ 9, 3 },
{ 9, 4 },
{ 10, 4 },
{ 11, 4 },
{ 12, 4 },
{ 12, 5 },
{ 13, 5 },
{ 14, 5 },
{ 15, 5 },
{ 15, 6 },
{ 16, 6 },
{ 16, 7 },
{ 16, 8 }
}

```

CAN bittiming table for dynamically calculating the bittiming settings.

According to the CAN protocol 1 bit-time can be made up of between 8..25 time quanta (TQ). The total TQ in a bit is SYNC + TSEG1 + TSEG2 with SYNC always being 1. The sample point is (SYNC + TSEG1) / (SYNC + TSEG1 + SEG2) * 100%. This array contains possible and valid time quanta configurations with a sample point between 68..78%.

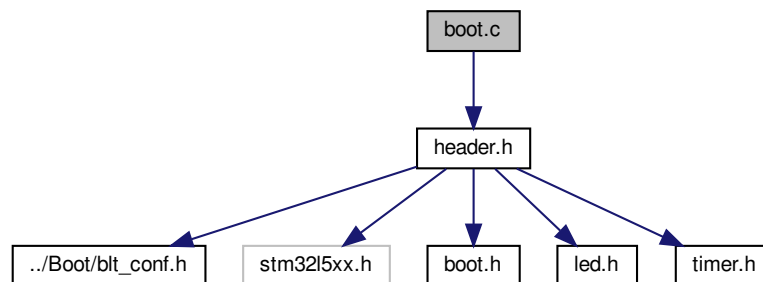
Referenced by CanGetSpeedConfig().

7.174 boot.c File Reference

Demo program bootloader interface source file.

```
#include "header.h"
```

Include dependency graph for Demo/ARMCM33_STM32L5_Nucleo_L552ZE_IAR/Prog/boot.c:



Data Structures

- struct [tCanBusTiming](#)
Structure type for grouping CAN bus timing related information.

Macros

- #define [RS232_CTO_RX_PACKET_TIMEOUT_MS](#) (100u)
Timeout time for the reception of a CTO packet. The timer is started upon reception of the first packet byte.

Functions

- static void [BootComRs232Init](#) (void)
Initializes the UART communication interface.
- static void [BootComRs232CheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- static void [BootComCanInit](#) (void)
Initializes the CAN communication interface.
- static void [BootComCanCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootComInit](#) (void)
Initializes the communication interface.
- void [BootComCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootActivate](#) (void)
Bootloader activation function.
- static unsigned char [Rs232ReceiveByte](#) (unsigned char *data)
Receives a communication interface byte if one is present.
- static unsigned char [CanGetSpeedConfig](#) (unsigned short baud, unsigned short *prescaler, unsigned char *tseg1, unsigned char *tseg2)
Search algorithm to match the desired baudrate to a possible bus timing configuration.

Variables

- static UART_HandleTypeDef [rs232Handle](#)
UART handle to be used in API calls.
- static const [tCanBusTiming](#) [canTiming](#) []
CAN bittiming table for dynamically calculating the bittiming settings.
- static FDCAN_HandleTypeDef [canHandle](#)
CAN handle to be used in API calls.

7.174.1 Detailed Description

Demo program bootloader interface source file.

7.174.2 Function Documentation

7.174.2.1 [BootActivate\(\)](#)

```
void BootActivate (
    void )
```

Bootloader activation function.

Returns

none.

Referenced by [BootComCanCheckActivationRequest\(\)](#), and [BootComRs232CheckActivationRequest\(\)](#).

7.174.2.2 BootComCanCheckActivationRequest()

```
static void BootComCanCheckActivationRequest (  
    void ) [static]
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

Referenced by BootComCheckActivationRequest().

7.174.2.3 BootComCanInit()

```
static void BootComCanInit (  
    void ) [static]
```

Initializes the CAN communication interface.

Returns

none.

Referenced by BootComInit().

7.174.2.4 BootComCheckActivationRequest()

```
void BootComCheckActivationRequest (  
    void )
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.174.2.5 BootComInit()

```
void BootComInit (
    void )
```

Initializes the communication interface.

Returns

none.

7.174.2.6 BootComRs232CheckActivationRequest()

```
static void BootComRs232CheckActivationRequest (
    void ) [static]
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

Referenced by BootComCheckActivationRequest().

7.174.2.7 BootComRs232Init()

```
static void BootComRs232Init (
    void ) [static]
```

Initializes the UART communication interface.

Returns

none.

Referenced by BootComInit().

7.174.2.8 CanGetSpeedConfig()

```
static unsigned char CanGetSpeedConfig (
    unsigned short baud,
    unsigned short * prescaler,
    unsigned char * tseg1,
    unsigned char * tseg2 ) [static]
```

Search algorithm to match the desired baudrate to a possible bus timing configuration.

Parameters

<i>baud</i>	The desired baudrate in kbps. Valid values are 10..1000.
<i>prescaler</i>	Pointer to where the value for the prescaler will be stored.
<i>tseg1</i>	Pointer to where the value for TSEG2 will be stored.
<i>tseg2</i>	Pointer to where the value for TSEG2 will be stored.

Returns

1 if the CAN bustiming register values were found, 0 otherwise.

Referenced by BootComCanInit().

7.174.2.9 Rs232ReceiveByte()

```
static unsigned char Rs232ReceiveByte (
    unsigned char * data ) [static]
```

Receives a communication interface byte if one is present.

Parameters

<i>data</i>	Pointer to byte where the data is to be stored.
-------------	---

Returns

1 if a byte was received, 0 otherwise.

Referenced by BootComRs232CheckActivationRequest().

7.174.3 Variable Documentation**7.174.3.1 canTiming**

```
const tCanBusTiming canTiming[] [static]
```

Initial value:

```
=
{
    { 5, 2 },
    { 6, 2 },
    { 6, 3 },
    { 7, 3 },
    { 8, 3 },
```



```

    { 9, 3 },
    { 9, 4 },
    { 10, 4 },
    { 11, 4 },
    { 12, 4 },
    { 12, 5 },
    { 13, 5 },
    { 14, 5 },
    { 15, 5 },
    { 15, 6 },
    { 16, 6 },
    { 16, 7 },
    { 16, 8 }
}

```

CAN bittiming table for dynamically calculating the bittiming settings.

According to the CAN protocol 1 bit-time can be made up of between 8..25 time quanta (TQ). The total TQ in a bit is SYNC + TSEG1 + TSEG2 with SYNC always being 1. The sample point is (SYNC + TSEG1) / (SYNC + TSEG1 + SEG2) * 100%. This array contains possible and valid time quanta configurations with a sample point between 68..78%.

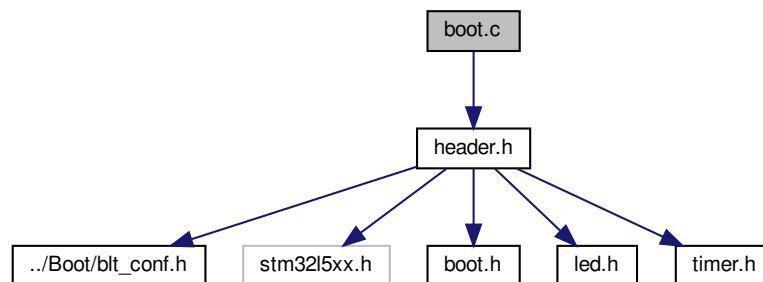
Referenced by CanGetSpeedConfig().

7.175 boot.c File Reference

Demo program bootloader interface source file.

```
#include "header.h"
```

Include dependency graph for Demo/ARMCM33_STM32L5_Nucleo_L552ZE_Keil/Prog/boot.c:



Data Structures

- struct [tCanBusTiming](#)
Structure type for grouping CAN bus timing related information.

Macros

- #define [RS232_CTO_RX_PACKET_TIMEOUT_MS](#) (100u)
Timeout time for the reception of a CTO packet. The timer is started upon reception of the first packet byte.

Functions

- static void [BootComRs232Init](#) (void)
Initializes the UART communication interface.
- static void [BootComRs232CheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- static void [BootComCanInit](#) (void)
Initializes the CAN communication interface.
- static void [BootComCanCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootComInit](#) (void)
Initializes the communication interface.
- void [BootComCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootActivate](#) (void)
Bootloader activation function.
- static unsigned char [Rs232ReceiveByte](#) (unsigned char *data)
Receives a communication interface byte if one is present.
- static unsigned char [CanGetSpeedConfig](#) (unsigned short baud, unsigned short *prescaler, unsigned char *tseg1, unsigned char *tseg2)
Search algorithm to match the desired baudrate to a possible bus timing configuration.

Variables

- static UART_HandleTypeDef [rs232Handle](#)
UART handle to be used in API calls.
- static const [tCanBusTiming](#) [canTiming](#) []
CAN bittiming table for dynamically calculating the bittiming settings.
- static FDCAN_HandleTypeDef [canHandle](#)
CAN handle to be used in API calls.

7.175.1 Detailed Description

Demo program bootloader interface source file.

7.175.2 Function Documentation

7.175.2.1 [BootActivate\(\)](#)

```
void BootActivate (
    void )
```

Bootloader activation function.

Returns

none.

Referenced by [BootComCanCheckActivationRequest\(\)](#), and [BootComRs232CheckActivationRequest\(\)](#).

7.175.2.2 BootComCanCheckActivationRequest()

```
static void BootComCanCheckActivationRequest (  
    void ) [static]
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

Referenced by BootComCheckActivationRequest().

7.175.2.3 BootComCanInit()

```
static void BootComCanInit (  
    void ) [static]
```

Initializes the CAN communication interface.

Returns

none.

Referenced by BootComInit().

7.175.2.4 BootComCheckActivationRequest()

```
void BootComCheckActivationRequest (  
    void )
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.175.2.5 BootComInit()

```
void BootComInit (
    void )
```

Initializes the communication interface.

Returns

none.

7.175.2.6 BootComRs232CheckActivationRequest()

```
static void BootComRs232CheckActivationRequest (
    void ) [static]
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

Referenced by BootComCheckActivationRequest().

7.175.2.7 BootComRs232Init()

```
static void BootComRs232Init (
    void ) [static]
```

Initializes the UART communication interface.

Returns

none.

Referenced by BootComInit().

7.175.2.8 CanGetSpeedConfig()

```
static unsigned char CanGetSpeedConfig (
    unsigned short baud,
    unsigned short * prescaler,
    unsigned char * tseg1,
    unsigned char * tseg2 ) [static]
```

Search algorithm to match the desired baudrate to a possible bus timing configuration.

Parameters

<i>baud</i>	The desired baudrate in kbps. Valid values are 10..1000.
<i>prescaler</i>	Pointer to where the value for the prescaler will be stored.
<i>tseg1</i>	Pointer to where the value for TSEG2 will be stored.
<i>tseg2</i>	Pointer to where the value for TSEG2 will be stored.

Returns

1 if the CAN bustiming register values were found, 0 otherwise.

Referenced by BootComCanInit().

7.175.2.9 Rs232ReceiveByte()

```
static unsigned char Rs232ReceiveByte (  
    unsigned char * data ) [static]
```

Receives a communication interface byte if one is present.

Parameters

<i>data</i>	Pointer to byte where the data is to be stored.
-------------	---

Returns

1 if a byte was received, 0 otherwise.

Referenced by BootComRs232CheckActivationRequest().

7.175.3 Variable Documentation**7.175.3.1 canTiming**

```
const tCanBusTiming canTiming[] [static]
```

Initial value:

```
=  
{  
    { 5, 2 },  
    { 6, 2 },  
    { 6, 3 },  
    { 7, 3 },  
    { 8, 3 },  
}
```

```

{ 9, 3 },
{ 9, 4 },
{ 10, 4 },
{ 11, 4 },
{ 12, 4 },
{ 12, 5 },
{ 13, 5 },
{ 14, 5 },
{ 15, 5 },
{ 15, 6 },
{ 16, 6 },
{ 16, 7 },
{ 16, 8 }
}

```

CAN bittiming table for dynamically calculating the bittiming settings.

According to the CAN protocol 1 bit-time can be made up of between 8..25 time quanta (TQ). The total TQ in a bit is SYNC + TSEG1 + TSEG2 with SYNC always being 1. The sample point is (SYNC + TSEG1) / (SYNC + TSEG1 + SEG2) * 100%. This array contains possible and valid time quanta configurations with a sample point between 68..78%.

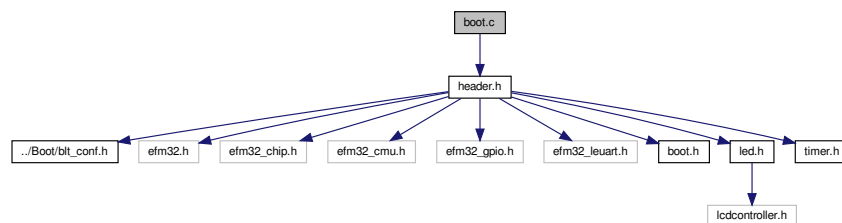
Referenced by CanGetSpeedConfig().

7.176 boot.c File Reference

Demo program bootloader interface source file.

```
#include "header.h"
```

Include dependency graph for Demo/ARMCM3_EFM32_Olimex_EM32G880F128STK_GCC/Prog/boot.c:



Macros

- `#define RS232_CTO_RX_PACKET_TIMEOUT_MS (100u)`
Timeout time for the reception of a CTO packet. The timer is started upon reception of the first packet byte.

Functions

- static void `BootComRs232Init` (void)
Initializes the UART communication interface.
- static void `BootComRs232CheckActivationRequest` (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void `BootComInit` (void)

Initializes the communication interface.

- void [BootComCheckActivationRequest](#) (void)

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

- void [BootActivate](#) (void)

Bootloader activation function.

- static unsigned char [Rs232ReceiveByte](#) (unsigned char *data)

Receives a communication interface byte if one is present.

7.176.1 Detailed Description

Demo program bootloader interface source file.

7.176.2 Function Documentation

7.176.2.1 BootActivate()

```
void BootActivate (  
    void )
```

Bootloader activation function.

Returns

none.

Referenced by [BootComRs232CheckActivationRequest\(\)](#).

7.176.2.2 BootComCheckActivationRequest()

```
void BootComCheckActivationRequest (  
    void )
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.176.2.3 BootComInit()

```
void BootComInit (
    void )
```

Initializes the communication interface.

Returns

none.

7.176.2.4 BootComRs232CheckActivationRequest()

```
static void BootComRs232CheckActivationRequest (
    void ) [static]
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

Referenced by BootComCheckActivationRequest().

7.176.2.5 BootComRs232Init()

```
static void BootComRs232Init (
    void ) [static]
```

Initializes the UART communication interface.

Returns

none.

Referenced by BootComInit().

7.176.2.6 Rs232ReceiveByte()

```
static unsigned char Rs232ReceiveByte (
    unsigned char * data ) [static]
```

Receives a communication interface byte if one is present.

Parameters

<i>data</i>	Pointer to byte where the data is to be stored.
-------------	---

Returns

1 if a byte was received, 0 otherwise.

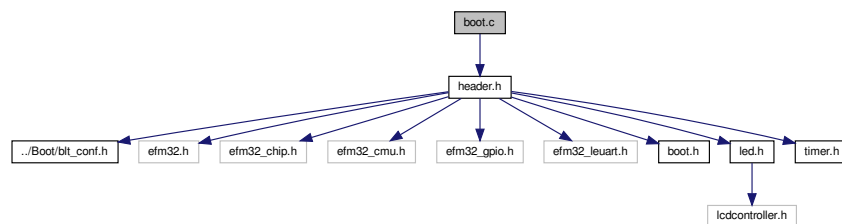
Referenced by `BootComRs232CheckActivationRequest()`.

7.177 boot.c File Reference

Demo program bootloader interface source file.

```
#include "header.h"
```

Include dependency graph for Demo/ARMCM3_EFM32_Olimex_EM32G880F128STK_IAR/Prog/boot.c:



Macros

- `#define RS232_CTO_RX_PACKET_TIMEOUT_MS (100u)`
Timeout time for the reception of a CTO packet. The timer is started upon reception of the first packet byte.

Functions

- static void `BootComRs232Init` (void)
Initializes the UART communication interface.
- static void `BootComRs232CheckActivationRequest` (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void `BootComInit` (void)
Initializes the communication interface.
- void `BootComCheckActivationRequest` (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void `BootActivate` (void)
Bootloader activation function.
- static unsigned char `Rs232ReceiveByte` (unsigned char *data)
Receives a communication interface byte if one is present.

7.177.1 Detailed Description

Demo program bootloader interface source file.

7.177.2 Function Documentation

7.177.2.1 BootActivate()

```
void BootActivate (
    void )
```

Bootloader activation function.

Returns

none.

Referenced by BootComRs232CheckActivationRequest().

7.177.2.2 BootComCheckActivationRequest()

```
void BootComCheckActivationRequest (
    void )
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.177.2.3 BootComInit()

```
void BootComInit (
    void )
```

Initializes the communication interface.

Returns

none.

7.177.2.4 BootComRs232CheckActivationRequest()

```
static void BootComRs232CheckActivationRequest (  
    void ) [static]
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

Referenced by BootComCheckActivationRequest().

7.177.2.5 BootComRs232Init()

```
static void BootComRs232Init (  
    void ) [static]
```

Initializes the UART communication interface.

Returns

none.

Referenced by BootComInit().

7.177.2.6 Rs232ReceiveByte()

```
static unsigned char Rs232ReceiveByte (  
    unsigned char * data ) [static]
```

Receives a communication interface byte if one is present.

Parameters

<i>data</i>	Pointer to byte where the data is to be stored.
-------------	---

Returns

1 if a byte was received, 0 otherwise.

Referenced by BootComRs232CheckActivationRequest().

7.178.2.1 BootActivate()

```
void BootActivate (
    void )
```

Bootloader activation function.

Returns

none.

Referenced by BootComRs232CheckActivationRequest().

7.178.2.2 BootComCheckActivationRequest()

```
void BootComCheckActivationRequest (
    void )
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.178.2.3 BootComInit()

```
void BootComInit (
    void )
```

Initializes the communication interface.

Returns

none.

7.178.2.4 BootComRs232CheckActivationRequest()

```
static void BootComRs232CheckActivationRequest (
    void ) [static]
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

Referenced by BootComCheckActivationRequest().

7.178.2.5 BootComRs232Init()

```
static void BootComRs232Init (
    void ) [static]
```

Initializes the UART communication interface.

Returns

none.

Referenced by BootComInit().

7.178.2.6 Rs232ReceiveByte()

```
static unsigned char Rs232ReceiveByte (
    unsigned char * data ) [static]
```

Receives a communication interface byte if one is present.

Parameters

<i>data</i>	Pointer to byte where the data is to be stored.
-------------	---

Returns

1 if a byte was received, 0 otherwise.

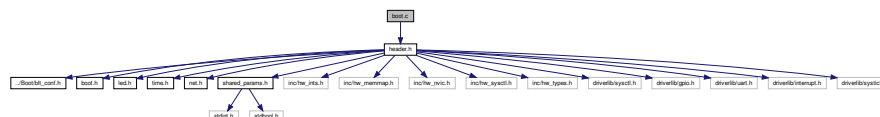
Referenced by BootComRs232CheckActivationRequest().

7.179 boot.c File Reference

Demo program bootloader interface source file.

```
#include "header.h"
```

Include dependency graph for Demo/ARMCM3_LM3S_EK_LM3S6965_IAR/Prog/boot.c:



Macros

- `#define RS232_CTO_RX_PACKET_TIMEOUT_MS (100u)`
Timeout time for the reception of a CTO packet. The timer is started upon reception of the first packet byte.

Functions

- static void [BootComRs232Init](#) (void)
Initializes the UART communication interface.
- static void [BootComRs232CheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootComInit](#) (void)
Initializes the communication interface.
- void [BootComCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootActivate](#) (void)
Bootloader activation function.
- static unsigned char [Rs232ReceiveByte](#) (unsigned char *data)
Receives a communication interface byte if one is present.

7.179.1 Detailed Description

Demo program bootloader interface source file.

7.179.2 Function Documentation

7.179.2.1 [BootActivate\(\)](#)

```
void BootActivate (  
    void )
```

Bootloader activation function.

Returns

none.

Referenced by [BootComRs232CheckActivationRequest\(\)](#).

7.179.2.2 [BootComCheckActivationRequest\(\)](#)

```
void BootComCheckActivationRequest (  
    void )
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.179.2.3 BootComInit()

```
void BootComInit (
    void )
```

Initializes the communication interface.

Returns

none.

7.179.2.4 BootComRs232CheckActivationRequest()

```
static void BootComRs232CheckActivationRequest (
    void ) [static]
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

Referenced by BootComCheckActivationRequest().

7.179.2.5 BootComRs232Init()

```
static void BootComRs232Init (
    void ) [static]
```

Initializes the UART communication interface.

Returns

none.

Referenced by BootComInit().

7.179.2.6 Rs232ReceiveByte()

```
static unsigned char Rs232ReceiveByte (
    unsigned char * data ) [static]
```

Receives a communication interface byte if one is present.

Parameters

<i>data</i>	Pointer to byte where the data is to be stored.
-------------	---

Returns

1 if a byte was received, 0 otherwise.

Referenced by `BootComRs232CheckActivationRequest()`.

7.180 boot.c File Reference

Demo program bootloader interface source file.

```
#include "header.h"
```

Include dependency graph for Demo/ARMCM3_LM3S_EK_LM3S8962_GCC/Prog/boot.c:



Macros

- `#define RS232_CTO_RX_PACKET_TIMEOUT_MS (100u)`
Timeout time for the reception of a CTO packet. The timer is started upon reception of the first packet byte.
- `#define CAN_RX_MSGOBJECT_IDX (0)`
Index of the used reception message object.

Functions

- static void `BootComRs232Init` (void)
Initializes the UART communication interface.
- static void `BootComRs232CheckActivationRequest` (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- static void `BootComCanInit` (void)
Initializes the CAN communication interface.
- static void `BootComCanCheckActivationRequest` (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void `BootComInit` (void)
Initializes the communication interface.
- void `BootComCheckActivationRequest` (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void `BootActivate` (void)
Bootloader activation function.
- static unsigned char `Rs232ReceiveByte` (unsigned char *data)
Receives a communication interface byte if one is present.
- static unsigned char `CanSetBittiming` (void)
*Attempts to match the bittiming parameters to the requested baudrate for a sample point between 65 and 75%, through a linear search algorithm. It is based on the equation: baudrate = CAN Clock Freq/((1+PropSeg+Phase1↔Seg+Phase2Seg)*Prescaler)*

Variables

- static const unsigned short `canBitNum2Mask []`
Lookup table to quickly and efficiently convert a bit number to a bit mask.

7.180.1 Detailed Description

Demo program bootloader interface source file.

7.180.2 Function Documentation

7.180.2.1 `BootActivate()`

```
void BootActivate (
    void )
```

Bootloader activation function.

Returns

none.

Referenced by `BootComCanCheckActivationRequest()`, and `BootComRs232CheckActivationRequest()`.

7.180.2.2 `BootComCanCheckActivationRequest()`

```
static void BootComCanCheckActivationRequest (
    void ) [static]
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

Referenced by `BootComCheckActivationRequest()`.

7.180.2.3 BootComCanInit()

```
static void BootComCanInit (
    void ) [static]
```

Initializes the CAN communication interface.

Returns

none.

Referenced by BootComInit().

7.180.2.4 BootComCheckActivationRequest()

```
void BootComCheckActivationRequest (
    void )
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.180.2.5 BootComInit()

```
void BootComInit (
    void )
```

Initializes the communication interface.

Returns

none.

7.180.2.6 BootComRs232CheckActivationRequest()

```
static void BootComRs232CheckActivationRequest (
    void ) [static]
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

Referenced by BootComCheckActivationRequest().

7.180.2.7 BootComRs232Init()

```
static void BootComRs232Init (
    void ) [static]
```

Initializes the UART communication interface.

Returns

none.

Referenced by BootComInit().

7.180.2.8 CanSetBittiming()

```
static unsigned char CanSetBittiming (
    void ) [static]
```

Attempts to match the bittiming parameters to the requested baudrate for a sample point between 65 and 75%, through a linear search algorithm. It is based on the equation: baudrate = CAN Clock Freq/((1+PropSeg+Phase1↔Seg+Phase2Seg)*Prescaler)

Returns

1 if a valid bittiming configuration was found and set. 0 otherwise.

Referenced by BootComCanInit().

7.180.2.9 Rs232ReceiveByte()

```
static unsigned char Rs232ReceiveByte (
    unsigned char * data ) [static]
```

Receives a communication interface byte if one is present.

Parameters

<i>data</i>	Pointer to byte where the data is to be stored.
-------------	---

Returns

1 if a byte was received, 0 otherwise.

Referenced by BootComRs232CheckActivationRequest().

7.181.1 Detailed Description

Demo program bootloader interface source file.

7.181.2 Function Documentation

7.181.2.1 BootActivate()

```
void BootActivate (
    void )
```

Bootloader activation function.

Returns

none.

Referenced by BootComCanCheckActivationRequest(), and BootComRs232CheckActivationRequest().

7.181.2.2 BootComCanCheckActivationRequest()

```
static void BootComCanCheckActivationRequest (
    void ) [static]
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

Referenced by BootComCheckActivationRequest().

7.181.2.3 BootComCanInit()

```
static void BootComCanInit (
    void ) [static]
```

Initializes the CAN communication interface.

Returns

none.

Referenced by BootComInit().

7.181.2.4 BootComCheckActivationRequest()

```
void BootComCheckActivationRequest (
    void )
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.181.2.5 BootComInit()

```
void BootComInit (
    void )
```

Initializes the communication interface.

Returns

none.

7.181.2.6 BootComRs232CheckActivationRequest()

```
static void BootComRs232CheckActivationRequest (
    void ) [static]
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

Referenced by BootComCheckActivationRequest().

7.181.2.7 BootComRs232Init()

```
static void BootComRs232Init (
    void ) [static]
```

Initializes the UART communication interface.

Returns

none.

Referenced by BootComInit().

7.181.2.8 CanSetBittiming()

```
static unsigned char CanSetBittiming (
    void ) [static]
```

Attempts to match the bittiming parameters to the requested baudrate for a sample point between 65 and 75%, through a linear search algorithm. It is based on the equation: baudrate = CAN Clock Freq/((1+PropSeg+Phase1↵Seg+Phase2Seg)*Prescaler)

Returns

1 if a valid bittiming configuration was found and set. 0 otherwise.

Referenced by BootComCanInit().

7.181.2.9 Rs232ReceiveByte()

```
static unsigned char Rs232ReceiveByte (
    unsigned char * data ) [static]
```

Receives a communication interface byte if one is present.

Parameters

<i>data</i>	Pointer to byte where the data is to be stored.
-------------	---

Returns

1 if a byte was received, 0 otherwise.

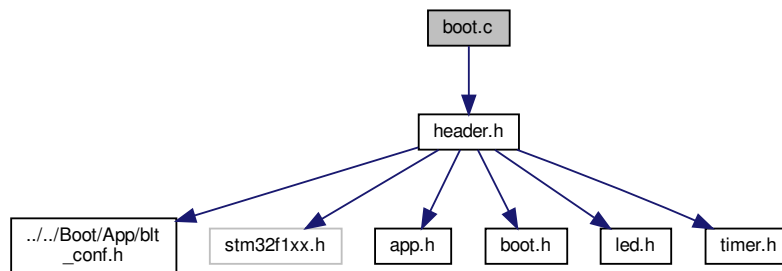
Referenced by BootComRs232CheckActivationRequest().

7.182 boot.c File Reference

Demo program bootloader interface source file.


```
#include "header.h"
```

Include dependency graph for Demo/ARMCM3_STM32F1_Nucleo_F103RB_CubeIDE/Prog/App/boot.c:



Macros

- `#define RS232_CTO_RX_PACKET_TIMEOUT_MS (100u)`
Timeout time for the reception of a CTO packet. The timer is started upon reception of the first packet byte.

Functions

- static void `BootComRs232Init` (void)
Initializes the UART communication interface.
- static void `BootComRs232CheckActivationRequest` (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void `BootComInit` (void)
Initializes the communication interface.
- void `BootComCheckActivationRequest` (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void `BootActivate` (void)
Bootloader activation function.
- static unsigned char `Rs232ReceiveByte` (unsigned char *data)
Receives a communication interface byte if one is present.

Variables

- static UART_HandleTypeDef `rs232Handle`
UART handle to be used in API calls.

7.182.1 Detailed Description

Demo program bootloader interface source file.

7.182.2 Function Documentation

7.182.2.1 BootActivate()

```
void BootActivate (
    void )
```

Bootloader activation function.

Returns

none.

Referenced by BootComRs232CheckActivationRequest().

7.182.2.2 BootComCheckActivationRequest()

```
void BootComCheckActivationRequest (
    void )
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.182.2.3 BootComInit()

```
void BootComInit (
    void )
```

Initializes the communication interface.

Returns

none.

7.182.2.4 BootComRs232CheckActivationRequest()

```
static void BootComRs232CheckActivationRequest (
    void ) [static]
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

Referenced by BootComCheckActivationRequest().

7.182.2.5 BootComRs232Init()

```
static void BootComRs232Init (
    void ) [static]
```

Initializes the UART communication interface.

Returns

none.

Referenced by BootComInit().

7.182.2.6 Rs232ReceiveByte()

```
static unsigned char Rs232ReceiveByte (
    unsigned char * data ) [static]
```

Receives a communication interface byte if one is present.

Parameters

<i>data</i>	Pointer to byte where the data is to be stored.
-------------	---

Returns

1 if a byte was received, 0 otherwise.

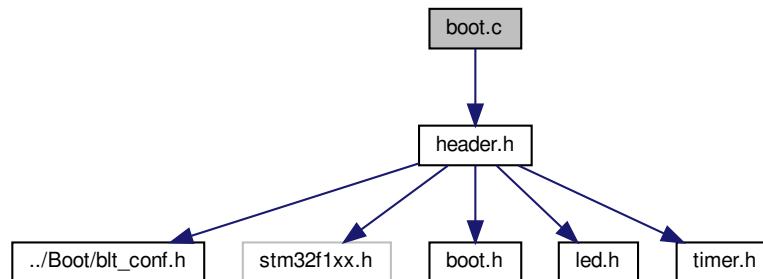
Referenced by BootComRs232CheckActivationRequest().

7.183 boot.c File Reference

Demo program bootloader interface source file.

```
#include "header.h"
```

Include dependency graph for Demo/ARMCM3_STM32F1_Nucleo_F103RB_GCC/Prog/boot.c:



Macros

- `#define RS232_CTO_RX_PACKET_TIMEOUT_MS (100u)`
Timeout time for the reception of a CTO packet. The timer is started upon reception of the first packet byte.

Functions

- static void `BootComRs232Init` (void)
Initializes the UART communication interface.
- static void `BootComRs232CheckActivationRequest` (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void `BootComInit` (void)
Initializes the communication interface.
- void `BootComCheckActivationRequest` (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void `BootActivate` (void)
Bootloader activation function.
- static unsigned char `Rs232ReceiveByte` (unsigned char *data)
Receives a communication interface byte if one is present.

Variables

- static UART_HandleTypeDef `rs232Handle`
UART handle to be used in API calls.

7.183.1 Detailed Description

Demo program bootloader interface source file.

7.183.2 Function Documentation

7.183.2.1 BootActivate()

```
void BootActivate (
    void )
```

Bootloader activation function.

Returns

none.

Referenced by BootComRs232CheckActivationRequest().

7.183.2.2 BootComCheckActivationRequest()

```
void BootComCheckActivationRequest (
    void )
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.183.2.3 BootComInit()

```
void BootComInit (
    void )
```

Initializes the communication interface.

Returns

none.

7.183.2.4 BootComRs232CheckActivationRequest()

```
static void BootComRs232CheckActivationRequest (
    void ) [static]
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

Referenced by BootComCheckActivationRequest().

7.183.2.5 BootComRs232Init()

```
static void BootComRs232Init (
    void ) [static]
```

Initializes the UART communication interface.

Returns

none.

Referenced by BootComInit().

7.183.2.6 Rs232ReceiveByte()

```
static unsigned char Rs232ReceiveByte (
    unsigned char * data ) [static]
```

Receives a communication interface byte if one is present.

Parameters

<i>data</i>	Pointer to byte where the data is to be stored.
-------------	---

Returns

1 if a byte was received, 0 otherwise.

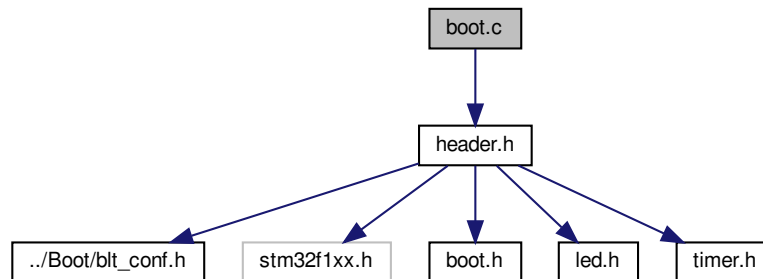
Referenced by BootComRs232CheckActivationRequest().

7.184 boot.c File Reference

Demo program bootloader interface source file.

```
#include "header.h"
```

Include dependency graph for Demo/ARMCM3_STM32F1_Nucleo_F103RB_IAR/Prog/boot.c:



Macros

- `#define RS232_CTO_RX_PACKET_TIMEOUT_MS (100u)`
Timeout time for the reception of a CTO packet. The timer is started upon reception of the first packet byte.

Functions

- static void `BootComRs232Init` (void)
Initializes the UART communication interface.
- static void `BootComRs232CheckActivationRequest` (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void `BootComInit` (void)
Initializes the communication interface.
- void `BootComCheckActivationRequest` (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void `BootActivate` (void)
Bootloader activation function.
- static unsigned char `Rs232ReceiveByte` (unsigned char *data)
Receives a communication interface byte if one is present.

Variables

- static UART_HandleTypeDef `rs232Handle`
UART handle to be used in API calls.

7.184.1 Detailed Description

Demo program bootloader interface source file.

7.184.2 Function Documentation

7.184.2.1 BootActivate()

```
void BootActivate (  
    void )
```

Bootloader activation function.

Returns

none.

Referenced by BootComRs232CheckActivationRequest().

7.184.2.2 BootComCheckActivationRequest()

```
void BootComCheckActivationRequest (  
    void )
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.184.2.3 BootComInit()

```
void BootComInit (  
    void )
```

Initializes the communication interface.

Returns

none.

7.184.2.4 BootComRs232CheckActivationRequest()

```
static void BootComRs232CheckActivationRequest (  
    void ) [static]
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

Referenced by BootComCheckActivationRequest().

7.184.2.5 BootComRs232Init()

```
static void BootComRs232Init (  
    void ) [static]
```

Initializes the UART communication interface.

Returns

none.

Referenced by BootComInit().

7.184.2.6 Rs232ReceiveByte()

```
static unsigned char Rs232ReceiveByte (  
    unsigned char * data ) [static]
```

Receives a communication interface byte if one is present.

Parameters

<i>data</i>	Pointer to byte where the data is to be stored.
-------------	---

Returns

1 if a byte was received, 0 otherwise.

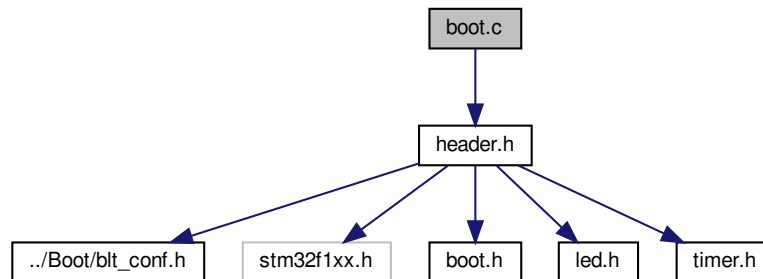
Referenced by BootComRs232CheckActivationRequest().

7.185 boot.c File Reference

Demo program bootloader interface source file.

```
#include "header.h"
```

Include dependency graph for Demo/ARMCM3_STM32F1_Nucleo_F103RB_Keil/Prog/boot.c:



Macros

- `#define RS232_CTO_RX_PACKET_TIMEOUT_MS (100u)`
Timeout time for the reception of a CTO packet. The timer is started upon reception of the first packet byte.

Functions

- static void `BootComRs232Init` (void)
Initializes the UART communication interface.
- static void `BootComRs232CheckActivationRequest` (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void `BootComInit` (void)
Initializes the communication interface.
- void `BootComCheckActivationRequest` (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void `BootActivate` (void)
Bootloader activation function.
- static unsigned char `Rs232ReceiveByte` (unsigned char *data)
Receives a communication interface byte if one is present.

Variables

- static UART_HandleTypeDef `rs232Handle`
UART handle to be used in API calls.

7.185.1 Detailed Description

Demo program bootloader interface source file.

7.185.2 Function Documentation

7.185.2.1 BootActivate()

```
void BootActivate (
    void )
```

Bootloader activation function.

Returns

none.

Referenced by BootComRs232CheckActivationRequest().

7.185.2.2 BootComCheckActivationRequest()

```
void BootComCheckActivationRequest (
    void )
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.185.2.3 BootComInit()

```
void BootComInit (
    void )
```

Initializes the communication interface.

Returns

none.

7.185.2.4 BootComRs232CheckActivationRequest()

```
static void BootComRs232CheckActivationRequest (
    void ) [static]
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

Referenced by BootComCheckActivationRequest().

7.185.2.5 BootComRs232Init()

```
static void BootComRs232Init (
    void ) [static]
```

Initializes the UART communication interface.

Returns

none.

Referenced by BootComInit().

7.185.2.6 Rs232ReceiveByte()

```
static unsigned char Rs232ReceiveByte (
    unsigned char * data ) [static]
```

Receives a communication interface byte if one is present.

Parameters

<i>data</i>	Pointer to byte where the data is to be stored.
-------------	---

Returns

1 if a byte was received, 0 otherwise.

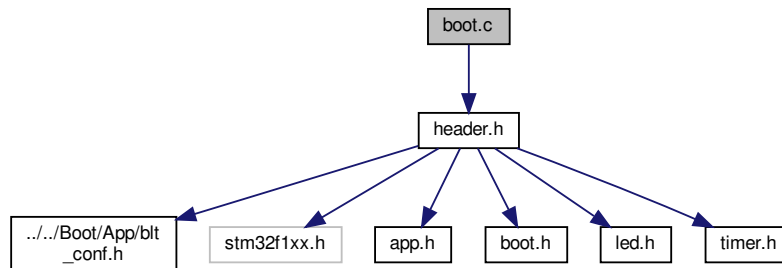
Referenced by BootComRs232CheckActivationRequest().

7.186 boot.c File Reference

Demo program bootloader interface source file.

```
#include "header.h"
```

Include dependency graph for Demo/ARMCM3_STM32F1_Olimex_STM32P103_CubeIDE/Prog/App/boot.c:



Data Structures

- struct [tCanBusTiming](#)
Structure type for grouping CAN bus timing related information.

Macros

- #define [RS232_CTO_RX_PACKET_TIMEOUT_MS](#) (100u)
Timeout time for the reception of a CTO packet. The timer is started upon reception of the first packet byte.

Functions

- static void [BootComRs232Init](#) (void)
Initializes the UART communication interface.
- static void [BootComRs232CheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- static void [BootComCanInit](#) (void)
Initializes the CAN communication interface.
- static void [BootComCanCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootComInit](#) (void)
Initializes the communication interface.
- void [BootComCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootActivate](#) (void)
Bootloader activation function.
- static unsigned char [Rs232ReceiveByte](#) (unsigned char *data)
Receives a communication interface byte if one is present.
- static unsigned char [CanGetSpeedConfig](#) (unsigned short baud, unsigned short *prescaler, unsigned char *tseg1, unsigned char *tseg2)
Search algorithm to match the desired baudrate to a possible bus timing configuration.

Variables

- static UART_HandleTypeDef [rs232Handle](#)
UART handle to be used in API calls.
- static const [tCanBusTiming](#) [canTiming](#) []
CAN bittiming table for dynamically calculating the bittiming settings.
- static CAN_HandleTypeDef [canHandle](#)
CAN handle to be used in API calls.

7.186.1 Detailed Description

Demo program bootloader interface source file.

7.186.2 Function Documentation

7.186.2.1 BootActivate()

```
void BootActivate (  
    void )
```

Bootloader activation function.

Returns

none.

Referenced by [BootComCanCheckActivationRequest\(\)](#), and [BootComRs232CheckActivationRequest\(\)](#).

7.186.2.2 BootComCanCheckActivationRequest()

```
static void BootComCanCheckActivationRequest (  
    void ) [static]
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

Referenced by [BootComCheckActivationRequest\(\)](#).

7.186.2.3 BootComCanInit()

```
static void BootComCanInit (
    void ) [static]
```

Initializes the CAN communication interface.

Returns

none.

Referenced by BootComInit().

7.186.2.4 BootComCheckActivationRequest()

```
void BootComCheckActivationRequest (
    void )
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.186.2.5 BootComInit()

```
void BootComInit (
    void )
```

Initializes the communication interface.

Returns

none.

7.186.2.6 BootComRs232CheckActivationRequest()

```
static void BootComRs232CheckActivationRequest (
    void ) [static]
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

Referenced by BootComCheckActivationRequest().

7.186.2.7 BootComRs232Init()

```
static void BootComRs232Init (  
    void ) [static]
```

Initializes the UART communication interface.

Returns

none.

Referenced by BootComInit().

7.186.2.8 CanGetSpeedConfig()

```
static unsigned char CanGetSpeedConfig (  
    unsigned short baud,  
    unsigned short * prescaler,  
    unsigned char * tseg1,  
    unsigned char * tseg2 ) [static]
```

Search algorithm to match the desired baudrate to a possible bus timing configuration.

Parameters

<i>baud</i>	The desired baudrate in kbps. Valid values are 10..1000.
<i>prescaler</i>	Pointer to where the value for the prescaler will be stored.
<i>tseg1</i>	Pointer to where the value for TSEG2 will be stored.
<i>tseg2</i>	Pointer to where the value for TSEG2 will be stored.

Returns

1 if the CAN bustiming register values were found, 0 otherwise.

Referenced by BootComCanInit().

7.186.2.9 Rs232ReceiveByte()

```
static unsigned char Rs232ReceiveByte (  
    unsigned char * data ) [static]
```

Receives a communication interface byte if one is present.

Parameters

<i>data</i>	Pointer to byte where the data is to be stored.
-------------	---

Returns

1 if a byte was received, 0 otherwise.

Referenced by `BootComRs232CheckActivationRequest()`.

7.186.3 Variable Documentation

7.186.3.1 canTiming

```
const tCanBusTiming canTiming[] [static]
```

Initial value:

```
=  
{  
    { 5, 2 },  
    { 6, 2 },  
    { 6, 3 },  
    { 7, 3 },  
    { 8, 3 },  
    { 9, 3 },  
    { 9, 4 },  
    { 10, 4 },  
    { 11, 4 },  
    { 12, 4 },  
    { 12, 5 },  
    { 13, 5 },  
    { 14, 5 },  
    { 15, 5 },  
    { 15, 6 },  
    { 16, 6 },  
    { 16, 7 },  
    { 16, 8 }  
}
```

CAN bittiming table for dynamically calculating the bittiming settings.

According to the CAN protocol 1 bit-time can be made up of between 8..25 time quanta (TQ). The total TQ in a bit is SYNC + TSEG1 + TSEG2 with SYNC always being 1. The sample point is (SYNC + TSEG1) / (SYNC + TSEG1 + SEG2) * 100%. This array contains possible and valid time quanta configurations with a sample point between 68..78%.

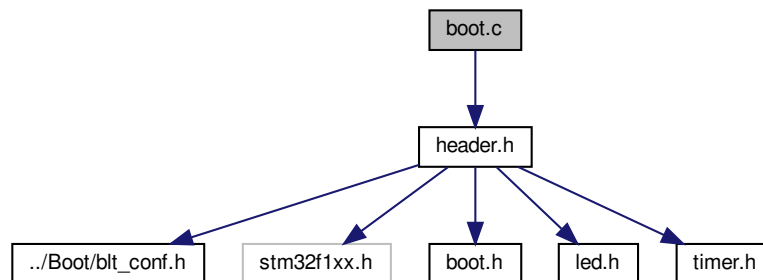
Referenced by `CanGetSpeedConfig()`.

7.187 boot.c File Reference

Demo program bootloader interface source file.

```
#include "header.h"
```

Include dependency graph for Demo/ARMCM3_STM32F1_Olimex_STM32P103_GCC/Prog/boot.c:



Data Structures

- struct [tCanBusTiming](#)
Structure type for grouping CAN bus timing related information.

Macros

- `#define RS232_CTO_RX_PACKET_TIMEOUT_MS (100u)`
Timeout time for the reception of a CTO packet. The timer is started upon reception of the first packet byte.

Functions

- static void [BootComRs232Init](#) (void)
Initializes the UART communication interface.
- static void [BootComRs232CheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- static void [BootComCanInit](#) (void)
Initializes the CAN communication interface.
- static void [BootComCanCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootComInit](#) (void)
Initializes the communication interface.
- void [BootComCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootActivate](#) (void)
Bootloader activation function.
- static unsigned char [Rs232ReceiveByte](#) (unsigned char *data)
Receives a communication interface byte if one is present.
- static unsigned char [CanGetSpeedConfig](#) (unsigned short baud, unsigned short *prescaler, unsigned char *tseg1, unsigned char *tseg2)
Search algorithm to match the desired baudrate to a possible bus timing configuration.

Variables

- static UART_HandleTypeDef [rs232Handle](#)
UART handle to be used in API calls.
- static const [tCanBusTiming](#) [canTiming](#) []
CAN bittiming table for dynamically calculating the bittiming settings.
- static CAN_HandleTypeDef [canHandle](#)
CAN handle to be used in API calls.

7.187.1 Detailed Description

Demo program bootloader interface source file.

7.187.2 Function Documentation

7.187.2.1 BootActivate()

```
void BootActivate (  
    void )
```

Bootloader activation function.

Returns

none.

Referenced by [BootComCanCheckActivationRequest\(\)](#), and [BootComRs232CheckActivationRequest\(\)](#).

7.187.2.2 BootComCanCheckActivationRequest()

```
static void BootComCanCheckActivationRequest (  
    void ) [static]
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

Referenced by [BootComCheckActivationRequest\(\)](#).

7.187.2.3 BootComCanInit()

```
static void BootComCanInit (  
    void ) [static]
```

Initializes the CAN communication interface.

Returns

none.

Referenced by BootComInit().

7.187.2.4 BootComCheckActivationRequest()

```
void BootComCheckActivationRequest (  
    void )
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.187.2.5 BootComInit()

```
void BootComInit (  
    void )
```

Initializes the communication interface.

Returns

none.

7.187.2.6 BootComRs232CheckActivationRequest()

```
static void BootComRs232CheckActivationRequest (  
    void ) [static]
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

Referenced by BootComCheckActivationRequest().

7.187.2.7 BootComRs232Init()

```
static void BootComRs232Init (  
    void ) [static]
```

Initializes the UART communication interface.

Returns

none.

Referenced by BootComInit().

7.187.2.8 CanGetSpeedConfig()

```
static unsigned char CanGetSpeedConfig (  
    unsigned short baud,  
    unsigned short * prescaler,  
    unsigned char * tseg1,  
    unsigned char * tseg2 ) [static]
```

Search algorithm to match the desired baudrate to a possible bus timing configuration.

Parameters

<i>baud</i>	The desired baudrate in kbps. Valid values are 10..1000.
<i>prescaler</i>	Pointer to where the value for the prescaler will be stored.
<i>tseg1</i>	Pointer to where the value for TSEG2 will be stored.
<i>tseg2</i>	Pointer to where the value for TSEG2 will be stored.

Returns

1 if the CAN bustiming register values were found, 0 otherwise.

Referenced by BootComCanInit().

7.187.2.9 Rs232ReceiveByte()

```
static unsigned char Rs232ReceiveByte (  
    unsigned char * data ) [static]
```

Receives a communication interface byte if one is present.

Parameters

<i>data</i>	Pointer to byte where the data is to be stored.
-------------	---

Returns

1 if a byte was received, 0 otherwise.

Referenced by `BootComRs232CheckActivationRequest()`.

7.187.3 Variable Documentation**7.187.3.1 canTiming**

```
const tCanBusTiming canTiming[] [static]
```

Initial value:

```
=
{
    { 5, 2 },
    { 6, 2 },
    { 6, 3 },
    { 7, 3 },
    { 8, 3 },
    { 9, 3 },
    { 9, 4 },
    { 10, 4 },
    { 11, 4 },
    { 12, 4 },
    { 12, 5 },
    { 13, 5 },
    { 14, 5 },
    { 15, 5 },
    { 15, 6 },
    { 16, 6 },
    { 16, 7 },
    { 16, 8 }
}
```

CAN bittiming table for dynamically calculating the bittiming settings.

According to the CAN protocol 1 bit-time can be made up of between 8..25 time quanta (TQ). The total TQ in a bit is SYNC + TSEG1 + TSEG2 with SYNC always being 1. The sample point is (SYNC + TSEG1) / (SYNC + TSEG1 + SEG2) * 100%. This array contains possible and valid time quanta configurations with a sample point between 68..78%.

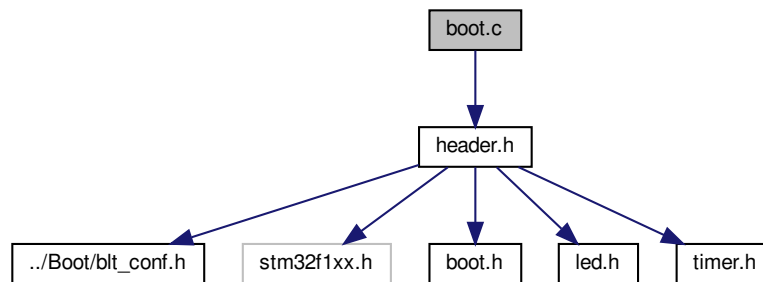
Referenced by `CanGetSpeedConfig()`.

7.188 boot.c File Reference

Demo program bootloader interface source file.

```
#include "header.h"
```

Include dependency graph for Demo/ARMCM3_STM32F1_Olimex_STM32P103_IAR/Prog/boot.c:



Data Structures

- struct [tCanBusTiming](#)
Structure type for grouping CAN bus timing related information.

Macros

- `#define RS232_CTO_RX_PACKET_TIMEOUT_MS (100u)`
Timeout time for the reception of a CTO packet. The timer is started upon reception of the first packet byte.

Functions

- static void [BootComRs232Init](#) (void)
Initializes the UART communication interface.
- static void [BootComRs232CheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- static void [BootComCanInit](#) (void)
Initializes the CAN communication interface.
- static void [BootComCanCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootComInit](#) (void)
Initializes the communication interface.
- void [BootComCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootActivate](#) (void)
Bootloader activation function.
- static unsigned char [Rs232ReceiveByte](#) (unsigned char *data)
Receives a communication interface byte if one is present.
- static unsigned char [CanGetSpeedConfig](#) (unsigned short baud, unsigned short *prescaler, unsigned char *tseg1, unsigned char *tseg2)
Search algorithm to match the desired baudrate to a possible bus timing configuration.

Variables

- static UART_HandleTypeDef [rs232Handle](#)
UART handle to be used in API calls.
- static const [tCanBusTiming](#) [canTiming](#) []
CAN bittiming table for dynamically calculating the bittiming settings.
- static CAN_HandleTypeDef [canHandle](#)
CAN handle to be used in API calls.

7.188.1 Detailed Description

Demo program bootloader interface source file.

7.188.2 Function Documentation

7.188.2.1 BootActivate()

```
void BootActivate (  
    void )
```

Bootloader activation function.

Returns

none.

Referenced by [BootComCanCheckActivationRequest\(\)](#), and [BootComRs232CheckActivationRequest\(\)](#).

7.188.2.2 BootComCanCheckActivationRequest()

```
static void BootComCanCheckActivationRequest (  
    void ) [static]
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

Referenced by [BootComCheckActivationRequest\(\)](#).

7.188.2.3 BootComCanInit()

```
static void BootComCanInit (
    void ) [static]
```

Initializes the CAN communication interface.

Returns

none.

Referenced by BootComInit().

7.188.2.4 BootComCheckActivationRequest()

```
void BootComCheckActivationRequest (
    void )
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.188.2.5 BootComInit()

```
void BootComInit (
    void )
```

Initializes the communication interface.

Returns

none.

7.188.2.6 BootComRs232CheckActivationRequest()

```
static void BootComRs232CheckActivationRequest (
    void ) [static]
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

Referenced by BootComCheckActivationRequest().

7.188.2.7 BootComRs232Init()

```
static void BootComRs232Init (  
    void ) [static]
```

Initializes the UART communication interface.

Returns

none.

Referenced by BootComInit().

7.188.2.8 CanGetSpeedConfig()

```
static unsigned char CanGetSpeedConfig (  
    unsigned short baud,  
    unsigned short * prescaler,  
    unsigned char * tseg1,  
    unsigned char * tseg2 ) [static]
```

Search algorithm to match the desired baudrate to a possible bus timing configuration.

Parameters

<i>baud</i>	The desired baudrate in kbps. Valid values are 10..1000.
<i>prescaler</i>	Pointer to where the value for the prescaler will be stored.
<i>tseg1</i>	Pointer to where the value for TSEG2 will be stored.
<i>tseg2</i>	Pointer to where the value for TSEG2 will be stored.

Returns

1 if the CAN bustiming register values were found, 0 otherwise.

Referenced by BootComCanInit().

7.188.2.9 Rs232ReceiveByte()

```
static unsigned char Rs232ReceiveByte (  
    unsigned char * data ) [static]
```

Receives a communication interface byte if one is present.

Parameters

<i>data</i>	Pointer to byte where the data is to be stored.
-------------	---

Returns

1 if a byte was received, 0 otherwise.

Referenced by `BootComRs232CheckActivationRequest()`.

7.188.3 Variable Documentation**7.188.3.1 canTiming**

```
const tCanBusTiming canTiming[] [static]
```

Initial value:

```
=
{
    { 5, 2 },
    { 6, 2 },
    { 6, 3 },
    { 7, 3 },
    { 8, 3 },
    { 9, 3 },
    { 9, 4 },
    { 10, 4 },
    { 11, 4 },
    { 12, 4 },
    { 12, 5 },
    { 13, 5 },
    { 14, 5 },
    { 15, 5 },
    { 15, 6 },
    { 16, 6 },
    { 16, 7 },
    { 16, 8 }
}
```

CAN bittiming table for dynamically calculating the bittiming settings.

According to the CAN protocol 1 bit-time can be made up of between 8..25 time quanta (TQ). The total TQ in a bit is SYNC + TSEG1 + TSEG2 with SYNC always being 1. The sample point is (SYNC + TSEG1) / (SYNC + TSEG1 + SEG2) * 100%. This array contains possible and valid time quanta configurations with a sample point between 68..78%.

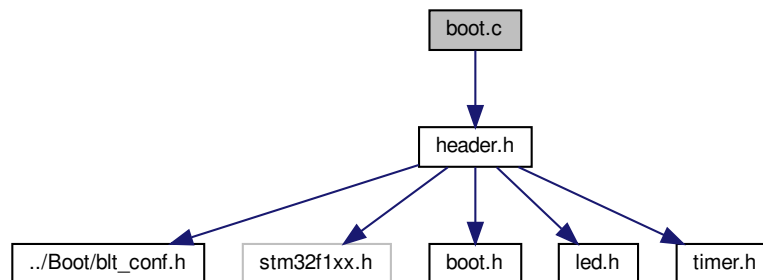
Referenced by `CanGetSpeedConfig()`.

7.189 boot.c File Reference

Demo program bootloader interface source file.

```
#include "header.h"
```

Include dependency graph for Demo/ARMCM3_STM32F1_Olimex_STM32P103_Keil/Prog/boot.c:



Data Structures

- struct [tCanBusTiming](#)
Structure type for grouping CAN bus timing related information.

Macros

- `#define RS232_CTO_RX_PACKET_TIMEOUT_MS (100u)`
Timeout time for the reception of a CTO packet. The timer is started upon reception of the first packet byte.

Functions

- static void [BootComRs232Init](#) (void)
Initializes the UART communication interface.
- static void [BootComRs232CheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- static void [BootComCanInit](#) (void)
Initializes the CAN communication interface.
- static void [BootComCanCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootComInit](#) (void)
Initializes the communication interface.
- void [BootComCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootActivate](#) (void)
Bootloader activation function.
- static unsigned char [Rs232ReceiveByte](#) (unsigned char *data)
Receives a communication interface byte if one is present.
- static unsigned char [CanGetSpeedConfig](#) (unsigned short baud, unsigned short *prescaler, unsigned char *tseg1, unsigned char *tseg2)
Search algorithm to match the desired baudrate to a possible bus timing configuration.

Variables

- static UART_HandleTypeDef [rs232Handle](#)
UART handle to be used in API calls.
- static const [tCanBusTiming](#) [canTiming](#) []
CAN bittiming table for dynamically calculating the bittiming settings.
- static CAN_HandleTypeDef [canHandle](#)
CAN handle to be used in API calls.

7.189.1 Detailed Description

Demo program bootloader interface source file.

7.189.2 Function Documentation

7.189.2.1 BootActivate()

```
void BootActivate (  
    void )
```

Bootloader activation function.

Returns

none.

Referenced by [BootComCanCheckActivationRequest\(\)](#), and [BootComRs232CheckActivationRequest\(\)](#).

7.189.2.2 BootComCanCheckActivationRequest()

```
static void BootComCanCheckActivationRequest (  
    void ) [static]
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

Referenced by [BootComCheckActivationRequest\(\)](#).

7.189.2.3 BootComCanInit()

```
static void BootComCanInit (
    void ) [static]
```

Initializes the CAN communication interface.

Returns

none.

Referenced by BootComInit().

7.189.2.4 BootComCheckActivationRequest()

```
void BootComCheckActivationRequest (
    void )
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.189.2.5 BootComInit()

```
void BootComInit (
    void )
```

Initializes the communication interface.

Returns

none.

7.189.2.6 BootComRs232CheckActivationRequest()

```
static void BootComRs232CheckActivationRequest (
    void ) [static]
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

Referenced by BootComCheckActivationRequest().

7.189.2.7 BootComRs232Init()

```
static void BootComRs232Init (  
    void ) [static]
```

Initializes the UART communication interface.

Returns

none.

Referenced by BootComInit().

7.189.2.8 CanGetSpeedConfig()

```
static unsigned char CanGetSpeedConfig (  
    unsigned short baud,  
    unsigned short * prescaler,  
    unsigned char * tseg1,  
    unsigned char * tseg2 ) [static]
```

Search algorithm to match the desired baudrate to a possible bus timing configuration.

Parameters

<i>baud</i>	The desired baudrate in kbps. Valid values are 10..1000.
<i>prescaler</i>	Pointer to where the value for the prescaler will be stored.
<i>tseg1</i>	Pointer to where the value for TSEG2 will be stored.
<i>tseg2</i>	Pointer to where the value for TSEG2 will be stored.

Returns

1 if the CAN bustiming register values were found, 0 otherwise.

Referenced by BootComCanInit().

7.189.2.9 Rs232ReceiveByte()

```
static unsigned char Rs232ReceiveByte (  
    unsigned char * data ) [static]
```

Receives a communication interface byte if one is present.

Parameters

<i>data</i>	Pointer to byte where the data is to be stored.
-------------	---

Returns

1 if a byte was received, 0 otherwise.

Referenced by `BootComRs232CheckActivationRequest()`.

7.189.3 Variable Documentation**7.189.3.1 canTiming**

```
const tCanBusTiming canTiming[] [static]
```

Initial value:

```
=  
{  
    { 5, 2 },  
    { 6, 2 },  
    { 6, 3 },  
    { 7, 3 },  
    { 8, 3 },  
    { 9, 3 },  
    { 9, 4 },  
    { 10, 4 },  
    { 11, 4 },  
    { 12, 4 },  
    { 12, 5 },  
    { 13, 5 },  
    { 14, 5 },  
    { 15, 5 },  
    { 15, 6 },  
    { 16, 6 },  
    { 16, 7 },  
    { 16, 8 }  
}
```

CAN bittiming table for dynamically calculating the bittiming settings.

According to the CAN protocol 1 bit-time can be made up of between 8..25 time quanta (TQ). The total TQ in a bit is SYNC + TSEG1 + TSEG2 with SYNC always being 1. The sample point is (SYNC + TSEG1) / (SYNC + TSEG1 + SEG2) * 100%. This array contains possible and valid time quanta configurations with a sample point between 68..78%.

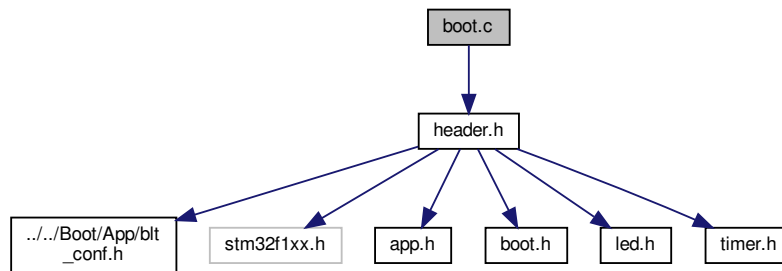
Referenced by `CanGetSpeedConfig()`.

7.190 boot.c File Reference

Demo program bootloader interface source file.

```
#include "header.h"
```

Include dependency graph for Demo/ARMCM3_STM32F1_Olimexino_STM32_CubeIDE/Prog/App/boot.c:



Data Structures

- struct [tCanBusTiming](#)
Structure type for grouping CAN bus timing related information.

Functions

- static void [BootComCanInit](#) (void)
Initializes the CAN communication interface.
- static void [BootComCanCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootComInit](#) (void)
Initializes the communication interface.
- void [BootComCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootActivate](#) (void)
Bootloader activation function.
- static unsigned char [CanGetSpeedConfig](#) (unsigned short baud, unsigned short *prescaler, unsigned char *tseg1, unsigned char *tseg2)
Search algorithm to match the desired baudrate to a possible bus timing configuration.

Variables

- static const [tCanBusTiming](#) [canTiming](#) []
CAN bittiming table for dynamically calculating the bittiming settings.
- static CAN_HandleTypeDef [canHandle](#)
CAN handle to be used in API calls.

7.190.1 Detailed Description

Demo program bootloader interface source file.

7.190.2 Function Documentation

7.190.2.1 BootActivate()

```
void BootActivate (
    void )
```

Bootloader activation function.

Returns

none.

Referenced by BootComCanCheckActivationRequest().

7.190.2.2 BootComCanCheckActivationRequest()

```
static void BootComCanCheckActivationRequest (
    void ) [static]
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

Referenced by BootComCheckActivationRequest().

7.190.2.3 BootComCanInit()

```
static void BootComCanInit (
    void ) [static]
```

Initializes the CAN communication interface.

Returns

none.

Referenced by BootComInit().

7.190.2.4 BootComCheckActivationRequest()

```
void BootComCheckActivationRequest (
    void )
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.190.2.5 BootComInit()

```
void BootComInit (
    void )
```

Initializes the communication interface.

Returns

none.

7.190.2.6 CanGetSpeedConfig()

```
static unsigned char CanGetSpeedConfig (
    unsigned short baud,
    unsigned short * prescaler,
    unsigned char * tseg1,
    unsigned char * tseg2 ) [static]
```

Search algorithm to match the desired baudrate to a possible bus timing configuration.

Parameters

<i>baud</i>	The desired baudrate in kbps. Valid values are 10..1000.
<i>prescaler</i>	Pointer to where the value for the prescaler will be stored.
<i>tseg1</i>	Pointer to where the value for TSEG2 will be stored.
<i>tseg2</i>	Pointer to where the value for TSEG2 will be stored.

Returns

1 if the CAN bustiming register values were found, 0 otherwise.

Referenced by BootComCanInit().

7.190.3 Variable Documentation

7.190.3.1 canTiming

```
const tCanBusTiming canTiming[] [static]
```

Initial value:

```
=
{
    { 5, 2 },
    { 6, 2 },
    { 6, 3 },
    { 7, 3 },
    { 8, 3 },
    { 9, 3 },
    { 9, 4 },
    { 10, 4 },
    { 11, 4 },
    { 12, 4 },
    { 12, 5 },
    { 13, 5 },
    { 14, 5 },
    { 15, 5 },
    { 15, 6 },
    { 16, 6 },
    { 16, 7 },
    { 16, 8 }
}
```

CAN bittiming table for dynamically calculating the bittiming settings.

According to the CAN protocol 1 bit-time can be made up of between 8..25 time quanta (TQ). The total TQ in a bit is SYNC + TSEG1 + TSEG2 with SYNC always being 1. The sample point is (SYNC + TSEG1) / (SYNC + TSEG1 + SEG2) * 100%. This array contains possible and valid time quanta configurations with a sample point between 68..78%.

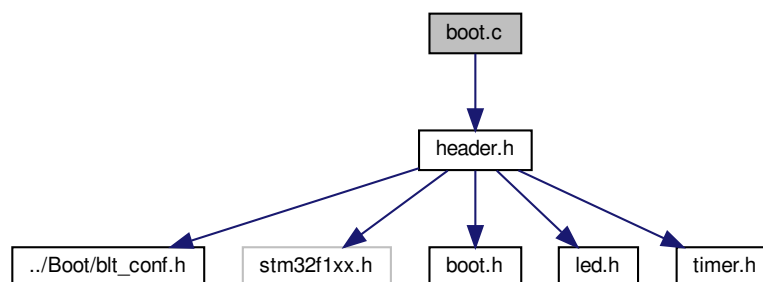
Referenced by CanGetSpeedConfig().

7.191 boot.c File Reference

Demo program bootloader interface source file.

```
#include "header.h"
```

Include dependency graph for Demo/ARMCM3_STM32F1_Olimexino_STM32_GCC/Prog/boot.c:



Data Structures

- struct [tCanBusTiming](#)
Structure type for grouping CAN bus timing related information.

Functions

- static void [BootComCanInit](#) (void)
Initializes the CAN communication interface.
- static void [BootComCanCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootComInit](#) (void)
Initializes the communication interface.
- void [BootComCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootActivate](#) (void)
Bootloader activation function.
- static unsigned char [CanGetSpeedConfig](#) (unsigned short baud, unsigned short *prescaler, unsigned char *tseg1, unsigned char *tseg2)
Search algorithm to match the desired baudrate to a possible bus timing configuration.

Variables

- static const [tCanBusTiming](#) [canTiming](#) []
CAN bittiming table for dynamically calculating the bittiming settings.
- static CAN_HandleTypeDef [canHandle](#)
CAN handle to be used in API calls.

7.191.1 Detailed Description

Demo program bootloader interface source file.

7.191.2 Function Documentation

7.191.2.1 [BootActivate\(\)](#)

```
void BootActivate (  
    void )
```

Bootloader activation function.

Returns

none.

Referenced by [BootComCanCheckActivationRequest\(\)](#).

7.191.2.2 BootComCanCheckActivationRequest()

```
static void BootComCanCheckActivationRequest (  
    void ) [static]
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

Referenced by BootComCheckActivationRequest().

7.191.2.3 BootComCanInit()

```
static void BootComCanInit (  
    void ) [static]
```

Initializes the CAN communication interface.

Returns

none.

Referenced by BootComInit().

7.191.2.4 BootComCheckActivationRequest()

```
void BootComCheckActivationRequest (  
    void )
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.191.2.5 BootComInit()

```
void BootComInit (
    void )
```

Initializes the communication interface.

Returns

none.

7.191.2.6 CanGetSpeedConfig()

```
static unsigned char CanGetSpeedConfig (
    unsigned short baud,
    unsigned short * prescaler,
    unsigned char * tseg1,
    unsigned char * tseg2 ) [static]
```

Search algorithm to match the desired baudrate to a possible bus timing configuration.

Parameters

<i>baud</i>	The desired baudrate in kbps. Valid values are 10..1000.
<i>prescaler</i>	Pointer to where the value for the prescaler will be stored.
<i>tseg1</i>	Pointer to where the value for TSEG2 will be stored.
<i>tseg2</i>	Pointer to where the value for TSEG2 will be stored.

Returns

1 if the CAN bustiming register values were found, 0 otherwise.

Referenced by BootComCanInit().

7.191.3 Variable Documentation

7.191.3.1 canTiming

```
const tCanBusTiming canTiming[] [static]
```

Initial value:

```

=
{
    { 5, 2 },
    { 6, 2 },
    { 6, 3 },
    { 7, 3 },
    { 8, 3 },
    { 9, 3 },
    { 9, 4 },
    { 10, 4 },
    { 11, 4 },
    { 12, 4 },
    { 12, 5 },
    { 13, 5 },
    { 14, 5 },
    { 15, 5 },
    { 15, 6 },
    { 16, 6 },
    { 16, 7 },
    { 16, 8 }
}

```

CAN bittiming table for dynamically calculating the bittiming settings.

According to the CAN protocol 1 bit-time can be made up of between 8..25 time quanta (TQ). The total TQ in a bit is SYNC + TSEG1 + TSEG2 with SYNC always being 1. The sample point is (SYNC + TSEG1) / (SYNC + TSEG1 + SEG2) * 100%. This array contains possible and valid time quanta configurations with a sample point between 68..78%.

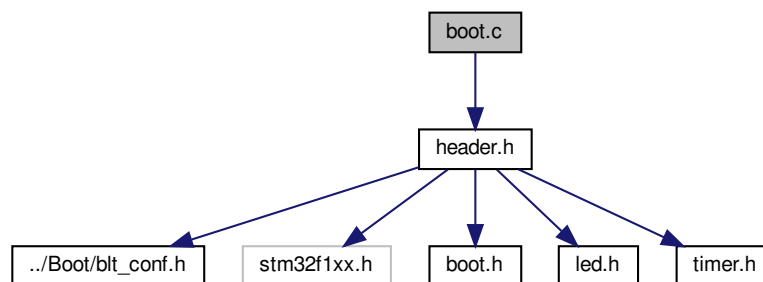
Referenced by CanGetSpeedConfig().

7.192 boot.c File Reference

Demo program bootloader interface source file.

```
#include "header.h"
```

Include dependency graph for Demo/ARMCM3_STM32F1_Olimexino_STM32_IAR/Prog/boot.c:



Data Structures

- struct [tCanBusTiming](#)

Structure type for grouping CAN bus timing related information.

Functions

- static void [BootComCanInit](#) (void)
Initializes the CAN communication interface.
- static void [BootComCanCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootComInit](#) (void)
Initializes the communication interface.
- void [BootComCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootActivate](#) (void)
Bootloader activation function.
- static unsigned char [CanGetSpeedConfig](#) (unsigned short baud, unsigned short *prescaler, unsigned char *tseg1, unsigned char *tseg2)
Search algorithm to match the desired baudrate to a possible bus timing configuration.

Variables

- static const [tCanBusTiming](#) [canTiming](#) []
CAN bittiming table for dynamically calculating the bittiming settings.
- static CAN_HandleTypeDef [canHandle](#)
CAN handle to be used in API calls.

7.192.1 Detailed Description

Demo program bootloader interface source file.

7.192.2 Function Documentation

7.192.2.1 [BootActivate\(\)](#)

```
void BootActivate (  
    void )
```

Bootloader activation function.

Returns

none.

Referenced by [BootComCanCheckActivationRequest\(\)](#).

7.192.2.2 BootComCanCheckActivationRequest()

```
static void BootComCanCheckActivationRequest (  
    void ) [static]
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

Referenced by BootComCheckActivationRequest().

7.192.2.3 BootComCanInit()

```
static void BootComCanInit (  
    void ) [static]
```

Initializes the CAN communication interface.

Returns

none.

Referenced by BootComInit().

7.192.2.4 BootComCheckActivationRequest()

```
void BootComCheckActivationRequest (  
    void )
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.192.2.5 BootComInit()

```
void BootComInit (
    void )
```

Initializes the communication interface.

Returns

none.

7.192.2.6 CanGetSpeedConfig()

```
static unsigned char CanGetSpeedConfig (
    unsigned short baud,
    unsigned short * prescaler,
    unsigned char * tseg1,
    unsigned char * tseg2 ) [static]
```

Search algorithm to match the desired baudrate to a possible bus timing configuration.

Parameters

<i>baud</i>	The desired baudrate in kbps. Valid values are 10..1000.
<i>prescaler</i>	Pointer to where the value for the prescaler will be stored.
<i>tseg1</i>	Pointer to where the value for TSEG2 will be stored.
<i>tseg2</i>	Pointer to where the value for TSEG2 will be stored.

Returns

1 if the CAN bustiming register values were found, 0 otherwise.

Referenced by BootComCanInit().

7.192.3 Variable Documentation

7.192.3.1 canTiming

```
const tCanBusTiming canTiming[] [static]
```

Initial value:

```

=
{
    { 5, 2 },
    { 6, 2 },
    { 6, 3 },
    { 7, 3 },
    { 8, 3 },
    { 9, 3 },
    { 9, 4 },
    { 10, 4 },
    { 11, 4 },
    { 12, 4 },
    { 12, 5 },
    { 13, 5 },
    { 14, 5 },
    { 15, 5 },
    { 15, 6 },
    { 16, 6 },
    { 16, 7 },
    { 16, 8 }
}

```

CAN bittiming table for dynamically calculating the bittiming settings.

According to the CAN protocol 1 bit-time can be made up of between 8..25 time quanta (TQ). The total TQ in a bit is SYNC + TSEG1 + TSEG2 with SYNC always being 1. The sample point is (SYNC + TSEG1) / (SYNC + TSEG1 + SEG2) * 100%. This array contains possible and valid time quanta configurations with a sample point between 68..78%.

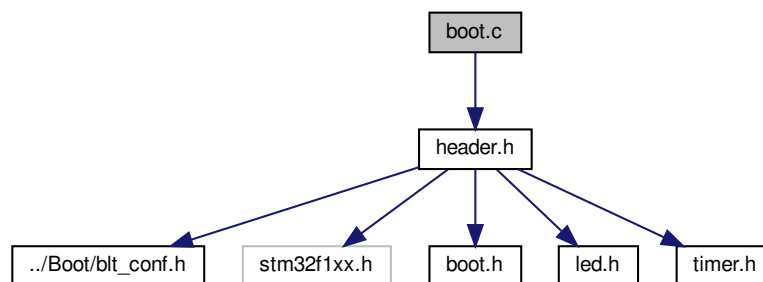
Referenced by `CanGetSpeedConfig()`.

7.193 boot.c File Reference

Demo program bootloader interface source file.

```
#include "header.h"
```

Include dependency graph for Demo/ARMCM3_STM32F1_Olimexino_STM32_Keil/Prog/boot.c:



Data Structures

- [struct `tCanBusTiming`](#)

Structure type for grouping CAN bus timing related information.

Functions

- static void [BootComCanInit](#) (void)
Initializes the CAN communication interface.
- static void [BootComCanCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootComInit](#) (void)
Initializes the communication interface.
- void [BootComCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootActivate](#) (void)
Bootloader activation function.
- static unsigned char [CanGetSpeedConfig](#) (unsigned short baud, unsigned short *prescaler, unsigned char *tseg1, unsigned char *tseg2)
Search algorithm to match the desired baudrate to a possible bus timing configuration.

Variables

- static const [tCanBusTiming](#) [canTiming](#) []
CAN bittiming table for dynamically calculating the bittiming settings.
- static CAN_HandleTypeDef [canHandle](#)
CAN handle to be used in API calls.

7.193.1 Detailed Description

Demo program bootloader interface source file.

7.193.2 Function Documentation

7.193.2.1 [BootActivate\(\)](#)

```
void BootActivate (  
    void )
```

Bootloader activation function.

Returns

none.

Referenced by [BootComCanCheckActivationRequest\(\)](#).

7.193.2.2 BootComCanCheckActivationRequest()

```
static void BootComCanCheckActivationRequest (  
    void ) [static]
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

Referenced by BootComCheckActivationRequest().

7.193.2.3 BootComCanInit()

```
static void BootComCanInit (  
    void ) [static]
```

Initializes the CAN communication interface.

Returns

none.

Referenced by BootComInit().

7.193.2.4 BootComCheckActivationRequest()

```
void BootComCheckActivationRequest (  
    void )
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.193.2.5 BootComInit()

```
void BootComInit (
    void )
```

Initializes the communication interface.

Returns

none.

7.193.2.6 CanGetSpeedConfig()

```
static unsigned char CanGetSpeedConfig (
    unsigned short baud,
    unsigned short * prescaler,
    unsigned char * tseg1,
    unsigned char * tseg2 ) [static]
```

Search algorithm to match the desired baudrate to a possible bus timing configuration.

Parameters

<i>baud</i>	The desired baudrate in kbps. Valid values are 10..1000.
<i>prescaler</i>	Pointer to where the value for the prescaler will be stored.
<i>tseg1</i>	Pointer to where the value for TSEG2 will be stored.
<i>tseg2</i>	Pointer to where the value for TSEG2 will be stored.

Returns

1 if the CAN bustiming register values were found, 0 otherwise.

Referenced by BootComCanInit().

7.193.3 Variable Documentation

7.193.3.1 canTiming

```
const tCanBusTiming canTiming[] [static]
```

Initial value:

```

=
{
    { 5, 2 },
    { 6, 2 },
    { 6, 3 },
    { 7, 3 },
    { 8, 3 },
    { 9, 3 },
    { 9, 4 },
    { 10, 4 },
    { 11, 4 },
    { 12, 4 },
    { 12, 5 },
    { 13, 5 },
    { 14, 5 },
    { 15, 5 },
    { 15, 6 },
    { 16, 6 },
    { 16, 7 },
    { 16, 8 }
}

```

CAN bittiming table for dynamically calculating the bittiming settings.

According to the CAN protocol 1 bit-time can be made up of between 8..25 time quanta (TQ). The total TQ in a bit is SYNC + TSEG1 + TSEG2 with SYNC always being 1. The sample point is (SYNC + TSEG1) / (SYNC + TSEG1 + SEG2) * 100%. This array contains possible and valid time quanta configurations with a sample point between 68..78%.

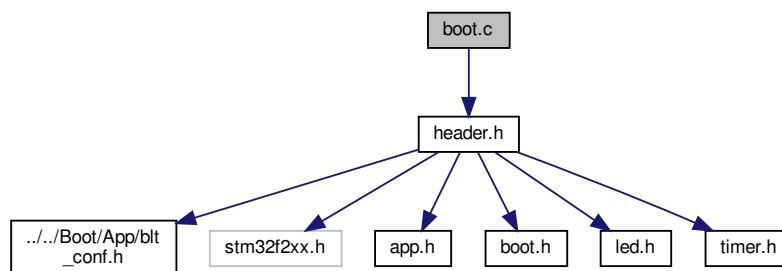
Referenced by CanGetSpeedConfig().

7.194 boot.c File Reference

Demo program bootloader interface source file.

```
#include "header.h"
```

Include dependency graph for Demo/ARMCM3_STM32F2_Olimex_STM32P207_CubeIDE/Prog/App/boot.c:



Data Structures

- struct [tCanBusTiming](#)

Structure type for grouping CAN bus timing related information.

Macros

- `#define RS232_CTO_RX_PACKET_TIMEOUT_MS (100u)`
Timeout time for the reception of a CTO packet. The timer is started upon reception of the first packet byte.

Functions

- static void `BootComRs232Init` (void)
Initializes the UART communication interface.
- static void `BootComRs232CheckActivationRequest` (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- static void `BootComCanInit` (void)
Initializes the CAN communication interface.
- static void `BootComCanCheckActivationRequest` (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void `BootComInit` (void)
Initializes the communication interface.
- void `BootComCheckActivationRequest` (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void `BootActivate` (void)
Bootloader activation function.
- static unsigned char `Rs232ReceiveByte` (unsigned char *data)
Receives a communication interface byte if one is present.
- static unsigned char `CanGetSpeedConfig` (unsigned short baud, unsigned short *prescaler, unsigned char *tseg1, unsigned char *tseg2)
Search algorithm to match the desired baudrate to a possible bus timing configuration.

Variables

- static UART_HandleTypeDef `rs232Handle`
UART handle to be used in API calls.
- static const `tCanBusTiming` `canTiming` []
CAN bittiming table for dynamically calculating the bittiming settings.
- static CAN_HandleTypeDef `canHandle`
CAN handle to be used in API calls.

7.194.1 Detailed Description

Demo program bootloader interface source file.

7.194.2 Function Documentation

7.194.2.1 BootActivate()

```
void BootActivate (
    void )
```

Bootloader activation function.

Returns

none.

Referenced by BootComCanCheckActivationRequest(), and BootComRs232CheckActivationRequest().

7.194.2.2 BootComCanCheckActivationRequest()

```
static void BootComCanCheckActivationRequest (
    void ) [static]
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

Referenced by BootComCheckActivationRequest().

7.194.2.3 BootComCanInit()

```
static void BootComCanInit (
    void ) [static]
```

Initializes the CAN communication interface.

Returns

none.

Referenced by BootComInit().

7.194.2.4 BootComCheckActivationRequest()

```
void BootComCheckActivationRequest (
    void )
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.194.2.5 BootComInit()

```
void BootComInit (
    void )
```

Initializes the communication interface.

Returns

none.

7.194.2.6 BootComRs232CheckActivationRequest()

```
static void BootComRs232CheckActivationRequest (
    void ) [static]
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

Referenced by BootComCheckActivationRequest().

7.194.2.7 BootComRs232Init()

```
static void BootComRs232Init (
    void ) [static]
```

Initializes the UART communication interface.

Returns

none.

Referenced by BootComInit().

7.194.2.8 CanGetSpeedConfig()

```
static unsigned char CanGetSpeedConfig (
    unsigned short baud,
    unsigned short * prescaler,
    unsigned char * tseg1,
    unsigned char * tseg2 ) [static]
```

Search algorithm to match the desired baudrate to a possible bus timing configuration.

Parameters

<i>baud</i>	The desired baudrate in kbps. Valid values are 10..1000.
<i>prescaler</i>	Pointer to where the value for the prescaler will be stored.
<i>tseg1</i>	Pointer to where the value for TSEG2 will be stored.
<i>tseg2</i>	Pointer to where the value for TSEG2 will be stored.

Returns

1 if the CAN bustiming register values were found, 0 otherwise.

Referenced by BootComCanInit().

7.194.2.9 Rs232ReceiveByte()

```
static unsigned char Rs232ReceiveByte (
    unsigned char * data ) [static]
```

Receives a communication interface byte if one is present.

Parameters

<i>data</i>	Pointer to byte where the data is to be stored.
-------------	---

Returns

1 if a byte was received, 0 otherwise.

Referenced by BootComRs232CheckActivationRequest().

7.194.3 Variable Documentation

7.194.3.1 canTiming

```
const tCanBusTiming canTiming[] [static]
```

Initial value:

```
=
{
    { 5, 2 },
    { 6, 2 },
    { 6, 3 },
    { 7, 3 },
    { 8, 3 },
    { 9, 3 },
    { 9, 4 },
    { 10, 4 },
    { 11, 4 },
    { 12, 4 },
    { 12, 5 },
    { 13, 5 },
    { 14, 5 },
    { 15, 5 },
    { 15, 6 },
    { 16, 6 },
    { 16, 7 },
    { 16, 8 }
}
```

CAN bittiming table for dynamically calculating the bittiming settings.

According to the CAN protocol 1 bit-time can be made up of between 8..25 time quanta (TQ). The total TQ in a bit is SYNC + TSEG1 + TSEG2 with SYNC always being 1. The sample point is (SYNC + TSEG1) / (SYNC + TSEG1 + SEG2) * 100%. This array contains possible and valid time quanta configurations with a sample point between 68..78%.

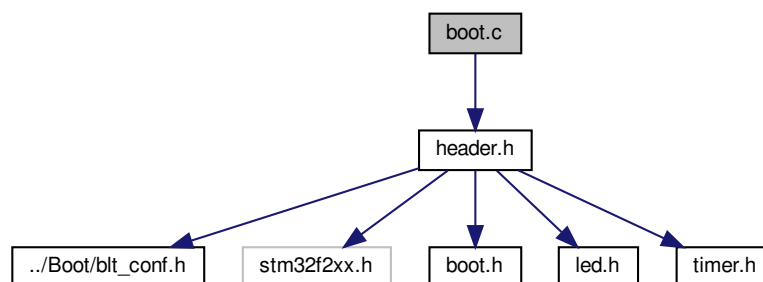
Referenced by CanGetSpeedConfig().

7.195 boot.c File Reference

Demo program bootloader interface source file.

```
#include "header.h"
```

Include dependency graph for Demo/ARMCM3_STM32F2_Olimex_STM32P207_GCC/Prog/boot.c:



Data Structures

- struct [tCanBusTiming](#)
Structure type for grouping CAN bus timing related information.

Macros

- #define [RS232_CTO_RX_PACKET_TIMEOUT_MS](#) (100u)
Timeout time for the reception of a CTO packet. The timer is started upon reception of the first packet byte.

Functions

- static void [BootComRs232Init](#) (void)
Initializes the UART communication interface.
- static void [BootComRs232CheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- static void [BootComCanInit](#) (void)
Initializes the CAN communication interface.
- static void [BootComCanCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootComInit](#) (void)
Initializes the communication interface.
- void [BootComCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootActivate](#) (void)
Bootloader activation function.
- static unsigned char [Rs232ReceiveByte](#) (unsigned char *data)
Receives a communication interface byte if one is present.
- static unsigned char [CanGetSpeedConfig](#) (unsigned short baud, unsigned short *prescaler, unsigned char *tseg1, unsigned char *tseg2)
Search algorithm to match the desired baudrate to a possible bus timing configuration.

Variables

- static UART_HandleTypeDef [rs232Handle](#)
UART handle to be used in API calls.
- static const [tCanBusTiming](#) [canTiming](#) []
CAN bittiming table for dynamically calculating the bittiming settings.
- static CAN_HandleTypeDef [canHandle](#)
CAN handle to be used in API calls.

7.195.1 Detailed Description

Demo program bootloader interface source file.

7.195.2 Function Documentation

7.195.2.1 BootActivate()

```
void BootActivate (
    void )
```

Bootloader activation function.

Returns

none.

Referenced by `BootComCanCheckActivationRequest()`, and `BootComRs232CheckActivationRequest()`.

7.195.2.2 BootComCanCheckActivationRequest()

```
static void BootComCanCheckActivationRequest (
    void ) [static]
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

Referenced by `BootComCheckActivationRequest()`.

7.195.2.3 BootComCanInit()

```
static void BootComCanInit (
    void ) [static]
```

Initializes the CAN communication interface.

Returns

none.

Referenced by `BootComInit()`.

7.195.2.4 BootComCheckActivationRequest()

```
void BootComCheckActivationRequest (  
    void )
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.195.2.5 BootComInit()

```
void BootComInit (  
    void )
```

Initializes the communication interface.

Returns

none.

7.195.2.6 BootComRs232CheckActivationRequest()

```
static void BootComRs232CheckActivationRequest (  
    void ) [static]
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

Referenced by BootComCheckActivationRequest().

7.195.2.7 BootComRs232Init()

```
static void BootComRs232Init (  
    void ) [static]
```

Initializes the UART communication interface.

Returns

none.

Referenced by BootComInit().

7.195.2.8 CanGetSpeedConfig()

```
static unsigned char CanGetSpeedConfig (  
    unsigned short baud,  
    unsigned short * prescaler,  
    unsigned char * tseg1,  
    unsigned char * tseg2 ) [static]
```

Search algorithm to match the desired baudrate to a possible bus timing configuration.

Parameters

<i>baud</i>	The desired baudrate in kbps. Valid values are 10..1000.
<i>prescaler</i>	Pointer to where the value for the prescaler will be stored.
<i>tseg1</i>	Pointer to where the value for TSEG2 will be stored.
<i>tseg2</i>	Pointer to where the value for TSEG2 will be stored.

Returns

1 if the CAN bustiming register values were found, 0 otherwise.

Referenced by BootComCanInit().

7.195.2.9 Rs232ReceiveByte()

```
static unsigned char Rs232ReceiveByte (  
    unsigned char * data ) [static]
```

Receives a communication interface byte if one is present.

Parameters

<i>data</i>	Pointer to byte where the data is to be stored.
-------------	---

Returns

1 if a byte was received, 0 otherwise.

Referenced by BootComRs232CheckActivationRequest().

7.195.3 Variable Documentation

7.195.3.1 canTiming

```
const tCanBusTiming canTiming[] [static]
```

Initial value:

```
=
{
    { 5, 2 },
    { 6, 2 },
    { 6, 3 },
    { 7, 3 },
    { 8, 3 },
    { 9, 3 },
    { 9, 4 },
    { 10, 4 },
    { 11, 4 },
    { 12, 4 },
    { 12, 5 },
    { 13, 5 },
    { 14, 5 },
    { 15, 5 },
    { 15, 6 },
    { 16, 6 },
    { 16, 7 },
    { 16, 8 }
}
```

CAN bittiming table for dynamically calculating the bittiming settings.

According to the CAN protocol 1 bit-time can be made up of between 8..25 time quanta (TQ). The total TQ in a bit is SYNC + TSEG1 + TSEG2 with SYNC always being 1. The sample point is (SYNC + TSEG1) / (SYNC + TSEG1 + SEG2) * 100%. This array contains possible and valid time quanta configurations with a sample point between 68..78%.

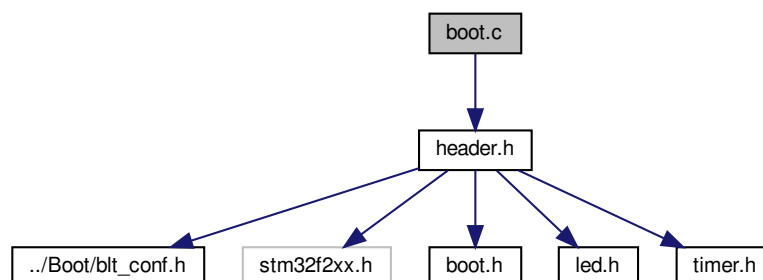
Referenced by CanGetSpeedConfig().

7.196 boot.c File Reference

Demo program bootloader interface source file.

```
#include "header.h"
```

Include dependency graph for Demo/ARMCM3_STM32F2_Olimex_STM32P207_IAR/Prog/boot.c:



Data Structures

- struct [tCanBusTiming](#)
Structure type for grouping CAN bus timing related information.

Macros

- #define [RS232_CTO_RX_PACKET_TIMEOUT_MS](#) (100u)
Timeout time for the reception of a CTO packet. The timer is started upon reception of the first packet byte.

Functions

- static void [BootComRs232Init](#) (void)
Initializes the UART communication interface.
- static void [BootComRs232CheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- static void [BootComCanInit](#) (void)
Initializes the CAN communication interface.
- static void [BootComCanCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootComInit](#) (void)
Initializes the communication interface.
- void [BootComCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootActivate](#) (void)
Bootloader activation function.
- static unsigned char [Rs232ReceiveByte](#) (unsigned char *data)
Receives a communication interface byte if one is present.
- static unsigned char [CanGetSpeedConfig](#) (unsigned short baud, unsigned short *prescaler, unsigned char *tseg1, unsigned char *tseg2)
Search algorithm to match the desired baudrate to a possible bus timing configuration.

Variables

- static UART_HandleTypeDef [rs232Handle](#)
UART handle to be used in API calls.
- static const [tCanBusTiming](#) [canTiming](#) []
CAN bittiming table for dynamically calculating the bittiming settings.
- static CAN_HandleTypeDef [canHandle](#)
CAN handle to be used in API calls.

7.196.1 Detailed Description

Demo program bootloader interface source file.

7.196.2 Function Documentation

7.196.2.1 BootActivate()

```
void BootActivate (
    void )
```

Bootloader activation function.

Returns

none.

Referenced by BootComCanCheckActivationRequest(), and BootComRs232CheckActivationRequest().

7.196.2.2 BootComCanCheckActivationRequest()

```
static void BootComCanCheckActivationRequest (
    void ) [static]
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

Referenced by BootComCheckActivationRequest().

7.196.2.3 BootComCanInit()

```
static void BootComCanInit (
    void ) [static]
```

Initializes the CAN communication interface.

Returns

none.

Referenced by BootComInit().

7.196.2.4 BootComCheckActivationRequest()

```
void BootComCheckActivationRequest (
    void )
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.196.2.5 BootComInit()

```
void BootComInit (
    void )
```

Initializes the communication interface.

Returns

none.

7.196.2.6 BootComRs232CheckActivationRequest()

```
static void BootComRs232CheckActivationRequest (
    void ) [static]
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

Referenced by BootComCheckActivationRequest().

7.196.2.7 BootComRs232Init()

```
static void BootComRs232Init (
    void ) [static]
```

Initializes the UART communication interface.

Returns

none.

Referenced by BootComInit().

7.196.2.8 CanGetSpeedConfig()

```
static unsigned char CanGetSpeedConfig (  
    unsigned short baud,  
    unsigned short * prescaler,  
    unsigned char * tseg1,  
    unsigned char * tseg2 ) [static]
```

Search algorithm to match the desired baudrate to a possible bus timing configuration.

Parameters

<i>baud</i>	The desired baudrate in kbps. Valid values are 10..1000.
<i>prescaler</i>	Pointer to where the value for the prescaler will be stored.
<i>tseg1</i>	Pointer to where the value for TSEG2 will be stored.
<i>tseg2</i>	Pointer to where the value for TSEG2 will be stored.

Returns

1 if the CAN bustiming register values were found, 0 otherwise.

Referenced by BootComCanInit().

7.196.2.9 Rs232ReceiveByte()

```
static unsigned char Rs232ReceiveByte (  
    unsigned char * data ) [static]
```

Receives a communication interface byte if one is present.

Parameters

<i>data</i>	Pointer to byte where the data is to be stored.
-------------	---

Returns

1 if a byte was received, 0 otherwise.

Referenced by BootComRs232CheckActivationRequest().

7.196.3 Variable Documentation

7.196.3.1 canTiming

```
const tCanBusTiming canTiming[] [static]
```

Initial value:

```
=
{
    { 5, 2 },
    { 6, 2 },
    { 6, 3 },
    { 7, 3 },
    { 8, 3 },
    { 9, 3 },
    { 9, 4 },
    { 10, 4 },
    { 11, 4 },
    { 12, 4 },
    { 12, 5 },
    { 13, 5 },
    { 14, 5 },
    { 15, 5 },
    { 15, 6 },
    { 16, 6 },
    { 16, 7 },
    { 16, 8 }
}
```

CAN bittiming table for dynamically calculating the bittiming settings.

According to the CAN protocol 1 bit-time can be made up of between 8..25 time quanta (TQ). The total TQ in a bit is SYNC + TSEG1 + TSEG2 with SYNC always being 1. The sample point is (SYNC + TSEG1) / (SYNC + TSEG1 + SEG2) * 100%. This array contains possible and valid time quanta configurations with a sample point between 68..78%.

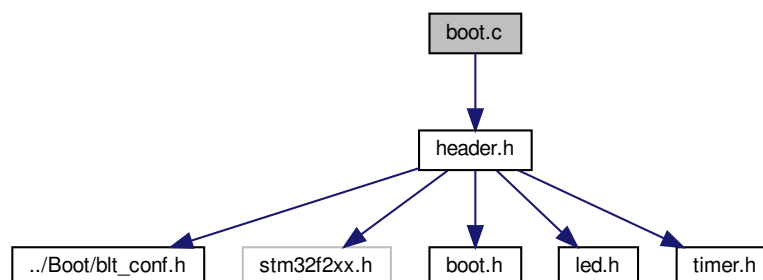
Referenced by CanGetSpeedConfig().

7.197 boot.c File Reference

Demo program bootloader interface source file.

```
#include "header.h"
```

Include dependency graph for Demo/ARMCM3_STM32F2_Olimex_STM32P207_Keil/Prog/boot.c:



Data Structures

- struct [tCanBusTiming](#)
Structure type for grouping CAN bus timing related information.

Macros

- #define [RS232_CTO_RX_PACKET_TIMEOUT_MS](#) (100u)
Timeout time for the reception of a CTO packet. The timer is started upon reception of the first packet byte.

Functions

- static void [BootComRs232Init](#) (void)
Initializes the UART communication interface.
- static void [BootComRs232CheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- static void [BootComCanInit](#) (void)
Initializes the CAN communication interface.
- static void [BootComCanCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootComInit](#) (void)
Initializes the communication interface.
- void [BootComCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootActivate](#) (void)
Bootloader activation function.
- static unsigned char [Rs232ReceiveByte](#) (unsigned char *data)
Receives a communication interface byte if one is present.
- static unsigned char [CanGetSpeedConfig](#) (unsigned short baud, unsigned short *prescaler, unsigned char *tseg1, unsigned char *tseg2)
Search algorithm to match the desired baudrate to a possible bus timing configuration.

Variables

- static UART_HandleTypeDef [rs232Handle](#)
UART handle to be used in API calls.
- static const [tCanBusTiming](#) [canTiming](#) []
CAN bittiming table for dynamically calculating the bittiming settings.
- static CAN_HandleTypeDef [canHandle](#)
CAN handle to be used in API calls.

7.197.1 Detailed Description

Demo program bootloader interface source file.

7.197.2 Function Documentation

7.197.2.1 BootActivate()

```
void BootActivate (
    void )
```

Bootloader activation function.

Returns

none.

Referenced by `BootComCanCheckActivationRequest()`, and `BootComRs232CheckActivationRequest()`.

7.197.2.2 BootComCanCheckActivationRequest()

```
static void BootComCanCheckActivationRequest (
    void ) [static]
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

Referenced by `BootComCheckActivationRequest()`.

7.197.2.3 BootComCanInit()

```
static void BootComCanInit (
    void ) [static]
```

Initializes the CAN communication interface.

Returns

none.

Referenced by `BootComInit()`.

7.197.2.4 BootComCheckActivationRequest()

```
void BootComCheckActivationRequest (  
    void )
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.197.2.5 BootComInit()

```
void BootComInit (  
    void )
```

Initializes the communication interface.

Returns

none.

7.197.2.6 BootComRs232CheckActivationRequest()

```
static void BootComRs232CheckActivationRequest (  
    void ) [static]
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

Referenced by BootComCheckActivationRequest().

7.197.2.7 BootComRs232Init()

```
static void BootComRs232Init (  
    void ) [static]
```

Initializes the UART communication interface.

Returns

none.

Referenced by BootComInit().

7.197.2.8 CanGetSpeedConfig()

```
static unsigned char CanGetSpeedConfig (
    unsigned short baud,
    unsigned short * prescaler,
    unsigned char * tseg1,
    unsigned char * tseg2 ) [static]
```

Search algorithm to match the desired baudrate to a possible bus timing configuration.

Parameters

<i>baud</i>	The desired baudrate in kbps. Valid values are 10..1000.
<i>prescaler</i>	Pointer to where the value for the prescaler will be stored.
<i>tseg1</i>	Pointer to where the value for TSEG2 will be stored.
<i>tseg2</i>	Pointer to where the value for TSEG2 will be stored.

Returns

1 if the CAN bustiming register values were found, 0 otherwise.

Referenced by BootComCanInit().

7.197.2.9 Rs232ReceiveByte()

```
static unsigned char Rs232ReceiveByte (
    unsigned char * data ) [static]
```

Receives a communication interface byte if one is present.

Parameters

<i>data</i>	Pointer to byte where the data is to be stored.
-------------	---

Returns

1 if a byte was received, 0 otherwise.

Referenced by BootComRs232CheckActivationRequest().

7.197.3 Variable Documentation

7.197.3.1 canTiming

```
const tCanBusTiming canTiming[] [static]
```

Initial value:

```
=
{
    { 5, 2 },
    { 6, 2 },
    { 6, 3 },
    { 7, 3 },
    { 8, 3 },
    { 9, 3 },
    { 9, 4 },
    { 10, 4 },
    { 11, 4 },
    { 12, 4 },
    { 12, 5 },
    { 13, 5 },
    { 14, 5 },
    { 15, 5 },
    { 15, 6 },
    { 16, 6 },
    { 16, 7 },
    { 16, 8 }
}
```

CAN bittiming table for dynamically calculating the bittiming settings.

According to the CAN protocol 1 bit-time can be made up of between 8..25 time quanta (TQ). The total TQ in a bit is SYNC + TSEG1 + TSEG2 with SYNC always being 1. The sample point is (SYNC + TSEG1) / (SYNC + TSEG1 + SEG2) * 100%. This array contains possible and valid time quanta configurations with a sample point between 68..78%.

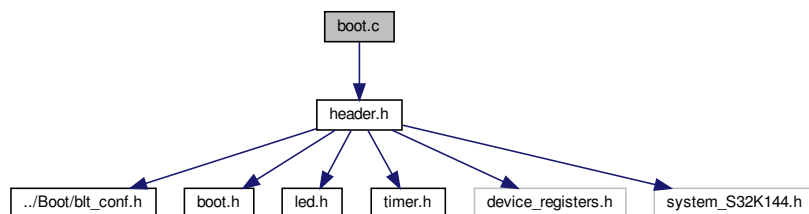
Referenced by CanGetSpeedConfig().

7.198 boot.c File Reference

Demo program bootloader interface source file.

```
#include "header.h"
```

Include dependency graph for Demo/ARMCM4_S32K14_S32K144EVB_GCC/Prog/boot.c:



Data Structures

- struct [tCanBusTiming](#)

Structure type for grouping CAN bus timing related information.

Macros

- `#define RS232_CTO_RX_PACKET_TIMEOUT_MS` (100u)
Timeout time for the reception of a CTO packet. The timer is started upon reception of the first packet byte.
- `#define LPUARTx` (LPUART1)
Set the peripheral LPUART base pointer.
- `#define PCC_LPUARTx_INDEX` (PCC_LPUART1_INDEX)
Set the PCC index offset for LPUART.
- `#define CAN_INIT_TIMEOUT_MS` (250U)
Timeout for entering/leaving CAN initialization mode in milliseconds.
- `#define CANx` (CAN0)
Set the peripheral CAN0 base pointer.
- `#define PCC_FlexCANx_INDEX` (PCC_FlexCAN0_INDEX)
Set the PCC index offset for CAN0.
- `#define CANx_MAX_MB_NUM` (FEATURE_CAN0_MAX_MB_NUM)
Set the number of message boxes supported by CAN0.
- `#define CAN_RX_MSGBOX_NUM` (9U)
The mailbox used for receiving the XCP command message.

Functions

- static void `BootComRs232Init` (void)
Initializes the UART communication interface.
- static void `BootComRs232CheckActivationRequest` (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- static void `BootComCanInit` (void)
Initializes the CAN communication interface.
- static void `BootComCanCheckActivationRequest` (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void `BootComInit` (void)
Initializes the communication interface.
- void `BootComCheckActivationRequest` (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void `BootActivate` (void)
Bootloader activation function.
- static unsigned char `Rs232ReceiveByte` (unsigned char *data)
Receives a communication interface byte if one is present.
- static unsigned char `CanGetSpeedConfig` (unsigned short baud, unsigned short *prescaler, tCanBusTiming *busTimingCfg)
Search algorithm to match the desired baudrate to a possible bus timing configuration.
- static void `CanFreezeModeEnter` (void)
Places the CAN controller in freeze mode. Note that the CAN controller can only be placed in freeze mode, if it is actually enabled.
- static void `CanFreezeModeExit` (void)
Leaves the CAN controller's freeze mode. Note that this operation can only be done, if it is actually enabled.
- static void `CanDisabledModeEnter` (void)
Places the CAN controller in disabled mode.
- static void `CanDisabledModeExit` (void)
Places the CAN controller in enabled mode.

Variables

- static const `tCanBusTiming canTiming []`
CAN bit timing table for dynamically calculating the bittiming settings.
- static volatile unsigned long `dummyTimerVal`
Dummy variable to store the CAN controller's free running timer value in. This is needed at the end of a CAN message reception to unlock the mailbox again. If this variable is declared locally within the function, it generates an unwanted compiler warning about assigning a value and not using it. For this reason this dummy variable is declare here as a module global.

7.198.1 Detailed Description

Demo program bootloader interface source file.

7.198.2 Function Documentation

7.198.2.1 BootActivate()

```
void BootActivate (
    void )
```

Bootloader activation function.

Returns

none.

Referenced by `BootComCanCheckActivationRequest()`, and `BootComRs232CheckActivationRequest()`.

7.198.2.2 BootComCanCheckActivationRequest()

```
static void BootComCanCheckActivationRequest (
    void ) [static]
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

Referenced by `BootComCheckActivationRequest()`.

7.198.2.3 BootComCanInit()

```
static void BootComCanInit (
    void ) [static]
```

Initializes the CAN communication interface.

Returns

none.

Referenced by BootComInit().

7.198.2.4 BootComCheckActivationRequest()

```
void BootComCheckActivationRequest (
    void )
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.198.2.5 BootComInit()

```
void BootComInit (
    void )
```

Initializes the communication interface.

Returns

none.

7.198.2.6 BootComRs232CheckActivationRequest()

```
static void BootComRs232CheckActivationRequest (
    void ) [static]
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

Referenced by BootComCheckActivationRequest().

7.198.2.7 BootComRs232Init()

```
static void BootComRs232Init (  
    void ) [static]
```

Initializes the UART communication interface.

Returns

none.

Referenced by BootComInit().

7.198.2.8 CanDisabledModeEnter()

```
static void CanDisabledModeEnter (  
    void ) [static]
```

Places the CAN controller in disabled mode.

Returns

none.

Referenced by BootComCanInit().

7.198.2.9 CanDisabledModeExit()

```
static void CanDisabledModeExit (  
    void ) [static]
```

Places the CAN controller in enabled mode.

Returns

none.

Referenced by BootComCanInit().

7.198.2.10 CanFreezeModeEnter()

```
static void CanFreezeModeEnter (
    void ) [static]
```

Places the CAN controller in freeze mode. Note that the CAN controller can only be placed in freeze mode, if it is actually enabled.

Returns

none.

Referenced by BootComCanInit().

7.198.2.11 CanFreezeModeExit()

```
static void CanFreezeModeExit (
    void ) [static]
```

Leaves the CAN controller's freeze mode. Note that this operation can only be done, if it is actually enabled.

Returns

none.

Referenced by BootComCanInit().

7.198.2.12 CanGetSpeedConfig()

```
static unsigned char CanGetSpeedConfig (
    unsigned short baud,
    unsigned short * prescaler,
    tCanBusTiming * busTimingCfg ) [static]
```

Search algorithm to match the desired baudrate to a possible bus timing configuration.

Parameters

<i>baud</i>	The desired baudrate in kbps. Valid values are 10..1000.
<i>prescaler</i>	Pointer to where the value for the prescaler will be stored.
<i>busTimingCfg</i>	Pointer to where the bus timing values will be stored.

Returns

1 if the CAN bustiming register values were found, 0 otherwise.

Referenced by `BootComCanInit()`.

7.198.2.13 `Rs232ReceiveByte()`

```
static unsigned char Rs232ReceiveByte (
    unsigned char * data ) [static]
```

Receives a communication interface byte if one is present.

Parameters

<i>data</i>	Pointer to byte where the data is to be stored.
-------------	---

Returns

1 if a byte was received, 0 otherwise.

Referenced by `BootComRs232CheckActivationRequest()`.

7.198.3 Variable Documentation

7.198.3.1 `canTiming`

```
const tCanBusTiming canTiming[] [static]
```

Initial value:

```
=
{
    { 8U, 3U, 2U, 2U },
    { 9U, 3U, 3U, 2U },
    { 10U, 3U, 3U, 3U },
    { 11U, 4U, 3U, 3U },
    { 12U, 4U, 4U, 3U },
    { 13U, 5U, 4U, 3U },
    { 14U, 5U, 4U, 4U },
    { 15U, 6U, 4U, 4U },
    { 16U, 6U, 5U, 4U },
    { 17U, 7U, 5U, 4U },
    { 18U, 7U, 5U, 5U },
    { 19U, 8U, 5U, 5U },
    { 20U, 8U, 6U, 5U },
    { 21U, 8U, 7U, 5U },
    { 22U, 8U, 7U, 6U },
    { 23U, 8U, 8U, 6U },
    { 24U, 8U, 8U, 7U },
    { 25U, 8U, 8U, 8U }
}
```

CAN bit timing table for dynamically calculating the bittiming settings.

According to the CAN protocol 1 bit-time can be made up of between 8..25 time quanta (TQ). The total TQ in a bit is SYNC + TSEG1 + TSEG2 with SYNC always being 1. The sample point is (SYNC + TSEG1) / (SYNC + TSEG1 + TSEG2)

- 100%. This array contains possible and valid time quanta configurations with a sample point between 68..78%. A visual representation of the TQ in a bit is: | SYNCSEG | TIME1SEG | TIME2SEG | Or with an alternative representation: | SYNCSEG | PROPSEG | PHASE1SEG | PHASE2SEG | With the alternative representation TIME1SEG = PROPSEG + PHASE1SEG.

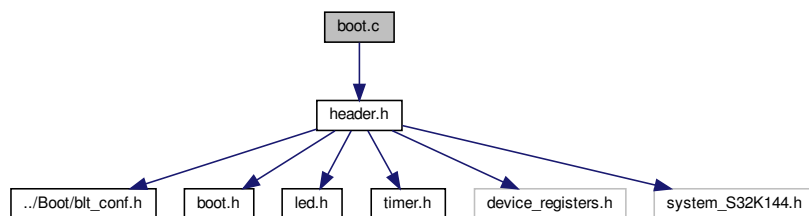
Referenced by CanGetSpeedConfig().

7.199 boot.c File Reference

Demo program bootloader interface source file.

```
#include "header.h"
```

Include dependency graph for Demo/ARMCM4_S32K14_S32K144EVB_IAR/Prog/boot.c:



Data Structures

- struct [tCanBusTiming](#)
Structure type for grouping CAN bus timing related information.

Macros

- [#define RS232_CTO_RX_PACKET_TIMEOUT_MS](#) (100u)
Timeout time for the reception of a CTO packet. The timer is started upon reception of the first packet byte.
- [#define LPUARTx](#) (LPUART1)
Set the peripheral LPUART base pointer.
- [#define PCC_LPUARTx_INDEX](#) (PCC_LPUART1_INDEX)
Set the PCC index offset for LPUART.
- [#define CAN_INIT_TIMEOUT_MS](#) (250U)
Timeout for entering/leaving CAN initialization mode in milliseconds.
- [#define CANx](#) (CAN0)
Set the peripheral CAN0 base pointer.
- [#define PCC_FlexCANx_INDEX](#) (PCC_FlexCAN0_INDEX)
Set the PCC index offset for CAN0.
- [#define CANx_MAX_MB_NUM](#) (FEATURE_CAN0_MAX_MB_NUM)
Set the number of message boxes supported by CAN0.
- [#define CAN_RX_MSGBOX_NUM](#) (9U)
The mailbox used for receiving the XCP command message.

Functions

- static void [BootComRs232Init](#) (void)
Initializes the UART communication interface.
- static void [BootComRs232CheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- static void [BootComCanInit](#) (void)
Initializes the CAN communication interface.
- static void [BootComCanCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootComInit](#) (void)
Initializes the communication interface.
- void [BootComCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootActivate](#) (void)
Bootloader activation function.
- static unsigned char [Rs232ReceiveByte](#) (unsigned char *data)
Receives a communication interface byte if one is present.
- static unsigned char [CanGetSpeedConfig](#) (unsigned short baud, unsigned short *prescaler, [tCanBusTiming](#) *busTimingCfg)
Search algorithm to match the desired baudrate to a possible bus timing configuration.
- static void [CanFreezeModeEnter](#) (void)
Places the CAN controller in freeze mode. Note that the CAN controller can only be placed in freeze mode, if it is actually enabled.
- static void [CanFreezeModeExit](#) (void)
Leaves the CAN controller's freeze mode. Note that this operation can only be done, if it is actually enabled.
- static void [CanDisabledModeEnter](#) (void)
Places the CAN controller in disabled mode.
- static void [CanDisabledModeExit](#) (void)
Places the CAN controller in enabled mode.

Variables

- static const [tCanBusTiming](#) [canTiming](#) []
CAN bit timing table for dynamically calculating the bittiming settings.
- static volatile unsigned long [dummyTimerVal](#)
Dummy variable to store the CAN controller's free running timer value in. This is needed at the end of a CAN message reception to unlock the mailbox again. If this variable is declared locally within the function, it generates an unwanted compiler warning about assigning a value and not using it. For this reason this dummy variable is declare here as a module global.

7.199.1 Detailed Description

Demo program bootloader interface source file.

7.199.2 Function Documentation

7.199.2.1 BootActivate()

```
void BootActivate (
    void )
```

Bootloader activation function.

Returns

none.

Referenced by `BootComCanCheckActivationRequest()`, and `BootComRs232CheckActivationRequest()`.

7.199.2.2 BootComCanCheckActivationRequest()

```
static void BootComCanCheckActivationRequest (
    void ) [static]
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

Referenced by `BootComCheckActivationRequest()`.

7.199.2.3 BootComCanInit()

```
static void BootComCanInit (
    void ) [static]
```

Initializes the CAN communication interface.

Returns

none.

Referenced by `BootComInit()`.

7.199.2.4 BootComCheckActivationRequest()

```
void BootComCheckActivationRequest (
    void )
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.199.2.5 BootComInit()

```
void BootComInit (
    void )
```

Initializes the communication interface.

Returns

none.

7.199.2.6 BootComRs232CheckActivationRequest()

```
static void BootComRs232CheckActivationRequest (
    void ) [static]
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

Referenced by BootComCheckActivationRequest().

7.199.2.7 BootComRs232Init()

```
static void BootComRs232Init (
    void ) [static]
```

Initializes the UART communication interface.

Returns

none.

Referenced by BootComInit().

7.199.2.8 CanDisabledModeEnter()

```
static void CanDisabledModeEnter (  
    void ) [static]
```

Places the CAN controller in disabled mode.

Returns

none.

Referenced by BootComCanInit().

7.199.2.9 CanDisabledModeExit()

```
static void CanDisabledModeExit (  
    void ) [static]
```

Places the CAN controller in enabled mode.

Returns

none.

Referenced by BootComCanInit().

7.199.2.10 CanFreezeModeEnter()

```
static void CanFreezeModeEnter (  
    void ) [static]
```

Places the CAN controller in freeze mode. Note that the CAN controller can only be placed in freeze mode, if it is actually enabled.

Returns

none.

Referenced by BootComCanInit().

7.199.2.11 CanFreezeModeExit()

```
static void CanFreezeModeExit (
    void ) [static]
```

Leaves the CAN controller's freeze mode. Note that this operation can only be done, if it is actually enabled.

Returns

none.

Referenced by BootComCanInit().

7.199.2.12 CanGetSpeedConfig()

```
static unsigned char CanGetSpeedConfig (
    unsigned short baud,
    unsigned short * prescaler,
    tCanBusTiming * busTimingCfg ) [static]
```

Search algorithm to match the desired baudrate to a possible bus timing configuration.

Parameters

<i>baud</i>	The desired baudrate in kbps. Valid values are 10..1000.
<i>prescaler</i>	Pointer to where the value for the prescaler will be stored.
<i>busTimingCfg</i>	Pointer to where the bus timing values will be stored.

Returns

1 if the CAN bustiming register values were found, 0 otherwise.

Referenced by BootComCanInit().

7.199.2.13 Rs232ReceiveByte()

```
static unsigned char Rs232ReceiveByte (
    unsigned char * data ) [static]
```

Receives a communication interface byte if one is present.

Parameters

<i>data</i>	Pointer to byte where the data is to be stored.
-------------	---

Returns

1 if a byte was received, 0 otherwise.

Referenced by BootComRs232CheckActivationRequest().

7.199.3 Variable Documentation**7.199.3.1 canTiming**

```
const tCanBusTiming canTiming[] [static]
```

Initial value:

```
=
{
    { 8U, 3U, 2U, 2U },
    { 9U, 3U, 3U, 2U },
    { 10U, 3U, 3U, 3U },
    { 11U, 4U, 3U, 3U },
    { 12U, 4U, 4U, 3U },
    { 13U, 5U, 4U, 3U },
    { 14U, 5U, 4U, 4U },
    { 15U, 6U, 4U, 4U },
    { 16U, 6U, 5U, 4U },
    { 17U, 7U, 5U, 4U },
    { 18U, 7U, 5U, 5U },
    { 19U, 8U, 5U, 5U },
    { 20U, 8U, 6U, 5U },
    { 21U, 8U, 7U, 5U },
    { 22U, 8U, 7U, 6U },
    { 23U, 8U, 8U, 6U },
    { 24U, 8U, 8U, 7U },
    { 25U, 8U, 8U, 8U }
}
```

CAN bit timing table for dynamically calculating the bittiming settings.

According to the CAN protocol 1 bit-time can be made up of between 8..25 time quanta (TQ). The total TQ in a bit is SYNC + TSEG1 + TSEG2 with SYNC always being 1. The sample point is (SYNC + TSEG1) / (SYNC + TSEG1 + TSEG2)

- 100%. This array contains possible and valid time quanta configurations with a sample point between 68..78%. A visual representation of the TQ in a bit is: | SYNCSEG | TIME1SEG | TIME2SEG | Or with an alternative representation: | SYNCSEG | PROPSEG | PHASE1SEG | PHASE2SEG | With the alternative representation TIME1SEG = PROPSEG + PHASE1SEG.

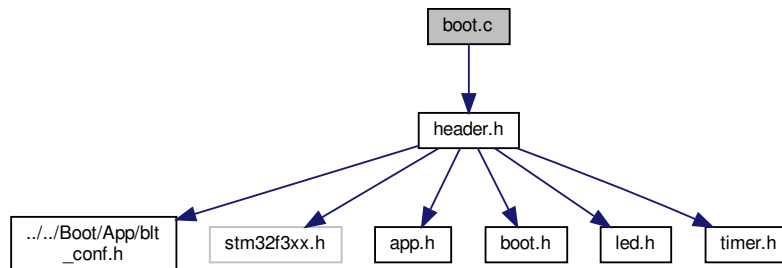
Referenced by CanGetSpeedConfig().

7.200 boot.c File Reference

Demo program bootloader interface source file.

```
#include "header.h"
```

Include dependency graph for Demo/ARMCM4_STM32F3_Nucleo_F303K8_CubeIDE/Prog/App/boot.c:



Data Structures

- struct [tCanBusTiming](#)
Structure type for grouping CAN bus timing related information.

Macros

- #define [RS232_CTO_RX_PACKET_TIMEOUT_MS](#) (100u)
Timeout time for the reception of a CTO packet. The timer is started upon reception of the first packet byte.

Functions

- static void [BootComRs232Init](#) (void)
Initializes the UART communication interface.
- static void [BootComRs232CheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- static void [BootComCanInit](#) (void)
Initializes the CAN communication interface.
- static void [BootComCanCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootComInit](#) (void)
Initializes the communication interface.
- void [BootComCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootActivate](#) (void)
Bootloader activation function.
- static unsigned char [Rs232ReceiveByte](#) (unsigned char *data)
Receives a communication interface byte if one is present.
- static unsigned char [CanGetSpeedConfig](#) (unsigned short baud, unsigned short *prescaler, unsigned char *tseg1, unsigned char *tseg2)
Search algorithm to match the desired baudrate to a possible bus timing configuration.

Variables

- static UART_HandleTypeDef [rs232Handle](#)
UART handle to be used in API calls.
- static const [tCanBusTiming](#) [canTiming](#) []
CAN bittiming table for dynamically calculating the bittiming settings.
- static CAN_HandleTypeDef [canHandle](#)
CAN handle to be used in API calls.

7.200.1 Detailed Description

Demo program bootloader interface source file.

7.200.2 Function Documentation

7.200.2.1 BootActivate()

```
void BootActivate (  
    void )
```

Bootloader activation function.

Returns

none.

Referenced by [BootComCanCheckActivationRequest\(\)](#), and [BootComRs232CheckActivationRequest\(\)](#).

7.200.2.2 BootComCanCheckActivationRequest()

```
static void BootComCanCheckActivationRequest (  
    void ) [static]
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

Referenced by [BootComCheckActivationRequest\(\)](#).

7.200.2.3 BootComCanInit()

```
static void BootComCanInit (  
    void ) [static]
```

Initializes the CAN communication interface.

Returns

none.

Referenced by BootComInit().

7.200.2.4 BootComCheckActivationRequest()

```
void BootComCheckActivationRequest (  
    void )
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.200.2.5 BootComInit()

```
void BootComInit (  
    void )
```

Initializes the communication interface.

Returns

none.

7.200.2.6 BootComRs232CheckActivationRequest()

```
static void BootComRs232CheckActivationRequest (  
    void ) [static]
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

Referenced by BootComCheckActivationRequest().

7.200.2.7 BootComRs232Init()

```
static void BootComRs232Init (  
    void ) [static]
```

Initializes the UART communication interface.

Returns

none.

Referenced by BootComInit().

7.200.2.8 CanGetSpeedConfig()

```
static unsigned char CanGetSpeedConfig (  
    unsigned short baud,  
    unsigned short * prescaler,  
    unsigned char * tseg1,  
    unsigned char * tseg2 ) [static]
```

Search algorithm to match the desired baudrate to a possible bus timing configuration.

Parameters

<i>baud</i>	The desired baudrate in kbps. Valid values are 10..1000.
<i>prescaler</i>	Pointer to where the value for the prescaler will be stored.
<i>tseg1</i>	Pointer to where the value for TSEG2 will be stored.
<i>tseg2</i>	Pointer to where the value for TSEG2 will be stored.

Returns

1 if the CAN bustiming register values were found, 0 otherwise.

Referenced by BootComCanInit().

7.200.2.9 Rs232ReceiveByte()

```
static unsigned char Rs232ReceiveByte (  
    unsigned char * data ) [static]
```

Receives a communication interface byte if one is present.

Parameters

<i>data</i>	Pointer to byte where the data is to be stored.
-------------	---

Returns

1 if a byte was received, 0 otherwise.

Referenced by `BootComRs232CheckActivationRequest()`.

7.200.3 Variable Documentation**7.200.3.1 canTiming**

```
const tCanBusTiming canTiming[] [static]
```

Initial value:

```
=
{
    { 5, 2 },
    { 6, 2 },
    { 6, 3 },
    { 7, 3 },
    { 8, 3 },
    { 9, 3 },
    { 9, 4 },
    { 10, 4 },
    { 11, 4 },
    { 12, 4 },
    { 12, 5 },
    { 13, 5 },
    { 14, 5 },
    { 15, 5 },
    { 15, 6 },
    { 16, 6 },
    { 16, 7 },
    { 16, 8 }
}
```

CAN bittiming table for dynamically calculating the bittiming settings.

According to the CAN protocol 1 bit-time can be made up of between 8..25 time quanta (TQ). The total TQ in a bit is SYNC + TSEG1 + TSEG2 with SYNC always being 1. The sample point is (SYNC + TSEG1) / (SYNC + TSEG1 + SEG2) * 100%. This array contains possible and valid time quanta configurations with a sample point between 68..78%.

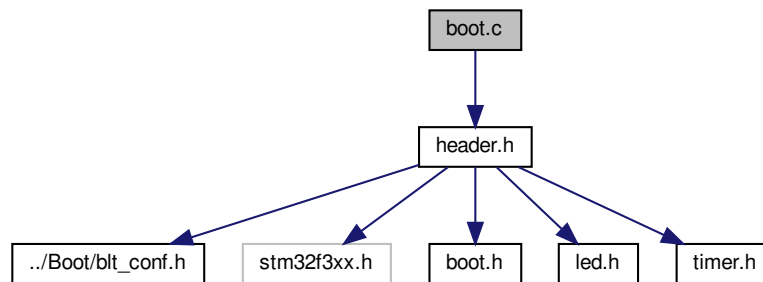
Referenced by `CanGetSpeedConfig()`.

7.201 boot.c File Reference

Demo program bootloader interface source file.

```
#include "header.h"
```

Include dependency graph for Demo/ARMCM4_STM32F3_Nucleo_F303K8_GCC/Prog/boot.c:



Data Structures

- struct [tCanBusTiming](#)
Structure type for grouping CAN bus timing related information.

Macros

- `#define RS232_CTO_RX_PACKET_TIMEOUT_MS (100u)`
Timeout time for the reception of a CTO packet. The timer is started upon reception of the first packet byte.

Functions

- static void [BootComRs232Init](#) (void)
Initializes the UART communication interface.
- static void [BootComRs232CheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- static void [BootComCanInit](#) (void)
Initializes the CAN communication interface.
- static void [BootComCanCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootComInit](#) (void)
Initializes the communication interface.
- void [BootComCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootActivate](#) (void)
Bootloader activation function.
- static unsigned char [Rs232ReceiveByte](#) (unsigned char *data)
Receives a communication interface byte if one is present.
- static unsigned char [CanGetSpeedConfig](#) (unsigned short baud, unsigned short *prescaler, unsigned char *tseg1, unsigned char *tseg2)
Search algorithm to match the desired baudrate to a possible bus timing configuration.

Variables

- static UART_HandleTypeDef [rs232Handle](#)
UART handle to be used in API calls.
- static const [tCanBusTiming](#) [canTiming](#) []
CAN bittiming table for dynamically calculating the bittiming settings.
- static CAN_HandleTypeDef [canHandle](#)
CAN handle to be used in API calls.

7.201.1 Detailed Description

Demo program bootloader interface source file.

7.201.2 Function Documentation

7.201.2.1 BootActivate()

```
void BootActivate (  
    void )
```

Bootloader activation function.

Returns

none.

Referenced by [BootComCanCheckActivationRequest\(\)](#), and [BootComRs232CheckActivationRequest\(\)](#).

7.201.2.2 BootComCanCheckActivationRequest()

```
static void BootComCanCheckActivationRequest (  
    void ) [static]
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

Referenced by [BootComCheckActivationRequest\(\)](#).

7.201.2.3 BootComCanInit()

```
static void BootComCanInit (  
    void ) [static]
```

Initializes the CAN communication interface.

Returns

none.

Referenced by BootComInit().

7.201.2.4 BootComCheckActivationRequest()

```
void BootComCheckActivationRequest (  
    void )
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.201.2.5 BootComInit()

```
void BootComInit (  
    void )
```

Initializes the communication interface.

Returns

none.

7.201.2.6 BootComRs232CheckActivationRequest()

```
static void BootComRs232CheckActivationRequest (  
    void ) [static]
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

Referenced by BootComCheckActivationRequest().

7.201.2.7 BootComRs232Init()

```
static void BootComRs232Init (  
    void ) [static]
```

Initializes the UART communication interface.

Returns

none.

Referenced by BootComInit().

7.201.2.8 CanGetSpeedConfig()

```
static unsigned char CanGetSpeedConfig (  
    unsigned short baud,  
    unsigned short * prescaler,  
    unsigned char * tseg1,  
    unsigned char * tseg2 ) [static]
```

Search algorithm to match the desired baudrate to a possible bus timing configuration.

Parameters

<i>baud</i>	The desired baudrate in kbps. Valid values are 10..1000.
<i>prescaler</i>	Pointer to where the value for the prescaler will be stored.
<i>tseg1</i>	Pointer to where the value for TSEG2 will be stored.
<i>tseg2</i>	Pointer to where the value for TSEG2 will be stored.

Returns

1 if the CAN bustiming register values were found, 0 otherwise.

Referenced by BootComCanInit().

7.201.2.9 Rs232ReceiveByte()

```
static unsigned char Rs232ReceiveByte (  
    unsigned char * data ) [static]
```

Receives a communication interface byte if one is present.

Parameters

<i>data</i>	Pointer to byte where the data is to be stored.
-------------	---

Returns

1 if a byte was received, 0 otherwise.

Referenced by `BootComRs232CheckActivationRequest()`.

7.201.3 Variable Documentation**7.201.3.1 canTiming**

```
const tCanBusTiming canTiming[] [static]
```

Initial value:

```
=
{
    { 5, 2 },
    { 6, 2 },
    { 6, 3 },
    { 7, 3 },
    { 8, 3 },
    { 9, 3 },
    { 9, 4 },
    { 10, 4 },
    { 11, 4 },
    { 12, 4 },
    { 12, 5 },
    { 13, 5 },
    { 14, 5 },
    { 15, 5 },
    { 15, 6 },
    { 16, 6 },
    { 16, 7 },
    { 16, 8 }
}
```

CAN bittiming table for dynamically calculating the bittiming settings.

According to the CAN protocol 1 bit-time can be made up of between 8..25 time quanta (TQ). The total TQ in a bit is SYNC + TSEG1 + TSEG2 with SYNC always being 1. The sample point is (SYNC + TSEG1) / (SYNC + TSEG1 + SEG2) * 100%. This array contains possible and valid time quanta configurations with a sample point between 68..78%.

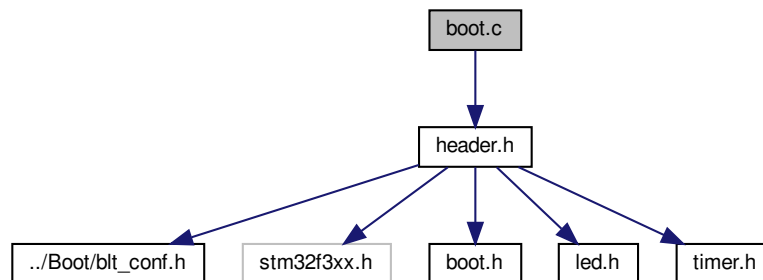
Referenced by `CanGetSpeedConfig()`.

7.202 boot.c File Reference

Demo program bootloader interface source file.

```
#include "header.h"
```

Include dependency graph for Demo/ARMCM4_STM32F3_Nucleo_F303K8_IAR/Prog/boot.c:



Data Structures

- struct [tCanBusTiming](#)
Structure type for grouping CAN bus timing related information.

Macros

- `#define RS232_CTO_RX_PACKET_TIMEOUT_MS (100u)`
Timeout time for the reception of a CTO packet. The timer is started upon reception of the first packet byte.

Functions

- static void [BootComRs232Init](#) (void)
Initializes the UART communication interface.
- static void [BootComRs232CheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- static void [BootComCanInit](#) (void)
Initializes the CAN communication interface.
- static void [BootComCanCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootComInit](#) (void)
Initializes the communication interface.
- void [BootComCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootActivate](#) (void)
Bootloader activation function.
- static unsigned char [Rs232ReceiveByte](#) (unsigned char *data)
Receives a communication interface byte if one is present.
- static unsigned char [CanGetSpeedConfig](#) (unsigned short baud, unsigned short *prescaler, unsigned char *tseg1, unsigned char *tseg2)
Search algorithm to match the desired baudrate to a possible bus timing configuration.

Variables

- static UART_HandleTypeDef [rs232Handle](#)
UART handle to be used in API calls.
- static const [tCanBusTiming](#) [canTiming](#) []
CAN bittiming table for dynamically calculating the bittiming settings.
- static CAN_HandleTypeDef [canHandle](#)
CAN handle to be used in API calls.

7.202.1 Detailed Description

Demo program bootloader interface source file.

7.202.2 Function Documentation

7.202.2.1 BootActivate()

```
void BootActivate (  
    void )
```

Bootloader activation function.

Returns

none.

Referenced by [BootComCanCheckActivationRequest\(\)](#), and [BootComRs232CheckActivationRequest\(\)](#).

7.202.2.2 BootComCanCheckActivationRequest()

```
static void BootComCanCheckActivationRequest (  
    void ) [static]
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

Referenced by [BootComCheckActivationRequest\(\)](#).

7.202.2.3 BootComCanInit()

```
static void BootComCanInit (  
    void ) [static]
```

Initializes the CAN communication interface.

Returns

none.

Referenced by BootComInit().

7.202.2.4 BootComCheckActivationRequest()

```
void BootComCheckActivationRequest (  
    void )
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.202.2.5 BootComInit()

```
void BootComInit (  
    void )
```

Initializes the communication interface.

Returns

none.

7.202.2.6 BootComRs232CheckActivationRequest()

```
static void BootComRs232CheckActivationRequest (  
    void ) [static]
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

Referenced by BootComCheckActivationRequest().

7.202.2.7 BootComRs232Init()

```
static void BootComRs232Init (  
    void ) [static]
```

Initializes the UART communication interface.

Returns

none.

Referenced by BootComInit().

7.202.2.8 CanGetSpeedConfig()

```
static unsigned char CanGetSpeedConfig (  
    unsigned short baud,  
    unsigned short * prescaler,  
    unsigned char * tseg1,  
    unsigned char * tseg2 ) [static]
```

Search algorithm to match the desired baudrate to a possible bus timing configuration.

Parameters

<i>baud</i>	The desired baudrate in kbps. Valid values are 10..1000.
<i>prescaler</i>	Pointer to where the value for the prescaler will be stored.
<i>tseg1</i>	Pointer to where the value for TSEG2 will be stored.
<i>tseg2</i>	Pointer to where the value for TSEG2 will be stored.

Returns

1 if the CAN bustiming register values were found, 0 otherwise.

Referenced by BootComCanInit().

7.202.2.9 Rs232ReceiveByte()

```
static unsigned char Rs232ReceiveByte (  
    unsigned char * data ) [static]
```

Receives a communication interface byte if one is present.

Parameters

<i>data</i>	Pointer to byte where the data is to be stored.
-------------	---

Returns

1 if a byte was received, 0 otherwise.

Referenced by `BootComRs232CheckActivationRequest()`.

7.202.3 Variable Documentation**7.202.3.1 canTiming**

```
const tCanBusTiming canTiming[] [static]
```

Initial value:

```
=  
{  
    { 5, 2 },  
    { 6, 2 },  
    { 6, 3 },  
    { 7, 3 },  
    { 8, 3 },  
    { 9, 3 },  
    { 9, 4 },  
    { 10, 4 },  
    { 11, 4 },  
    { 12, 4 },  
    { 12, 5 },  
    { 13, 5 },  
    { 14, 5 },  
    { 15, 5 },  
    { 15, 6 },  
    { 16, 6 },  
    { 16, 7 },  
    { 16, 8 }  
}
```

CAN bittiming table for dynamically calculating the bittiming settings.

According to the CAN protocol 1 bit-time can be made up of between 8..25 time quanta (TQ). The total TQ in a bit is SYNC + TSEG1 + TSEG2 with SYNC always being 1. The sample point is (SYNC + TSEG1) / (SYNC + TSEG1 + SEG2) * 100%. This array contains possible and valid time quanta configurations with a sample point between 68..78%.

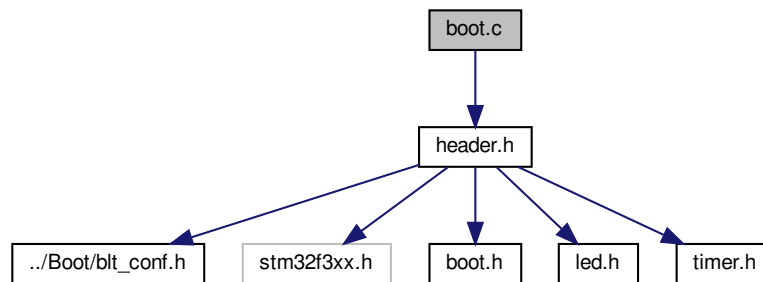
Referenced by `CanGetSpeedConfig()`.

7.203 boot.c File Reference

Demo program bootloader interface source file.

```
#include "header.h"
```

Include dependency graph for Demo/ARMCM4_STM32F3_Nucleo_F303K8_Keil/Prog/boot.c:



Data Structures

- struct [tCanBusTiming](#)
Structure type for grouping CAN bus timing related information.

Macros

- `#define RS232_CTO_RX_PACKET_TIMEOUT_MS (100u)`
Timeout time for the reception of a CTO packet. The timer is started upon reception of the first packet byte.

Functions

- static void [BootComRs232Init](#) (void)
Initializes the UART communication interface.
- static void [BootComRs232CheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- static void [BootComCanInit](#) (void)
Initializes the CAN communication interface.
- static void [BootComCanCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootComInit](#) (void)
Initializes the communication interface.
- void [BootComCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootActivate](#) (void)
Bootloader activation function.
- static unsigned char [Rs232ReceiveByte](#) (unsigned char *data)
Receives a communication interface byte if one is present.
- static unsigned char [CanGetSpeedConfig](#) (unsigned short baud, unsigned short *prescaler, unsigned char *tseg1, unsigned char *tseg2)
Search algorithm to match the desired baudrate to a possible bus timing configuration.

Variables

- static UART_HandleTypeDef [rs232Handle](#)
UART handle to be used in API calls.
- static const [tCanBusTiming](#) [canTiming](#) []
CAN bittiming table for dynamically calculating the bittiming settings.
- static CAN_HandleTypeDef [canHandle](#)
CAN handle to be used in API calls.

7.203.1 Detailed Description

Demo program bootloader interface source file.

7.203.2 Function Documentation

7.203.2.1 BootActivate()

```
void BootActivate (  
    void )
```

Bootloader activation function.

Returns

none.

Referenced by [BootComCanCheckActivationRequest\(\)](#), and [BootComRs232CheckActivationRequest\(\)](#).

7.203.2.2 BootComCanCheckActivationRequest()

```
static void BootComCanCheckActivationRequest (  
    void ) [static]
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

Referenced by [BootComCheckActivationRequest\(\)](#).

7.203.2.3 BootComCanInit()

```
static void BootComCanInit (  
    void ) [static]
```

Initializes the CAN communication interface.

Returns

none.

Referenced by BootComInit().

7.203.2.4 BootComCheckActivationRequest()

```
void BootComCheckActivationRequest (  
    void )
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.203.2.5 BootComInit()

```
void BootComInit (  
    void )
```

Initializes the communication interface.

Returns

none.

7.203.2.6 BootComRs232CheckActivationRequest()

```
static void BootComRs232CheckActivationRequest (  
    void ) [static]
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

Referenced by BootComCheckActivationRequest().

7.203.2.7 BootComRs232Init()

```
static void BootComRs232Init (  
    void ) [static]
```

Initializes the UART communication interface.

Returns

none.

Referenced by BootComInit().

7.203.2.8 CanGetSpeedConfig()

```
static unsigned char CanGetSpeedConfig (  
    unsigned short baud,  
    unsigned short * prescaler,  
    unsigned char * tseg1,  
    unsigned char * tseg2 ) [static]
```

Search algorithm to match the desired baudrate to a possible bus timing configuration.

Parameters

<i>baud</i>	The desired baudrate in kbps. Valid values are 10..1000.
<i>prescaler</i>	Pointer to where the value for the prescaler will be stored.
<i>tseg1</i>	Pointer to where the value for TSEG2 will be stored.
<i>tseg2</i>	Pointer to where the value for TSEG2 will be stored.

Returns

1 if the CAN bustiming register values were found, 0 otherwise.

Referenced by BootComCanInit().

7.203.2.9 Rs232ReceiveByte()

```
static unsigned char Rs232ReceiveByte (  
    unsigned char * data ) [static]
```

Receives a communication interface byte if one is present.

Parameters

<i>data</i>	Pointer to byte where the data is to be stored.
-------------	---

Returns

1 if a byte was received, 0 otherwise.

Referenced by `BootComRs232CheckActivationRequest()`.

7.203.3 Variable Documentation**7.203.3.1 canTiming**

```
const tCanBusTiming canTiming[] [static]
```

Initial value:

```
=
{
    { 5, 2 },
    { 6, 2 },
    { 6, 3 },
    { 7, 3 },
    { 8, 3 },
    { 9, 3 },
    { 9, 4 },
    { 10, 4 },
    { 11, 4 },
    { 12, 4 },
    { 12, 5 },
    { 13, 5 },
    { 14, 5 },
    { 15, 5 },
    { 15, 6 },
    { 16, 6 },
    { 16, 7 },
    { 16, 8 }
}
```

CAN bittiming table for dynamically calculating the bittiming settings.

According to the CAN protocol 1 bit-time can be made up of between 8..25 time quanta (TQ). The total TQ in a bit is SYNC + TSEG1 + TSEG2 with SYNC always being 1. The sample point is (SYNC + TSEG1) / (SYNC + TSEG1 + SEG2) * 100%. This array contains possible and valid time quanta configurations with a sample point between 68..78%.

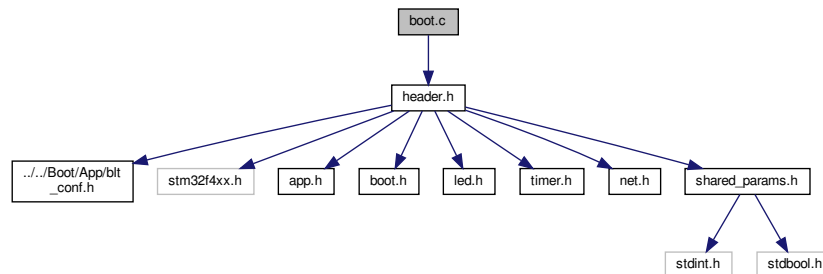
Referenced by `CanGetSpeedConfig()`.

7.204 boot.c File Reference

Demo program bootloader interface source file.

```
#include "header.h"
```

Include dependency graph for Demo/ARMCM4_STM32F4_Nucleo_F429ZI_CubeIDE/Prog/App/boot.c:



Data Structures

- struct [tCanBusTiming](#)
Structure type for grouping CAN bus timing related information.

Macros

- `#define` [RS232_CTO_RX_PACKET_TIMEOUT_MS](#) (100u)
Timeout time for the reception of a CTO packet. The timer is started upon reception of the first packet byte.

Functions

- static void [BootComRs232Init](#) (void)
Initializes the UART communication interface.
- static void [BootComRs232CheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- static void [BootComCanInit](#) (void)
Initializes the CAN communication interface.
- static void [BootComCanCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootComInit](#) (void)
Initializes the communication interface.
- void [BootComCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootActivate](#) (void)
Bootloader activation function.
- static unsigned char [Rs232ReceiveByte](#) (unsigned char *data)
Receives a communication interface byte if one is present.
- static unsigned char [CanGetSpeedConfig](#) (unsigned short baud, unsigned short *prescaler, unsigned char *tseg1, unsigned char *tseg2)
Search algorithm to match the desired baudrate to a possible bus timing configuration.

Variables

- static UART_HandleTypeDef [rs232Handle](#)
UART handle to be used in API calls.
- static const [tCanBusTiming](#) [canTiming](#) []
CAN bittiming table for dynamically calculating the bittiming settings.
- static CAN_HandleTypeDef [canHandle](#)
CAN handle to be used in API calls.

7.204.1 Detailed Description

Demo program bootloader interface source file.

7.204.2 Function Documentation

7.204.2.1 BootActivate()

```
void BootActivate (  
    void )
```

Bootloader activation function.

Returns

none.

Referenced by [BootComCanCheckActivationRequest\(\)](#), and [BootComRs232CheckActivationRequest\(\)](#).

7.204.2.2 BootComCanCheckActivationRequest()

```
static void BootComCanCheckActivationRequest (  
    void ) [static]
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

Referenced by [BootComCheckActivationRequest\(\)](#).

7.204.2.3 BootComCanInit()

```
static void BootComCanInit (  
    void ) [static]
```

Initializes the CAN communication interface.

Returns

none.

Referenced by BootComInit().

7.204.2.4 BootComCheckActivationRequest()

```
void BootComCheckActivationRequest (  
    void )
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.204.2.5 BootComInit()

```
void BootComInit (  
    void )
```

Initializes the communication interface.

Returns

none.

7.204.2.6 BootComRs232CheckActivationRequest()

```
static void BootComRs232CheckActivationRequest (  
    void ) [static]
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

Referenced by BootComCheckActivationRequest().

7.204.2.7 BootComRs232Init()

```
static void BootComRs232Init (  
    void ) [static]
```

Initializes the UART communication interface.

Returns

none.

Referenced by BootComInit().

7.204.2.8 CanGetSpeedConfig()

```
static unsigned char CanGetSpeedConfig (  
    unsigned short baud,  
    unsigned short * prescaler,  
    unsigned char * tseg1,  
    unsigned char * tseg2 ) [static]
```

Search algorithm to match the desired baudrate to a possible bus timing configuration.

Parameters

<i>baud</i>	The desired baudrate in kbps. Valid values are 10..1000.
<i>prescaler</i>	Pointer to where the value for the prescaler will be stored.
<i>tseg1</i>	Pointer to where the value for TSEG2 will be stored.
<i>tseg2</i>	Pointer to where the value for TSEG2 will be stored.

Returns

1 if the CAN bustiming register values were found, 0 otherwise.

Referenced by BootComCanInit().

7.204.2.9 Rs232ReceiveByte()

```
static unsigned char Rs232ReceiveByte (  
    unsigned char * data ) [static]
```

Receives a communication interface byte if one is present.

Parameters

<i>data</i>	Pointer to byte where the data is to be stored.
-------------	---

Returns

1 if a byte was received, 0 otherwise.

Referenced by `BootComRs232CheckActivationRequest()`.

7.204.3 Variable Documentation**7.204.3.1 canTiming**

```
const tCanBusTiming canTiming[] [static]
```

Initial value:

```
=
{
    { 5, 2 },
    { 6, 2 },
    { 6, 3 },
    { 7, 3 },
    { 8, 3 },
    { 9, 3 },
    { 9, 4 },
    { 10, 4 },
    { 11, 4 },
    { 12, 4 },
    { 12, 5 },
    { 13, 5 },
    { 14, 5 },
    { 15, 5 },
    { 15, 6 },
    { 16, 6 },
    { 16, 7 },
    { 16, 8 }
}
```

CAN bittiming table for dynamically calculating the bittiming settings.

According to the CAN protocol 1 bit-time can be made up of between 8..25 time quanta (TQ). The total TQ in a bit is SYNC + TSEG1 + TSEG2 with SYNC always being 1. The sample point is (SYNC + TSEG1) / (SYNC + TSEG1 + SEG2) * 100%. This array contains possible and valid time quanta configurations with a sample point between 68..78%.

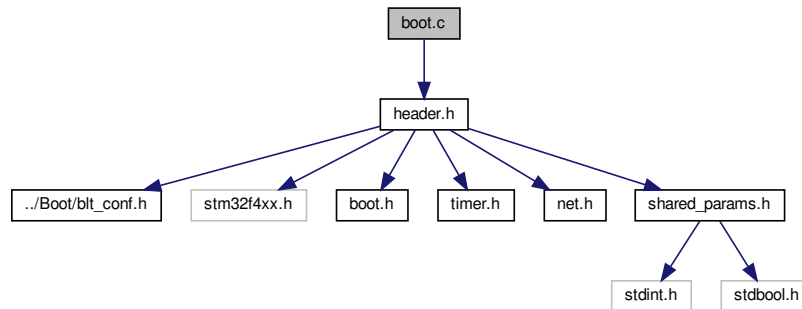
Referenced by `CanGetSpeedConfig()`.

7.205 boot.c File Reference

Demo program bootloader interface source file.

```
#include "header.h"
```

Include dependency graph for Demo/ARMCM4_STM32F4_Nucleo_F429ZI_GCC/Prog/boot.c:



Data Structures

- struct [tCanBusTiming](#)
Structure type for grouping CAN bus timing related information.

Macros

- #define [RS232_CTO_RX_PACKET_TIMEOUT_MS](#) (100u)
Timeout time for the reception of a CTO packet. The timer is started upon reception of the first packet byte.

Functions

- static void [BootComRs232Init](#) (void)
Initializes the UART communication interface.
- static void [BootComRs232CheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- static void [BootComCanInit](#) (void)
Initializes the CAN communication interface.
- static void [BootComCanCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootComInit](#) (void)
Initializes the communication interface.
- void [BootComCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootActivate](#) (void)
Bootloader activation function.
- static unsigned char [Rs232ReceiveByte](#) (unsigned char *data)
Receives a communication interface byte if one is present.
- static unsigned char [CanGetSpeedConfig](#) (unsigned short baud, unsigned short *prescaler, unsigned char *tseg1, unsigned char *tseg2)
Search algorithm to match the desired baudrate to a possible bus timing configuration.

Variables

- static UART_HandleTypeDef [rs232Handle](#)
UART handle to be used in API calls.
- static const [tCanBusTiming](#) [canTiming](#) []
CAN bittiming table for dynamically calculating the bittiming settings.
- static CAN_HandleTypeDef [canHandle](#)
CAN handle to be used in API calls.

7.205.1 Detailed Description

Demo program bootloader interface source file.

7.205.2 Function Documentation

7.205.2.1 BootActivate()

```
void BootActivate (  
    void )
```

Bootloader activation function.

Returns

none.

Referenced by [BootComCanCheckActivationRequest\(\)](#), and [BootComRs232CheckActivationRequest\(\)](#).

7.205.2.2 BootComCanCheckActivationRequest()

```
static void BootComCanCheckActivationRequest (  
    void ) [static]
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

Referenced by [BootComCheckActivationRequest\(\)](#).

7.205.2.3 BootComCanInit()

```
static void BootComCanInit (  
    void ) [static]
```

Initializes the CAN communication interface.

Returns

none.

Referenced by BootComInit().

7.205.2.4 BootComCheckActivationRequest()

```
void BootComCheckActivationRequest (  
    void )
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.205.2.5 BootComInit()

```
void BootComInit (  
    void )
```

Initializes the communication interface.

Returns

none.

7.205.2.6 BootComRs232CheckActivationRequest()

```
static void BootComRs232CheckActivationRequest (  
    void ) [static]
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

Referenced by BootComCheckActivationRequest().

7.205.2.7 BootComRs232Init()

```
static void BootComRs232Init (  
    void ) [static]
```

Initializes the UART communication interface.

Returns

none.

Referenced by BootComInit().

7.205.2.8 CanGetSpeedConfig()

```
static unsigned char CanGetSpeedConfig (  
    unsigned short baud,  
    unsigned short * prescaler,  
    unsigned char * tseg1,  
    unsigned char * tseg2 ) [static]
```

Search algorithm to match the desired baudrate to a possible bus timing configuration.

Parameters

<i>baud</i>	The desired baudrate in kbps. Valid values are 10..1000.
<i>prescaler</i>	Pointer to where the value for the prescaler will be stored.
<i>tseg1</i>	Pointer to where the value for TSEG2 will be stored.
<i>tseg2</i>	Pointer to where the value for TSEG2 will be stored.

Returns

1 if the CAN bustiming register values were found, 0 otherwise.

Referenced by BootComCanInit().

7.205.2.9 Rs232ReceiveByte()

```
static unsigned char Rs232ReceiveByte (  
    unsigned char * data ) [static]
```

Receives a communication interface byte if one is present.

Parameters

<i>data</i>	Pointer to byte where the data is to be stored.
-------------	---

Returns

1 if a byte was received, 0 otherwise.

Referenced by `BootComRs232CheckActivationRequest()`.

7.205.3 Variable Documentation**7.205.3.1 canTiming**

```
const tCanBusTiming canTiming[] [static]
```

Initial value:

```
=
{
    { 5, 2 },
    { 6, 2 },
    { 6, 3 },
    { 7, 3 },
    { 8, 3 },
    { 9, 3 },
    { 9, 4 },
    { 10, 4 },
    { 11, 4 },
    { 12, 4 },
    { 12, 5 },
    { 13, 5 },
    { 14, 5 },
    { 15, 5 },
    { 15, 6 },
    { 16, 6 },
    { 16, 7 },
    { 16, 8 }
}
```

CAN bittiming table for dynamically calculating the bittiming settings.

According to the CAN protocol 1 bit-time can be made up of between 8..25 time quanta (TQ). The total TQ in a bit is SYNC + TSEG1 + TSEG2 with SYNC always being 1. The sample point is (SYNC + TSEG1) / (SYNC + TSEG1 + SEG2) * 100%. This array contains possible and valid time quanta configurations with a sample point between 68..78%.

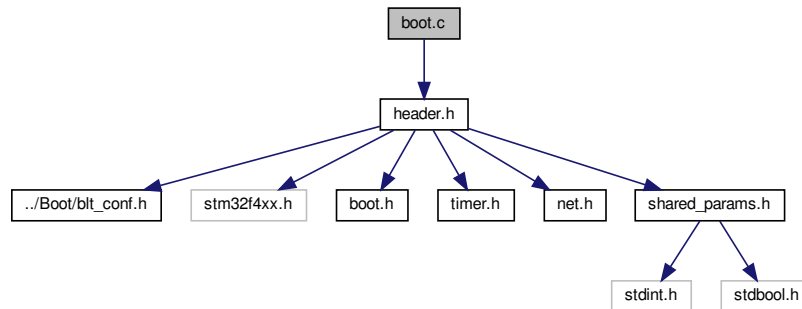
Referenced by `CanGetSpeedConfig()`.

7.206 boot.c File Reference

Demo program bootloader interface source file.

```
#include "header.h"
```

Include dependency graph for Demo/ARMCM4_STM32F4_Nucleo_F429ZI_IAR/Prog/boot.c:



Data Structures

- struct [tCanBusTiming](#)
Structure type for grouping CAN bus timing related information.

Macros

- #define [RS232_CTO_RX_PACKET_TIMEOUT_MS](#) (100u)
Timeout time for the reception of a CTO packet. The timer is started upon reception of the first packet byte.

Functions

- static void [BootComRs232Init](#) (void)
Initializes the UART communication interface.
- static void [BootComRs232CheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- static void [BootComCanInit](#) (void)
Initializes the CAN communication interface.
- static void [BootComCanCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootComInit](#) (void)
Initializes the communication interface.
- void [BootComCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootActivate](#) (void)
Bootloader activation function.
- static unsigned char [Rs232ReceiveByte](#) (unsigned char *data)
Receives a communication interface byte if one is present.
- static unsigned char [CanGetSpeedConfig](#) (unsigned short baud, unsigned short *prescaler, unsigned char *tseg1, unsigned char *tseg2)
Search algorithm to match the desired baudrate to a possible bus timing configuration.

Variables

- static UART_HandleTypeDef [rs232Handle](#)
UART handle to be used in API calls.
- static const [tCanBusTiming](#) [canTiming](#) []
CAN bittiming table for dynamically calculating the bittiming settings.
- static CAN_HandleTypeDef [canHandle](#)
CAN handle to be used in API calls.

7.206.1 Detailed Description

Demo program bootloader interface source file.

7.206.2 Function Documentation

7.206.2.1 BootActivate()

```
void BootActivate (  
    void )
```

Bootloader activation function.

Returns

none.

Referenced by [BootComCanCheckActivationRequest\(\)](#), and [BootComRs232CheckActivationRequest\(\)](#).

7.206.2.2 BootComCanCheckActivationRequest()

```
static void BootComCanCheckActivationRequest (  
    void ) [static]
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

Referenced by [BootComCheckActivationRequest\(\)](#).

7.206.2.3 BootComCanInit()

```
static void BootComCanInit (  
    void ) [static]
```

Initializes the CAN communication interface.

Returns

none.

Referenced by BootComInit().

7.206.2.4 BootComCheckActivationRequest()

```
void BootComCheckActivationRequest (  
    void )
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.206.2.5 BootComInit()

```
void BootComInit (  
    void )
```

Initializes the communication interface.

Returns

none.

7.206.2.6 BootComRs232CheckActivationRequest()

```
static void BootComRs232CheckActivationRequest (  
    void ) [static]
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

Referenced by BootComCheckActivationRequest().

7.206.2.7 BootComRs232Init()

```
static void BootComRs232Init (  
    void ) [static]
```

Initializes the UART communication interface.

Returns

none.

Referenced by BootComInit().

7.206.2.8 CanGetSpeedConfig()

```
static unsigned char CanGetSpeedConfig (  
    unsigned short baud,  
    unsigned short * prescaler,  
    unsigned char * tseg1,  
    unsigned char * tseg2 ) [static]
```

Search algorithm to match the desired baudrate to a possible bus timing configuration.

Parameters

<i>baud</i>	The desired baudrate in kbps. Valid values are 10..1000.
<i>prescaler</i>	Pointer to where the value for the prescaler will be stored.
<i>tseg1</i>	Pointer to where the value for TSEG2 will be stored.
<i>tseg2</i>	Pointer to where the value for TSEG2 will be stored.

Returns

1 if the CAN bustiming register values were found, 0 otherwise.

Referenced by BootComCanInit().

7.206.2.9 Rs232ReceiveByte()

```
static unsigned char Rs232ReceiveByte (  
    unsigned char * data ) [static]
```

Receives a communication interface byte if one is present.

Parameters

<i>data</i>	Pointer to byte where the data is to be stored.
-------------	---

Returns

1 if a byte was received, 0 otherwise.

Referenced by `BootComRs232CheckActivationRequest()`.

7.206.3 Variable Documentation**7.206.3.1 canTiming**

```
const tCanBusTiming canTiming[] [static]
```

Initial value:

```
=
{
    { 5, 2 },
    { 6, 2 },
    { 6, 3 },
    { 7, 3 },
    { 8, 3 },
    { 9, 3 },
    { 9, 4 },
    { 10, 4 },
    { 11, 4 },
    { 12, 4 },
    { 12, 5 },
    { 13, 5 },
    { 14, 5 },
    { 15, 5 },
    { 15, 6 },
    { 16, 6 },
    { 16, 7 },
    { 16, 8 }
}
```

CAN bittiming table for dynamically calculating the bittiming settings.

According to the CAN protocol 1 bit-time can be made up of between 8..25 time quanta (TQ). The total TQ in a bit is SYNC + TSEG1 + TSEG2 with SYNC always being 1. The sample point is (SYNC + TSEG1) / (SYNC + TSEG1 + SEG2) * 100%. This array contains possible and valid time quanta configurations with a sample point between 68..78%.

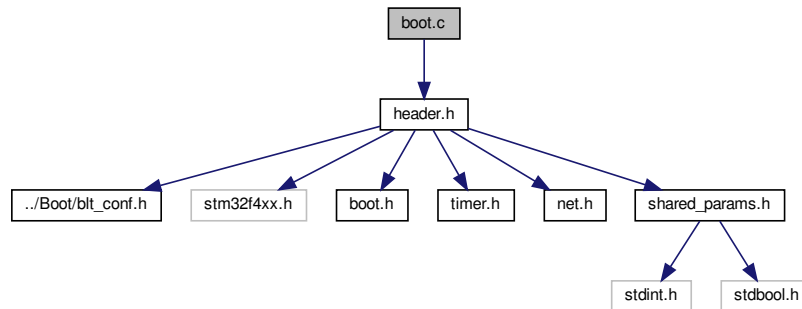
Referenced by `CanGetSpeedConfig()`.

7.207 boot.c File Reference

Demo program bootloader interface source file.

```
#include "header.h"
```

Include dependency graph for Demo/ARMCM4_STM32F4_Nucleo_F429ZI_Keil/Prog/boot.c:



Data Structures

- struct [tCanBusTiming](#)
Structure type for grouping CAN bus timing related information.

Macros

- #define [RS232_CTO_RX_PACKET_TIMEOUT_MS](#) (100u)
Timeout time for the reception of a CTO packet. The timer is started upon reception of the first packet byte.

Functions

- static void [BootComRs232Init](#) (void)
Initializes the UART communication interface.
- static void [BootComRs232CheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- static void [BootComCanInit](#) (void)
Initializes the CAN communication interface.
- static void [BootComCanCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootComInit](#) (void)
Initializes the communication interface.
- void [BootComCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootActivate](#) (void)
Bootloader activation function.
- static unsigned char [Rs232ReceiveByte](#) (unsigned char *data)
Receives a communication interface byte if one is present.
- static unsigned char [CanGetSpeedConfig](#) (unsigned short baud, unsigned short *prescaler, unsigned char *tseg1, unsigned char *tseg2)
Search algorithm to match the desired baudrate to a possible bus timing configuration.

Variables

- static UART_HandleTypeDef [rs232Handle](#)
UART handle to be used in API calls.
- static const [tCanBusTiming](#) [canTiming](#) []
CAN bittiming table for dynamically calculating the bittiming settings.
- static CAN_HandleTypeDef [canHandle](#)
CAN handle to be used in API calls.

7.207.1 Detailed Description

Demo program bootloader interface source file.

7.207.2 Function Documentation

7.207.2.1 BootActivate()

```
void BootActivate (  
    void )
```

Bootloader activation function.

Returns

none.

Referenced by [BootComCanCheckActivationRequest\(\)](#), and [BootComRs232CheckActivationRequest\(\)](#).

7.207.2.2 BootComCanCheckActivationRequest()

```
static void BootComCanCheckActivationRequest (  
    void ) [static]
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

Referenced by [BootComCheckActivationRequest\(\)](#).

7.207.2.3 BootComCanInit()

```
static void BootComCanInit (  
    void ) [static]
```

Initializes the CAN communication interface.

Returns

none.

Referenced by BootComInit().

7.207.2.4 BootComCheckActivationRequest()

```
void BootComCheckActivationRequest (  
    void )
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.207.2.5 BootComInit()

```
void BootComInit (  
    void )
```

Initializes the communication interface.

Returns

none.

7.207.2.6 BootComRs232CheckActivationRequest()

```
static void BootComRs232CheckActivationRequest (  
    void ) [static]
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

Referenced by BootComCheckActivationRequest().

7.207.2.7 BootComRs232Init()

```
static void BootComRs232Init (  
    void ) [static]
```

Initializes the UART communication interface.

Returns

none.

Referenced by BootComInit().

7.207.2.8 CanGetSpeedConfig()

```
static unsigned char CanGetSpeedConfig (  
    unsigned short baud,  
    unsigned short * prescaler,  
    unsigned char * tseg1,  
    unsigned char * tseg2 ) [static]
```

Search algorithm to match the desired baudrate to a possible bus timing configuration.

Parameters

<i>baud</i>	The desired baudrate in kbps. Valid values are 10..1000.
<i>prescaler</i>	Pointer to where the value for the prescaler will be stored.
<i>tseg1</i>	Pointer to where the value for TSEG2 will be stored.
<i>tseg2</i>	Pointer to where the value for TSEG2 will be stored.

Returns

1 if the CAN bustiming register values were found, 0 otherwise.

Referenced by BootComCanInit().

7.207.2.9 Rs232ReceiveByte()

```
static unsigned char Rs232ReceiveByte (  
    unsigned char * data ) [static]
```

Receives a communication interface byte if one is present.

Parameters

<i>data</i>	Pointer to byte where the data is to be stored.
-------------	---

Returns

1 if a byte was received, 0 otherwise.

Referenced by `BootComRs232CheckActivationRequest()`.

7.207.3 Variable Documentation**7.207.3.1 canTiming**

```
const tCanBusTiming canTiming[] [static]
```

Initial value:

```
=
{
    { 5, 2 },
    { 6, 2 },
    { 6, 3 },
    { 7, 3 },
    { 8, 3 },
    { 9, 3 },
    { 9, 4 },
    { 10, 4 },
    { 11, 4 },
    { 12, 4 },
    { 12, 5 },
    { 13, 5 },
    { 14, 5 },
    { 15, 5 },
    { 15, 6 },
    { 16, 6 },
    { 16, 7 },
    { 16, 8 }
}
```

CAN bittiming table for dynamically calculating the bittiming settings.

According to the CAN protocol 1 bit-time can be made up of between 8..25 time quanta (TQ). The total TQ in a bit is SYNC + TSEG1 + TSEG2 with SYNC always being 1. The sample point is (SYNC + TSEG1) / (SYNC + TSEG1 + SEG2) * 100%. This array contains possible and valid time quanta configurations with a sample point between 68..78%.

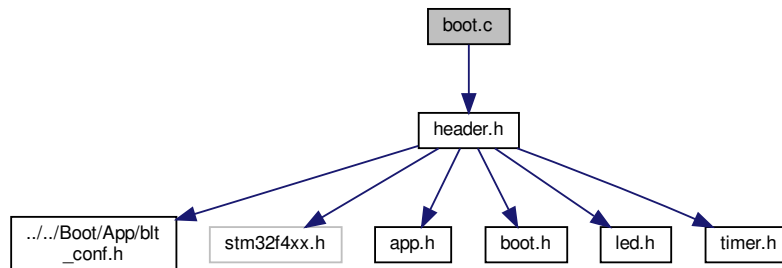
Referenced by `CanGetSpeedConfig()`.

7.208 boot.c File Reference

Demo program bootloader interface source file.

```
#include "header.h"
```

Include dependency graph for Demo/ARMCM4_STM32F4_Olimex_STM32P405_CubeIDE/Prog/App/boot.c:



Data Structures

- struct [tCanBusTiming](#)
Structure type for grouping CAN bus timing related information.

Macros

- #define [RS232_CTO_RX_PACKET_TIMEOUT_MS](#) (100u)
Timeout time for the reception of a CTO packet. The timer is started upon reception of the first packet byte.

Functions

- static void [BootComRs232Init](#) (void)
Initializes the UART communication interface.
- static void [BootComRs232CheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- static void [BootComCanInit](#) (void)
Initializes the CAN communication interface.
- static void [BootComCanCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootComInit](#) (void)
Initializes the communication interface.
- void [BootComCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootActivate](#) (void)
Bootloader activation function.
- static unsigned char [Rs232ReceiveByte](#) (unsigned char *data)
Receives a communication interface byte if one is present.
- static unsigned char [CanGetSpeedConfig](#) (unsigned short baud, unsigned short *prescaler, unsigned char *tseg1, unsigned char *tseg2)
Search algorithm to match the desired baudrate to a possible bus timing configuration.

Variables

- static UART_HandleTypeDef [rs232Handle](#)
UART handle to be used in API calls.
- static const [tCanBusTiming](#) [canTiming](#) []
CAN bittiming table for dynamically calculating the bittiming settings.
- static CAN_HandleTypeDef [canHandle](#)
CAN handle to be used in API calls.

7.208.1 Detailed Description

Demo program bootloader interface source file.

7.208.2 Function Documentation

7.208.2.1 BootActivate()

```
void BootActivate (  
    void )
```

Bootloader activation function.

Returns

none.

Referenced by [BootComCanCheckActivationRequest\(\)](#), and [BootComRs232CheckActivationRequest\(\)](#).

7.208.2.2 BootComCanCheckActivationRequest()

```
static void BootComCanCheckActivationRequest (  
    void ) [static]
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

Referenced by [BootComCheckActivationRequest\(\)](#).

7.208.2.3 BootComCanInit()

```
static void BootComCanInit (
    void ) [static]
```

Initializes the CAN communication interface.

Returns

none.

Referenced by BootComInit().

7.208.2.4 BootComCheckActivationRequest()

```
void BootComCheckActivationRequest (
    void )
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.208.2.5 BootComInit()

```
void BootComInit (
    void )
```

Initializes the communication interface.

Returns

none.

7.208.2.6 BootComRs232CheckActivationRequest()

```
static void BootComRs232CheckActivationRequest (
    void ) [static]
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

Referenced by BootComCheckActivationRequest().

7.208.2.7 BootComRs232Init()

```
static void BootComRs232Init (  
    void ) [static]
```

Initializes the UART communication interface.

Returns

none.

Referenced by BootComInit().

7.208.2.8 CanGetSpeedConfig()

```
static unsigned char CanGetSpeedConfig (  
    unsigned short baud,  
    unsigned short * prescaler,  
    unsigned char * tseg1,  
    unsigned char * tseg2 ) [static]
```

Search algorithm to match the desired baudrate to a possible bus timing configuration.

Parameters

<i>baud</i>	The desired baudrate in kbps. Valid values are 10..1000.
<i>prescaler</i>	Pointer to where the value for the prescaler will be stored.
<i>tseg1</i>	Pointer to where the value for TSEG2 will be stored.
<i>tseg2</i>	Pointer to where the value for TSEG2 will be stored.

Returns

1 if the CAN bustiming register values were found, 0 otherwise.

Referenced by BootComCanInit().

7.208.2.9 Rs232ReceiveByte()

```
static unsigned char Rs232ReceiveByte (  
    unsigned char * data ) [static]
```

Receives a communication interface byte if one is present.

Parameters

<i>data</i>	Pointer to byte where the data is to be stored.
-------------	---

Returns

1 if a byte was received, 0 otherwise.

Referenced by `BootComRs232CheckActivationRequest()`.

7.208.3 Variable Documentation**7.208.3.1 canTiming**

```
const tCanBusTiming canTiming[] [static]
```

Initial value:

```
=  
{  
    { 5, 2 },  
    { 6, 2 },  
    { 6, 3 },  
    { 7, 3 },  
    { 8, 3 },  
    { 9, 3 },  
    { 9, 4 },  
    { 10, 4 },  
    { 11, 4 },  
    { 12, 4 },  
    { 12, 5 },  
    { 13, 5 },  
    { 14, 5 },  
    { 15, 5 },  
    { 15, 6 },  
    { 16, 6 },  
    { 16, 7 },  
    { 16, 8 }  
}
```

CAN bittiming table for dynamically calculating the bittiming settings.

According to the CAN protocol 1 bit-time can be made up of between 8..25 time quanta (TQ). The total TQ in a bit is SYNC + TSEG1 + TSEG2 with SYNC always being 1. The sample point is (SYNC + TSEG1) / (SYNC + TSEG1 + SEG2) * 100%. This array contains possible and valid time quanta configurations with a sample point between 68..78%.

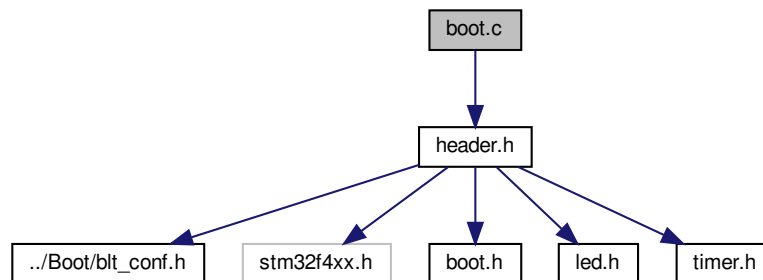
Referenced by `CanGetSpeedConfig()`.

7.209 boot.c File Reference

Demo program bootloader interface source file.

```
#include "header.h"
```

Include dependency graph for Demo/ARMCM4_STM32F4_Olimex_STM32P405_GCC/Prog/boot.c:



Data Structures

- struct [tCanBusTiming](#)
Structure type for grouping CAN bus timing related information.

Macros

- `#define RS232_CTO_RX_PACKET_TIMEOUT_MS (100u)`
Timeout time for the reception of a CTO packet. The timer is started upon reception of the first packet byte.

Functions

- static void [BootComRs232Init](#) (void)
Initializes the UART communication interface.
- static void [BootComRs232CheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- static void [BootComCanInit](#) (void)
Initializes the CAN communication interface.
- static void [BootComCanCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootComInit](#) (void)
Initializes the communication interface.
- void [BootComCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootActivate](#) (void)
Bootloader activation function.
- static unsigned char [Rs232ReceiveByte](#) (unsigned char *data)
Receives a communication interface byte if one is present.
- static unsigned char [CanGetSpeedConfig](#) (unsigned short baud, unsigned short *prescaler, unsigned char *tseg1, unsigned char *tseg2)
Search algorithm to match the desired baudrate to a possible bus timing configuration.

Variables

- static UART_HandleTypeDef [rs232Handle](#)
UART handle to be used in API calls.
- static const [tCanBusTiming](#) [canTiming](#) []
CAN bittiming table for dynamically calculating the bittiming settings.
- static CAN_HandleTypeDef [canHandle](#)
CAN handle to be used in API calls.

7.209.1 Detailed Description

Demo program bootloader interface source file.

7.209.2 Function Documentation

7.209.2.1 BootActivate()

```
void BootActivate (  
    void )
```

Bootloader activation function.

Returns

none.

Referenced by [BootComCanCheckActivationRequest\(\)](#), and [BootComRs232CheckActivationRequest\(\)](#).

7.209.2.2 BootComCanCheckActivationRequest()

```
static void BootComCanCheckActivationRequest (  
    void ) [static]
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

Referenced by [BootComCheckActivationRequest\(\)](#).

7.209.2.3 BootComCanInit()

```
static void BootComCanInit (
    void ) [static]
```

Initializes the CAN communication interface.

Returns

none.

Referenced by BootComInit().

7.209.2.4 BootComCheckActivationRequest()

```
void BootComCheckActivationRequest (
    void )
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.209.2.5 BootComInit()

```
void BootComInit (
    void )
```

Initializes the communication interface.

Returns

none.

7.209.2.6 BootComRs232CheckActivationRequest()

```
static void BootComRs232CheckActivationRequest (
    void ) [static]
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

Referenced by BootComCheckActivationRequest().

7.209.2.7 BootComRs232Init()

```
static void BootComRs232Init (  
    void ) [static]
```

Initializes the UART communication interface.

Returns

none.

Referenced by BootComInit().

7.209.2.8 CanGetSpeedConfig()

```
static unsigned char CanGetSpeedConfig (  
    unsigned short baud,  
    unsigned short * prescaler,  
    unsigned char * tseg1,  
    unsigned char * tseg2 ) [static]
```

Search algorithm to match the desired baudrate to a possible bus timing configuration.

Parameters

<i>baud</i>	The desired baudrate in kbps. Valid values are 10..1000.
<i>prescaler</i>	Pointer to where the value for the prescaler will be stored.
<i>tseg1</i>	Pointer to where the value for TSEG2 will be stored.
<i>tseg2</i>	Pointer to where the value for TSEG2 will be stored.

Returns

1 if the CAN bustiming register values were found, 0 otherwise.

Referenced by BootComCanInit().

7.209.2.9 Rs232ReceiveByte()

```
static unsigned char Rs232ReceiveByte (  
    unsigned char * data ) [static]
```

Receives a communication interface byte if one is present.

Parameters

<i>data</i>	Pointer to byte where the data is to be stored.
-------------	---

Returns

1 if a byte was received, 0 otherwise.

Referenced by `BootComRs232CheckActivationRequest()`.

7.209.3 Variable Documentation

7.209.3.1 canTiming

```
const tCanBusTiming canTiming[] [static]
```

Initial value:

```
=
{
    { 5, 2 },
    { 6, 2 },
    { 6, 3 },
    { 7, 3 },
    { 8, 3 },
    { 9, 3 },
    { 9, 4 },
    { 10, 4 },
    { 11, 4 },
    { 12, 4 },
    { 12, 5 },
    { 13, 5 },
    { 14, 5 },
    { 15, 5 },
    { 15, 6 },
    { 16, 6 },
    { 16, 7 },
    { 16, 8 }
}
```

CAN bittiming table for dynamically calculating the bittiming settings.

According to the CAN protocol 1 bit-time can be made up of between 8..25 time quanta (TQ). The total TQ in a bit is SYNC + TSEG1 + TSEG2 with SYNC always being 1. The sample point is (SYNC + TSEG1) / (SYNC + TSEG1 + SEG2) * 100%. This array contains possible and valid time quanta configurations with a sample point between 68..78%.

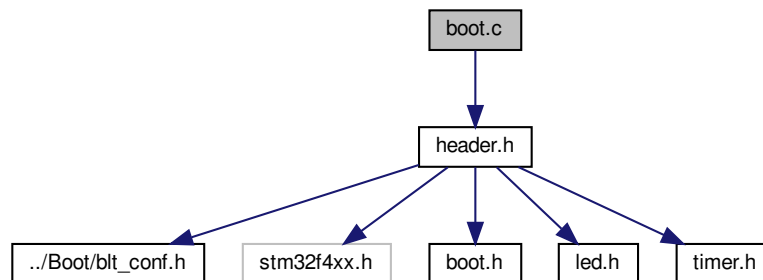
Referenced by `CanGetSpeedConfig()`.

7.210 boot.c File Reference

Demo program bootloader interface source file.

```
#include "header.h"
```

Include dependency graph for Demo/ARMCM4_STM32F4_Olimex_STM32P405_IAR/Prog/boot.c:



Data Structures

- struct [tCanBusTiming](#)
Structure type for grouping CAN bus timing related information.

Macros

- `#define RS232_CTO_RX_PACKET_TIMEOUT_MS (100u)`
Timeout time for the reception of a CTO packet. The timer is started upon reception of the first packet byte.

Functions

- static void [BootComRs232Init](#) (void)
Initializes the UART communication interface.
- static void [BootComRs232CheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- static void [BootComCanInit](#) (void)
Initializes the CAN communication interface.
- static void [BootComCanCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootComInit](#) (void)
Initializes the communication interface.
- void [BootComCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootActivate](#) (void)
Bootloader activation function.
- static unsigned char [Rs232ReceiveByte](#) (unsigned char *data)
Receives a communication interface byte if one is present.
- static unsigned char [CanGetSpeedConfig](#) (unsigned short baud, unsigned short *prescaler, unsigned char *tseg1, unsigned char *tseg2)
Search algorithm to match the desired baudrate to a possible bus timing configuration.

Variables

- static UART_HandleTypeDef [rs232Handle](#)
UART handle to be used in API calls.
- static const [tCanBusTiming](#) [canTiming](#) []
CAN bittiming table for dynamically calculating the bittiming settings.
- static CAN_HandleTypeDef [canHandle](#)
CAN handle to be used in API calls.

7.210.1 Detailed Description

Demo program bootloader interface source file.

7.210.2 Function Documentation

7.210.2.1 BootActivate()

```
void BootActivate (  
    void )
```

Bootloader activation function.

Returns

none.

Referenced by [BootComCanCheckActivationRequest\(\)](#), and [BootComRs232CheckActivationRequest\(\)](#).

7.210.2.2 BootComCanCheckActivationRequest()

```
static void BootComCanCheckActivationRequest (  
    void ) [static]
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

Referenced by [BootComCheckActivationRequest\(\)](#).

7.210.2.3 BootComCanInit()

```
static void BootComCanInit (
    void ) [static]
```

Initializes the CAN communication interface.

Returns

none.

Referenced by BootComInit().

7.210.2.4 BootComCheckActivationRequest()

```
void BootComCheckActivationRequest (
    void )
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.210.2.5 BootComInit()

```
void BootComInit (
    void )
```

Initializes the communication interface.

Returns

none.

7.210.2.6 BootComRs232CheckActivationRequest()

```
static void BootComRs232CheckActivationRequest (
    void ) [static]
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

Referenced by BootComCheckActivationRequest().

7.210.2.7 BootComRs232Init()

```
static void BootComRs232Init (  
    void ) [static]
```

Initializes the UART communication interface.

Returns

none.

Referenced by BootComInit().

7.210.2.8 CanGetSpeedConfig()

```
static unsigned char CanGetSpeedConfig (  
    unsigned short baud,  
    unsigned short * prescaler,  
    unsigned char * tseg1,  
    unsigned char * tseg2 ) [static]
```

Search algorithm to match the desired baudrate to a possible bus timing configuration.

Parameters

<i>baud</i>	The desired baudrate in kbps. Valid values are 10..1000.
<i>prescaler</i>	Pointer to where the value for the prescaler will be stored.
<i>tseg1</i>	Pointer to where the value for TSEG2 will be stored.
<i>tseg2</i>	Pointer to where the value for TSEG2 will be stored.

Returns

1 if the CAN bustiming register values were found, 0 otherwise.

Referenced by BootComCanInit().

7.210.2.9 Rs232ReceiveByte()

```
static unsigned char Rs232ReceiveByte (  
    unsigned char * data ) [static]
```

Receives a communication interface byte if one is present.

Parameters

<i>data</i>	Pointer to byte where the data is to be stored.
-------------	---

Returns

1 if a byte was received, 0 otherwise.

Referenced by `BootComRs232CheckActivationRequest()`.

7.210.3 Variable Documentation**7.210.3.1 canTiming**

```
const tCanBusTiming canTiming[] [static]
```

Initial value:

```
=
{
    { 5, 2 },
    { 6, 2 },
    { 6, 3 },
    { 7, 3 },
    { 8, 3 },
    { 9, 3 },
    { 9, 4 },
    { 10, 4 },
    { 11, 4 },
    { 12, 4 },
    { 12, 5 },
    { 13, 5 },
    { 14, 5 },
    { 15, 5 },
    { 15, 6 },
    { 16, 6 },
    { 16, 7 },
    { 16, 8 }
}
```

CAN bittiming table for dynamically calculating the bittiming settings.

According to the CAN protocol 1 bit-time can be made up of between 8..25 time quanta (TQ). The total TQ in a bit is SYNC + TSEG1 + TSEG2 with SYNC always being 1. The sample point is (SYNC + TSEG1) / (SYNC + TSEG1 + SEG2) * 100%. This array contains possible and valid time quanta configurations with a sample point between 68..78%.

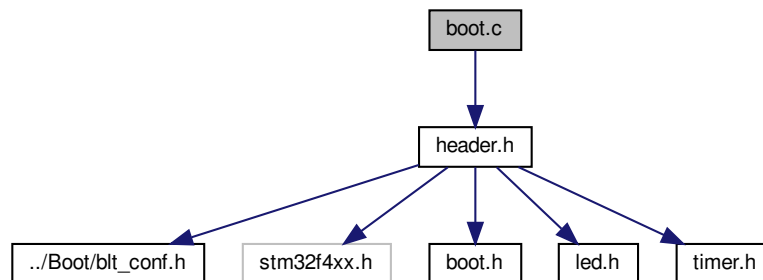
Referenced by `CanGetSpeedConfig()`.

7.211 boot.c File Reference

Demo program bootloader interface source file.

```
#include "header.h"
```

Include dependency graph for Demo/ARMCM4_STM32F4_Olimex_STM32P405_Keil/Prog/boot.c:



Data Structures

- struct [tCanBusTiming](#)
Structure type for grouping CAN bus timing related information.

Macros

- `#define RS232_CTO_RX_PACKET_TIMEOUT_MS (100u)`
Timeout time for the reception of a CTO packet. The timer is started upon reception of the first packet byte.

Functions

- static void [BootComRs232Init](#) (void)
Initializes the UART communication interface.
- static void [BootComRs232CheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- static void [BootComCanInit](#) (void)
Initializes the CAN communication interface.
- static void [BootComCanCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootComInit](#) (void)
Initializes the communication interface.
- void [BootComCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootActivate](#) (void)
Bootloader activation function.
- static unsigned char [Rs232ReceiveByte](#) (unsigned char *data)
Receives a communication interface byte if one is present.
- static unsigned char [CanGetSpeedConfig](#) (unsigned short baud, unsigned short *prescaler, unsigned char *tseg1, unsigned char *tseg2)
Search algorithm to match the desired baudrate to a possible bus timing configuration.

Variables

- static UART_HandleTypeDef [rs232Handle](#)
UART handle to be used in API calls.
- static const [tCanBusTiming](#) [canTiming](#) []
CAN bittiming table for dynamically calculating the bittiming settings.
- static CAN_HandleTypeDef [canHandle](#)
CAN handle to be used in API calls.

7.211.1 Detailed Description

Demo program bootloader interface source file.

7.211.2 Function Documentation

7.211.2.1 BootActivate()

```
void BootActivate (  
    void )
```

Bootloader activation function.

Returns

none.

Referenced by [BootComCanCheckActivationRequest\(\)](#), and [BootComRs232CheckActivationRequest\(\)](#).

7.211.2.2 BootComCanCheckActivationRequest()

```
static void BootComCanCheckActivationRequest (  
    void ) [static]
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

Referenced by [BootComCheckActivationRequest\(\)](#).

7.211.2.3 BootComCanInit()

```
static void BootComCanInit (  
    void ) [static]
```

Initializes the CAN communication interface.

Returns

none.

Referenced by BootComInit().

7.211.2.4 BootComCheckActivationRequest()

```
void BootComCheckActivationRequest (  
    void )
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.211.2.5 BootComInit()

```
void BootComInit (  
    void )
```

Initializes the communication interface.

Returns

none.

7.211.2.6 BootComRs232CheckActivationRequest()

```
static void BootComRs232CheckActivationRequest (  
    void ) [static]
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

Referenced by BootComCheckActivationRequest().

7.211.2.7 BootComRs232Init()

```
static void BootComRs232Init (  
    void ) [static]
```

Initializes the UART communication interface.

Returns

none.

Referenced by BootComInit().

7.211.2.8 CanGetSpeedConfig()

```
static unsigned char CanGetSpeedConfig (  
    unsigned short baud,  
    unsigned short * prescaler,  
    unsigned char * tseg1,  
    unsigned char * tseg2 ) [static]
```

Search algorithm to match the desired baudrate to a possible bus timing configuration.

Parameters

<i>baud</i>	The desired baudrate in kbps. Valid values are 10..1000.
<i>prescaler</i>	Pointer to where the value for the prescaler will be stored.
<i>tseg1</i>	Pointer to where the value for TSEG2 will be stored.
<i>tseg2</i>	Pointer to where the value for TSEG2 will be stored.

Returns

1 if the CAN bustiming register values were found, 0 otherwise.

Referenced by BootComCanInit().

7.211.2.9 Rs232ReceiveByte()

```
static unsigned char Rs232ReceiveByte (  
    unsigned char * data ) [static]
```

Receives a communication interface byte if one is present.

Parameters

<i>data</i>	Pointer to byte where the data is to be stored.
-------------	---

Returns

1 if a byte was received, 0 otherwise.

Referenced by `BootComRs232CheckActivationRequest()`.

7.211.3 Variable Documentation

7.211.3.1 canTiming

```
const tCanBusTiming canTiming[] [static]
```

Initial value:

```
=
{
    { 5, 2 },
    { 6, 2 },
    { 6, 3 },
    { 7, 3 },
    { 8, 3 },
    { 9, 3 },
    { 9, 4 },
    { 10, 4 },
    { 11, 4 },
    { 12, 4 },
    { 12, 5 },
    { 13, 5 },
    { 14, 5 },
    { 15, 5 },
    { 15, 6 },
    { 16, 6 },
    { 16, 7 },
    { 16, 8 }
}
```

CAN bittiming table for dynamically calculating the bittiming settings.

According to the CAN protocol 1 bit-time can be made up of between 8..25 time quanta (TQ). The total TQ in a bit is SYNC + TSEG1 + TSEG2 with SYNC always being 1. The sample point is (SYNC + TSEG1) / (SYNC + TSEG1 + SEG2) * 100%. This array contains possible and valid time quanta configurations with a sample point between 68..78%.

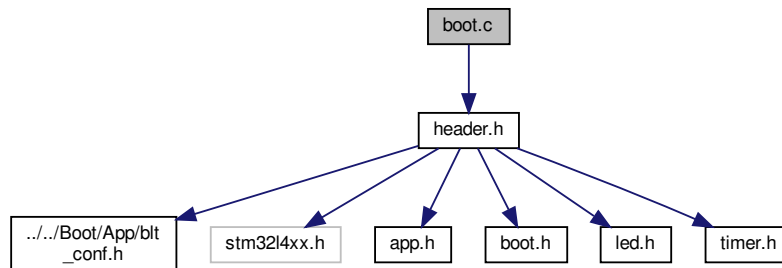
Referenced by `CanGetSpeedConfig()`.

7.212 boot.c File Reference

Demo program bootloader interface source file.

```
#include "header.h"
```

Include dependency graph for Demo/ARMCM4_STM32L4_Nucleo_L476RG_CubeIDE/Prog/App/boot.c:



Data Structures

- struct [tCanBusTiming](#)
Structure type for grouping CAN bus timing related information.

Macros

- #define [RS232_CTO_RX_PACKET_TIMEOUT_MS](#) (100u)
Timeout time for the reception of a CTO packet. The timer is started upon reception of the first packet byte.

Functions

- static void [BootComRs232Init](#) (void)
Initializes the UART communication interface.
- static void [BootComRs232CheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- static void [BootComCanInit](#) (void)
Initializes the CAN communication interface.
- static void [BootComCanCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootComInit](#) (void)
Initializes the communication interface.
- void [BootComCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootActivate](#) (void)
Bootloader activation function.
- static unsigned char [Rs232ReceiveByte](#) (unsigned char *data)
Receives a communication interface byte if one is present.
- static unsigned char [CanGetSpeedConfig](#) (unsigned short baud, unsigned short *prescaler, unsigned char *tseg1, unsigned char *tseg2)
Search algorithm to match the desired baudrate to a possible bus timing configuration.

Variables

- static UART_HandleTypeDef [rs232Handle](#)
UART handle to be used in API calls.
- static const [tCanBusTiming](#) [canTiming](#) []
CAN bittiming table for dynamically calculating the bittiming settings.
- static CAN_HandleTypeDef [canHandle](#)
CAN handle to be used in API calls.

7.212.1 Detailed Description

Demo program bootloader interface source file.

7.212.2 Function Documentation

7.212.2.1 BootActivate()

```
void BootActivate (  
    void )
```

Bootloader activation function.

Returns

none.

Referenced by [BootComCanCheckActivationRequest\(\)](#), and [BootComRs232CheckActivationRequest\(\)](#).

7.212.2.2 BootComCanCheckActivationRequest()

```
static void BootComCanCheckActivationRequest (  
    void ) [static]
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

Referenced by [BootComCheckActivationRequest\(\)](#).

7.212.2.3 BootComCanInit()

```
static void BootComCanInit (
    void ) [static]
```

Initializes the CAN communication interface.

Returns

none.

Referenced by BootComInit().

7.212.2.4 BootComCheckActivationRequest()

```
void BootComCheckActivationRequest (
    void )
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.212.2.5 BootComInit()

```
void BootComInit (
    void )
```

Initializes the communication interface.

Returns

none.

7.212.2.6 BootComRs232CheckActivationRequest()

```
static void BootComRs232CheckActivationRequest (
    void ) [static]
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

Referenced by BootComCheckActivationRequest().

7.212.2.7 BootComRs232Init()

```
static void BootComRs232Init (  
    void ) [static]
```

Initializes the UART communication interface.

Returns

none.

Referenced by BootComInit().

7.212.2.8 CanGetSpeedConfig()

```
static unsigned char CanGetSpeedConfig (  
    unsigned short baud,  
    unsigned short * prescaler,  
    unsigned char * tseg1,  
    unsigned char * tseg2 ) [static]
```

Search algorithm to match the desired baudrate to a possible bus timing configuration.

Parameters

<i>baud</i>	The desired baudrate in kbps. Valid values are 10..1000.
<i>prescaler</i>	Pointer to where the value for the prescaler will be stored.
<i>tseg1</i>	Pointer to where the value for TSEG2 will be stored.
<i>tseg2</i>	Pointer to where the value for TSEG2 will be stored.

Returns

1 if the CAN bustiming register values were found, 0 otherwise.

Referenced by BootComCanInit().

7.212.2.9 Rs232ReceiveByte()

```
static unsigned char Rs232ReceiveByte (  
    unsigned char * data ) [static]
```

Receives a communication interface byte if one is present.

Parameters

<i>data</i>	Pointer to byte where the data is to be stored.
-------------	---

Returns

1 if a byte was received, 0 otherwise.

Referenced by `BootComRs232CheckActivationRequest()`.

7.212.3 Variable Documentation**7.212.3.1 canTiming**

```
const tCanBusTiming canTiming[] [static]
```

Initial value:

```
=
{
    { 5, 2 },
    { 6, 2 },
    { 6, 3 },
    { 7, 3 },
    { 8, 3 },
    { 9, 3 },
    { 9, 4 },
    { 10, 4 },
    { 11, 4 },
    { 12, 4 },
    { 12, 5 },
    { 13, 5 },
    { 14, 5 },
    { 15, 5 },
    { 15, 6 },
    { 16, 6 },
    { 16, 7 },
    { 16, 8 }
}
```

CAN bittiming table for dynamically calculating the bittiming settings.

According to the CAN protocol 1 bit-time can be made up of between 8..25 time quanta (TQ). The total TQ in a bit is SYNC + TSEG1 + TSEG2 with SYNC always being 1. The sample point is (SYNC + TSEG1) / (SYNC + TSEG1 + SEG2) * 100%. This array contains possible and valid time quanta configurations with a sample point between 68..78%.

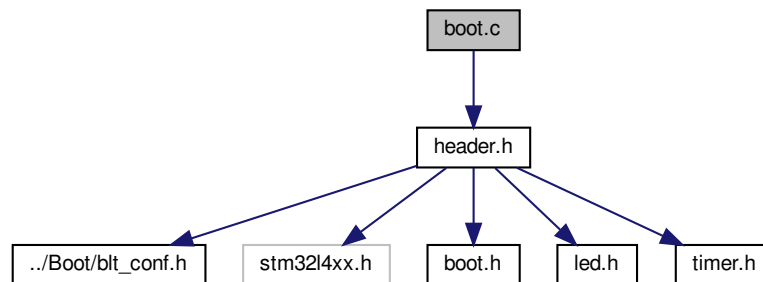
Referenced by `CanGetSpeedConfig()`.

7.213 boot.c File Reference

Demo program bootloader interface source file.

```
#include "header.h"
```

Include dependency graph for Demo/ARMCM4_STM32L4_Nucleo_L476RG_GCC/Prog/boot.c:



Data Structures

- struct [tCanBusTiming](#)
Structure type for grouping CAN bus timing related information.

Macros

- `#define RS232_CTO_RX_PACKET_TIMEOUT_MS (100u)`
Timeout time for the reception of a CTO packet. The timer is started upon reception of the first packet byte.

Functions

- static void [BootComRs232Init](#) (void)
Initializes the UART communication interface.
- static void [BootComRs232CheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- static void [BootComCanInit](#) (void)
Initializes the CAN communication interface.
- static void [BootComCanCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootComInit](#) (void)
Initializes the communication interface.
- void [BootComCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootActivate](#) (void)
Bootloader activation function.
- static unsigned char [Rs232ReceiveByte](#) (unsigned char *data)
Receives a communication interface byte if one is present.
- static unsigned char [CanGetSpeedConfig](#) (unsigned short baud, unsigned short *prescaler, unsigned char *tseg1, unsigned char *tseg2)
Search algorithm to match the desired baudrate to a possible bus timing configuration.

Variables

- static UART_HandleTypeDef [rs232Handle](#)
UART handle to be used in API calls.
- static const [tCanBusTiming](#) [canTiming](#) []
CAN bittiming table for dynamically calculating the bittiming settings.
- static CAN_HandleTypeDef [canHandle](#)
CAN handle to be used in API calls.

7.213.1 Detailed Description

Demo program bootloader interface source file.

7.213.2 Function Documentation

7.213.2.1 BootActivate()

```
void BootActivate (  
    void )
```

Bootloader activation function.

Returns

none.

Referenced by [BootComCanCheckActivationRequest\(\)](#), and [BootComRs232CheckActivationRequest\(\)](#).

7.213.2.2 BootComCanCheckActivationRequest()

```
static void BootComCanCheckActivationRequest (  
    void ) [static]
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

Referenced by [BootComCheckActivationRequest\(\)](#).

7.213.2.3 BootComCanInit()

```
static void BootComCanInit (  
    void ) [static]
```

Initializes the CAN communication interface.

Returns

none.

Referenced by BootComInit().

7.213.2.4 BootComCheckActivationRequest()

```
void BootComCheckActivationRequest (  
    void )
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.213.2.5 BootComInit()

```
void BootComInit (  
    void )
```

Initializes the communication interface.

Returns

none.

7.213.2.6 BootComRs232CheckActivationRequest()

```
static void BootComRs232CheckActivationRequest (  
    void ) [static]
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

Referenced by BootComCheckActivationRequest().

7.213.2.7 BootComRs232Init()

```
static void BootComRs232Init (  
    void ) [static]
```

Initializes the UART communication interface.

Returns

none.

Referenced by BootComInit().

7.213.2.8 CanGetSpeedConfig()

```
static unsigned char CanGetSpeedConfig (  
    unsigned short baud,  
    unsigned short * prescaler,  
    unsigned char * tseg1,  
    unsigned char * tseg2 ) [static]
```

Search algorithm to match the desired baudrate to a possible bus timing configuration.

Parameters

<i>baud</i>	The desired baudrate in kbps. Valid values are 10..1000.
<i>prescaler</i>	Pointer to where the value for the prescaler will be stored.
<i>tseg1</i>	Pointer to where the value for TSEG2 will be stored.
<i>tseg2</i>	Pointer to where the value for TSEG2 will be stored.

Returns

1 if the CAN bustiming register values were found, 0 otherwise.

Referenced by BootComCanInit().

7.213.2.9 Rs232ReceiveByte()

```
static unsigned char Rs232ReceiveByte (  
    unsigned char * data ) [static]
```

Receives a communication interface byte if one is present.

Parameters

<i>data</i>	Pointer to byte where the data is to be stored.
-------------	---

Returns

1 if a byte was received, 0 otherwise.

Referenced by `BootComRs232CheckActivationRequest()`.

7.213.3 Variable Documentation**7.213.3.1 canTiming**

```
const tCanBusTiming canTiming[] [static]
```

Initial value:

```
=
{
    { 5, 2 },
    { 6, 2 },
    { 6, 3 },
    { 7, 3 },
    { 8, 3 },
    { 9, 3 },
    { 9, 4 },
    { 10, 4 },
    { 11, 4 },
    { 12, 4 },
    { 12, 5 },
    { 13, 5 },
    { 14, 5 },
    { 15, 5 },
    { 15, 6 },
    { 16, 6 },
    { 16, 7 },
    { 16, 8 }
}
```

CAN bittiming table for dynamically calculating the bittiming settings.

According to the CAN protocol 1 bit-time can be made up of between 8..25 time quanta (TQ). The total TQ in a bit is SYNC + TSEG1 + TSEG2 with SYNC always being 1. The sample point is (SYNC + TSEG1) / (SYNC + TSEG1 + SEG2) * 100%. This array contains possible and valid time quanta configurations with a sample point between 68..78%.

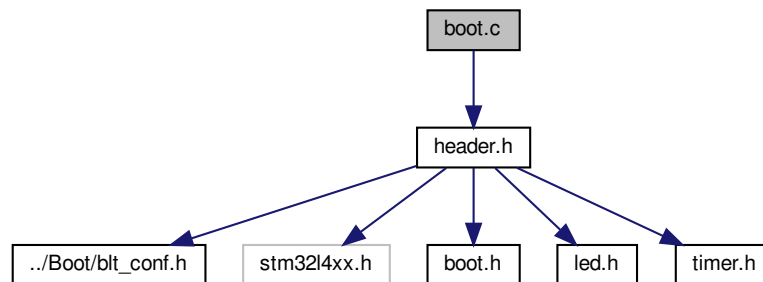
Referenced by `CanGetSpeedConfig()`.

7.214 boot.c File Reference

Demo program bootloader interface source file.

```
#include "header.h"
```

Include dependency graph for Demo/ARMCM4_STM32L4_Nucleo_L476RG_IAR/Prog/boot.c:



Data Structures

- struct [tCanBusTiming](#)
Structure type for grouping CAN bus timing related information.

Macros

- `#define RS232_CTO_RX_PACKET_TIMEOUT_MS (100u)`
Timeout time for the reception of a CTO packet. The timer is started upon reception of the first packet byte.

Functions

- static void [BootComRs232Init](#) (void)
Initializes the UART communication interface.
- static void [BootComRs232CheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- static void [BootComCanInit](#) (void)
Initializes the CAN communication interface.
- static void [BootComCanCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootComInit](#) (void)
Initializes the communication interface.
- void [BootComCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootActivate](#) (void)
Bootloader activation function.
- static unsigned char [Rs232ReceiveByte](#) (unsigned char *data)
Receives a communication interface byte if one is present.
- static unsigned char [CanGetSpeedConfig](#) (unsigned short baud, unsigned short *prescaler, unsigned char *tseg1, unsigned char *tseg2)
Search algorithm to match the desired baudrate to a possible bus timing configuration.

Variables

- static UART_HandleTypeDef [rs232Handle](#)
UART handle to be used in API calls.
- static const [tCanBusTiming](#) [canTiming](#) []
CAN bittiming table for dynamically calculating the bittiming settings.
- static CAN_HandleTypeDef [canHandle](#)
CAN handle to be used in API calls.

7.214.1 Detailed Description

Demo program bootloader interface source file.

7.214.2 Function Documentation

7.214.2.1 BootActivate()

```
void BootActivate (  
    void )
```

Bootloader activation function.

Returns

none.

Referenced by [BootComCanCheckActivationRequest\(\)](#), and [BootComRs232CheckActivationRequest\(\)](#).

7.214.2.2 BootComCanCheckActivationRequest()

```
static void BootComCanCheckActivationRequest (  
    void ) [static]
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

Referenced by [BootComCheckActivationRequest\(\)](#).

7.214.2.3 BootComCanInit()

```
static void BootComCanInit (  
    void ) [static]
```

Initializes the CAN communication interface.

Returns

none.

Referenced by BootComInit().

7.214.2.4 BootComCheckActivationRequest()

```
void BootComCheckActivationRequest (  
    void )
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.214.2.5 BootComInit()

```
void BootComInit (  
    void )
```

Initializes the communication interface.

Returns

none.

7.214.2.6 BootComRs232CheckActivationRequest()

```
static void BootComRs232CheckActivationRequest (  
    void ) [static]
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

Referenced by BootComCheckActivationRequest().

7.214.2.7 BootComRs232Init()

```
static void BootComRs232Init (  
    void ) [static]
```

Initializes the UART communication interface.

Returns

none.

Referenced by BootComInit().

7.214.2.8 CanGetSpeedConfig()

```
static unsigned char CanGetSpeedConfig (  
    unsigned short baud,  
    unsigned short * prescaler,  
    unsigned char * tseg1,  
    unsigned char * tseg2 ) [static]
```

Search algorithm to match the desired baudrate to a possible bus timing configuration.

Parameters

<i>baud</i>	The desired baudrate in kbps. Valid values are 10..1000.
<i>prescaler</i>	Pointer to where the value for the prescaler will be stored.
<i>tseg1</i>	Pointer to where the value for TSEG2 will be stored.
<i>tseg2</i>	Pointer to where the value for TSEG2 will be stored.

Returns

1 if the CAN bustiming register values were found, 0 otherwise.

Referenced by BootComCanInit().

7.214.2.9 Rs232ReceiveByte()

```
static unsigned char Rs232ReceiveByte (  
    unsigned char * data ) [static]
```

Receives a communication interface byte if one is present.

Parameters

<i>data</i>	Pointer to byte where the data is to be stored.
-------------	---

Returns

1 if a byte was received, 0 otherwise.

Referenced by `BootComRs232CheckActivationRequest()`.

7.214.3 Variable Documentation**7.214.3.1 canTiming**

```
const tCanBusTiming canTiming[] [static]
```

Initial value:

```
=  
{  
    { 5, 2 },  
    { 6, 2 },  
    { 6, 3 },  
    { 7, 3 },  
    { 8, 3 },  
    { 9, 3 },  
    { 9, 4 },  
    { 10, 4 },  
    { 11, 4 },  
    { 12, 4 },  
    { 12, 5 },  
    { 13, 5 },  
    { 14, 5 },  
    { 15, 5 },  
    { 15, 6 },  
    { 16, 6 },  
    { 16, 7 },  
    { 16, 8 }  
}
```

CAN bittiming table for dynamically calculating the bittiming settings.

According to the CAN protocol 1 bit-time can be made up of between 8..25 time quanta (TQ). The total TQ in a bit is SYNC + TSEG1 + TSEG2 with SYNC always being 1. The sample point is (SYNC + TSEG1) / (SYNC + TSEG1 + SEG2) * 100%. This array contains possible and valid time quanta configurations with a sample point between 68..78%.

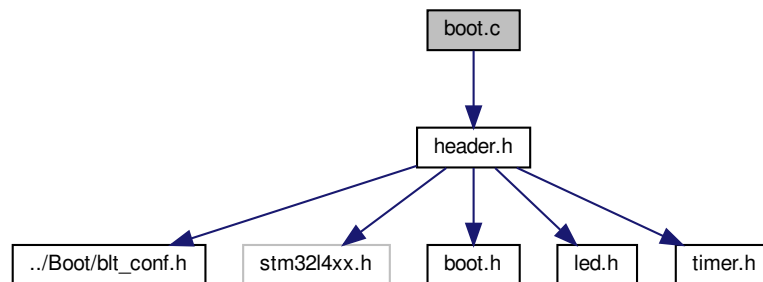
Referenced by `CanGetSpeedConfig()`.

7.215 boot.c File Reference

Demo program bootloader interface source file.

```
#include "header.h"
```

Include dependency graph for Demo/ARMCM4_STM32L4_Nucleo_L476RG_Keil/Prog/boot.c:



Data Structures

- struct [tCanBusTiming](#)
Structure type for grouping CAN bus timing related information.

Macros

- `#define RS232_CTO_RX_PACKET_TIMEOUT_MS (100u)`
Timeout time for the reception of a CTO packet. The timer is started upon reception of the first packet byte.

Functions

- static void [BootComRs232Init](#) (void)
Initializes the UART communication interface.
- static void [BootComRs232CheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- static void [BootComCanInit](#) (void)
Initializes the CAN communication interface.
- static void [BootComCanCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootComInit](#) (void)
Initializes the communication interface.
- void [BootComCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootActivate](#) (void)
Bootloader activation function.
- static unsigned char [Rs232ReceiveByte](#) (unsigned char *data)
Receives a communication interface byte if one is present.
- static unsigned char [CanGetSpeedConfig](#) (unsigned short baud, unsigned short *prescaler, unsigned char *tseg1, unsigned char *tseg2)
Search algorithm to match the desired baudrate to a possible bus timing configuration.

Variables

- static UART_HandleTypeDef [rs232Handle](#)
UART handle to be used in API calls.
- static const [tCanBusTiming](#) [canTiming](#) []
CAN bittiming table for dynamically calculating the bittiming settings.
- static CAN_HandleTypeDef [canHandle](#)
CAN handle to be used in API calls.

7.215.1 Detailed Description

Demo program bootloader interface source file.

7.215.2 Function Documentation

7.215.2.1 BootActivate()

```
void BootActivate (
    void )
```

Bootloader activation function.

Returns

none.

Referenced by [BootComCanCheckActivationRequest\(\)](#), and [BootComRs232CheckActivationRequest\(\)](#).

7.215.2.2 BootComCanCheckActivationRequest()

```
static void BootComCanCheckActivationRequest (
    void ) [static]
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

Referenced by [BootComCheckActivationRequest\(\)](#).

7.215.2.3 BootComCanInit()

```
static void BootComCanInit (  
    void ) [static]
```

Initializes the CAN communication interface.

Returns

none.

Referenced by BootComInit().

7.215.2.4 BootComCheckActivationRequest()

```
void BootComCheckActivationRequest (  
    void )
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.215.2.5 BootComInit()

```
void BootComInit (  
    void )
```

Initializes the communication interface.

Returns

none.

7.215.2.6 BootComRs232CheckActivationRequest()

```
static void BootComRs232CheckActivationRequest (  
    void ) [static]
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

Referenced by BootComCheckActivationRequest().

7.215.2.7 BootComRs232Init()

```
static void BootComRs232Init (  
    void ) [static]
```

Initializes the UART communication interface.

Returns

none.

Referenced by BootComInit().

7.215.2.8 CanGetSpeedConfig()

```
static unsigned char CanGetSpeedConfig (  
    unsigned short baud,  
    unsigned short * prescaler,  
    unsigned char * tseg1,  
    unsigned char * tseg2 ) [static]
```

Search algorithm to match the desired baudrate to a possible bus timing configuration.

Parameters

<i>baud</i>	The desired baudrate in kbps. Valid values are 10..1000.
<i>prescaler</i>	Pointer to where the value for the prescaler will be stored.
<i>tseg1</i>	Pointer to where the value for TSEG2 will be stored.
<i>tseg2</i>	Pointer to where the value for TSEG2 will be stored.

Returns

1 if the CAN bustiming register values were found, 0 otherwise.

Referenced by BootComCanInit().

7.215.2.9 Rs232ReceiveByte()

```
static unsigned char Rs232ReceiveByte (  
    unsigned char * data ) [static]
```

Receives a communication interface byte if one is present.

Parameters

<i>data</i>	Pointer to byte where the data is to be stored.
-------------	---

Macros

- `#define RS232_CTO_RX_PACKET_TIMEOUT_MS (100u)`
Timeout time for the reception of a CTO packet. The timer is started upon reception of the first packet byte.

Functions

- static void `BootComRs232Init` (void)
Initializes the UART communication interface.
- static void `BootComRs232CheckActivationRequest` (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void `BootComInit` (void)
Initializes the communication interface.
- void `BootComCheckActivationRequest` (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void `BootActivate` (void)
Bootloader activation function.
- static unsigned char `Rs232ReceiveByte` (unsigned char *data)
Receives a communication interface byte if one is present.

7.216.1 Detailed Description

Demo program bootloader interface source file.

7.216.2 Function Documentation

7.216.2.1 `BootActivate()`

```
void BootActivate (  
    void )
```

Bootloader activation function.

Returns

none.

Referenced by `BootComRs232CheckActivationRequest()`.

7.216.2.2 BootComCheckActivationRequest()

```
void BootComCheckActivationRequest (
    void )
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.216.2.3 BootComInit()

```
void BootComInit (
    void )
```

Initializes the communication interface.

Returns

none.

7.216.2.4 BootComRs232CheckActivationRequest()

```
static void BootComRs232CheckActivationRequest (
    void ) [static]
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

Referenced by BootComCheckActivationRequest().

7.216.2.5 BootComRs232Init()

```
static void BootComRs232Init (
    void ) [static]
```

Initializes the UART communication interface.

Returns

none.

Referenced by BootComInit().

7.216.2.6 Rs232ReceiveByte()

```
static unsigned char Rs232ReceiveByte (
    unsigned char * data ) [static]
```

Receives a communication interface byte if one is present.

Parameters

<i>data</i>	Pointer to byte where the data is to be stored.
-------------	---

Returns

1 if a byte was received, 0 otherwise.

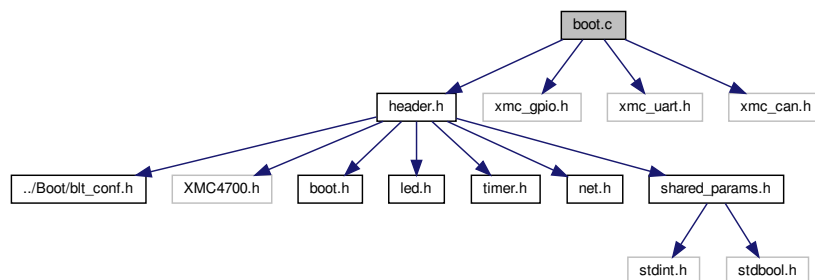
Referenced by `BootComRs232CheckActivationRequest()`.

7.217 boot.c File Reference

Demo program bootloader interface source file.

```
#include "header.h"
#include "xmc_gpio.h"
#include "xmc_uart.h"
#include "xmc_can.h"
```

Include dependency graph for Demo/ARMCM4_XMC4_XMC4700_Relax_Kit_GCC/Prog/boot.c:



Macros

- `#define RS232_CTO_RX_PACKET_TIMEOUT_MS (100u)`
Timeout time for the reception of a CTO packet. The timer is started upon reception of the first packet byte.

Functions

- static void `BootComRs232Init` (void)
Initializes the UART communication interface.
- static void `BootComRs232CheckActivationRequest` (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- static void `BootComCanInit` (void)
Initializes the CAN communication interface.
- static void `BootComCanCheckActivationRequest` (void)

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

- void [BootComInit](#) (void)

Initializes the communication interface.

- void [BootComCheckActivationRequest](#) (void)

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

- void [BootActivate](#) (void)

Bootloader activation function.

- static unsigned char [Rs232ReceiveByte](#) (unsigned char *data)

Receives a communication interface byte if one is present.

Variables

- static XMC_CAN_MO_t [receiveMsgObj](#)

Receive message object data structure.

7.217.1 Detailed Description

Demo program bootloader interface source file.

7.217.2 Function Documentation

7.217.2.1 BootActivate()

```
void BootActivate (
    void )
```

Bootloader activation function.

Returns

none.

Referenced by [BootComCanCheckActivationRequest\(\)](#), and [BootComRs232CheckActivationRequest\(\)](#).

7.217.2.2 BootComCanCheckActivationRequest()

```
static void BootComCanCheckActivationRequest (
    void ) [static]
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

Referenced by [BootComCheckActivationRequest\(\)](#).

7.217.2.3 BootComCanInit()

```
static void BootComCanInit (
    void ) [static]
```

Initializes the CAN communication interface.

Returns

none.

Referenced by BootComInit().

7.217.2.4 BootComCheckActivationRequest()

```
void BootComCheckActivationRequest (
    void )
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.217.2.5 BootComInit()

```
void BootComInit (
    void )
```

Initializes the communication interface.

Returns

none.

7.217.2.6 BootComRs232CheckActivationRequest()

```
static void BootComRs232CheckActivationRequest (
    void ) [static]
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

Referenced by BootComCheckActivationRequest().

7.217.2.7 BootComRs232Init()

```
static void BootComRs232Init (  
    void ) [static]
```

Initializes the UART communication interface.

Returns

none.

Referenced by BootComInit().

7.217.2.8 Rs232ReceiveByte()

```
static unsigned char Rs232ReceiveByte (  
    unsigned char * data ) [static]
```

Receives a communication interface byte if one is present.

Parameters

<i>data</i>	Pointer to byte where the data is to be stored.
-------------	---

Returns

1 if a byte was received, 0 otherwise.

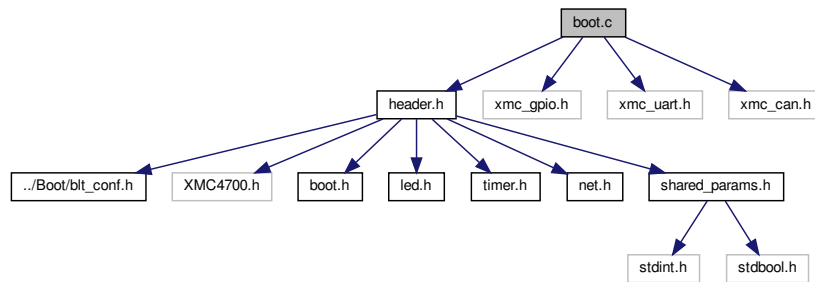
Referenced by BootComRs232CheckActivationRequest().

7.218 boot.c File Reference

Demo program bootloader interface source file.

```
#include "header.h"  
#include "xmc_gpio.h"  
#include "xmc_uart.h"  
#include "xmc_can.h"
```

Include dependency graph for Demo/ARMCM4_XMC4_XMC4700_Relax_Kit_IAR/Prog/boot.c:



Macros

- `#define RS232_CTO_RX_PACKET_TIMEOUT_MS (100u)`
Timeout time for the reception of a CTO packet. The timer is started upon reception of the first packet byte.

Functions

- static void `BootComRs232Init` (void)
Initializes the UART communication interface.
- static void `BootComRs232CheckActivationRequest` (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- static void `BootComCanInit` (void)
Initializes the CAN communication interface.
- static void `BootComCanCheckActivationRequest` (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void `BootComInit` (void)
Initializes the communication interface.
- void `BootComCheckActivationRequest` (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void `BootActivate` (void)
Bootloader activation function.
- static unsigned char `Rs232ReceiveByte` (unsigned char *data)
Receives a communication interface byte if one is present.

Variables

- static XMC_CAN_MO_t `receiveMsgObj`
Receive message object data structure.

7.218.1 Detailed Description

Demo program bootloader interface source file.

7.218.2 Function Documentation

7.218.2.1 BootActivate()

```
void BootActivate (
    void )
```

Bootloader activation function.

Returns

none.

Referenced by `BootComCanCheckActivationRequest()`, and `BootComRs232CheckActivationRequest()`.

7.218.2.2 BootComCanCheckActivationRequest()

```
static void BootComCanCheckActivationRequest (
    void ) [static]
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

Referenced by `BootComCheckActivationRequest()`.

7.218.2.3 BootComCanInit()

```
static void BootComCanInit (
    void ) [static]
```

Initializes the CAN communication interface.

Returns

none.

Referenced by `BootComInit()`.

7.218.2.4 BootComCheckActivationRequest()

```
void BootComCheckActivationRequest (
    void )
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.218.2.5 BootComInit()

```
void BootComInit (
    void )
```

Initializes the communication interface.

Returns

none.

7.218.2.6 BootComRs232CheckActivationRequest()

```
static void BootComRs232CheckActivationRequest (
    void ) [static]
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

Referenced by BootComCheckActivationRequest().

7.218.2.7 BootComRs232Init()

```
static void BootComRs232Init (
    void ) [static]
```

Initializes the UART communication interface.

Returns

none.

Referenced by BootComInit().

7.218.2.8 Rs232ReceiveByte()

```
static unsigned char Rs232ReceiveByte (
    unsigned char * data ) [static]
```

Receives a communication interface byte if one is present.

Parameters

<i>data</i>	Pointer to byte where the data is to be stored.
-------------	---

Returns

1 if a byte was received, 0 otherwise.

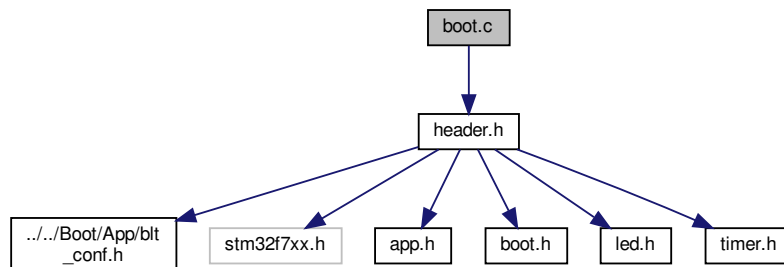
Referenced by `BootComRs232CheckActivationRequest()`.

7.219 boot.c File Reference

Demo program bootloader interface source file.

```
#include "header.h"
```

Include dependency graph for Demo/ARMCM7_STM32F7_Nucleo_F746ZG_CubeIDE/Prog/App/boot.c:



Data Structures

- struct [tCanBusTiming](#)

Structure type for grouping CAN bus timing related information.

Macros

- #define [RS232_CTO_RX_PACKET_TIMEOUT_MS](#) (100u)

Timeout time for the reception of a CTO packet. The timer is started upon reception of the first packet byte.

Functions

- static void [BootComRs232Init](#) (void)
Initializes the UART communication interface.
- static void [BootComRs232CheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- static void [BootComCanInit](#) (void)
Initializes the CAN communication interface.
- static void [BootComCanCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootComInit](#) (void)
Initializes the communication interface.
- void [BootComCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootActivate](#) (void)
Bootloader activation function.
- static unsigned char [Rs232ReceiveByte](#) (unsigned char *data)
Receives a communication interface byte if one is present.
- static unsigned char [CanGetSpeedConfig](#) (unsigned short baud, unsigned short *prescaler, unsigned char *tseg1, unsigned char *tseg2)
Search algorithm to match the desired baudrate to a possible bus timing configuration.

Variables

- static UART_HandleTypeDef [rs232Handle](#)
UART handle to be used in API calls.
- static const [tCanBusTiming](#) [canTiming](#) []
CAN bittiming table for dynamically calculating the bittiming settings.
- static CAN_HandleTypeDef [canHandle](#)
CAN handle to be used in API calls.

7.219.1 Detailed Description

Demo program bootloader interface source file.

7.219.2 Function Documentation

7.219.2.1 [BootActivate\(\)](#)

```
void BootActivate (
    void )
```

Bootloader activation function.

Returns

none.

Referenced by [BootComCanCheckActivationRequest\(\)](#), and [BootComRs232CheckActivationRequest\(\)](#).

7.219.2.2 BootComCanCheckActivationRequest()

```
static void BootComCanCheckActivationRequest (  
    void ) [static]
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

Referenced by BootComCheckActivationRequest().

7.219.2.3 BootComCanInit()

```
static void BootComCanInit (  
    void ) [static]
```

Initializes the CAN communication interface.

Returns

none.

Referenced by BootComInit().

7.219.2.4 BootComCheckActivationRequest()

```
void BootComCheckActivationRequest (  
    void )
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.219.2.5 BootComInit()

```
void BootComInit (
    void )
```

Initializes the communication interface.

Returns

none.

7.219.2.6 BootComRs232CheckActivationRequest()

```
static void BootComRs232CheckActivationRequest (
    void ) [static]
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

Referenced by BootComCheckActivationRequest().

7.219.2.7 BootComRs232Init()

```
static void BootComRs232Init (
    void ) [static]
```

Initializes the UART communication interface.

Returns

none.

Referenced by BootComInit().

7.219.2.8 CanGetSpeedConfig()

```
static unsigned char CanGetSpeedConfig (
    unsigned short baud,
    unsigned short * prescaler,
    unsigned char * tseg1,
    unsigned char * tseg2 ) [static]
```

Search algorithm to match the desired baudrate to a possible bus timing configuration.

Parameters

<i>baud</i>	The desired baudrate in kbps. Valid values are 10..1000.
<i>prescaler</i>	Pointer to where the value for the prescaler will be stored.
<i>tseg1</i>	Pointer to where the value for TSEG2 will be stored.
<i>tseg2</i>	Pointer to where the value for TSEG2 will be stored.

Returns

1 if the CAN bustiming register values were found, 0 otherwise.

Referenced by BootComCanInit().

7.219.2.9 Rs232ReceiveByte()

```
static unsigned char Rs232ReceiveByte (  
    unsigned char * data ) [static]
```

Receives a communication interface byte if one is present.

Parameters

<i>data</i>	Pointer to byte where the data is to be stored.
-------------	---

Returns

1 if a byte was received, 0 otherwise.

Referenced by BootComRs232CheckActivationRequest().

7.219.3 Variable Documentation**7.219.3.1 canTiming**

```
const tCanBusTiming canTiming[] [static]
```

Initial value:

```
=  
{  
    { 5, 2 },  
    { 6, 2 },  
    { 6, 3 },  
    { 7, 3 },  
    { 8, 3 },  
}
```

```

{ 9, 3 },
{ 9, 4 },
{ 10, 4 },
{ 11, 4 },
{ 12, 4 },
{ 12, 5 },
{ 13, 5 },
{ 14, 5 },
{ 15, 5 },
{ 15, 6 },
{ 16, 6 },
{ 16, 7 },
{ 16, 8 }
}

```

CAN bittiming table for dynamically calculating the bittiming settings.

According to the CAN protocol 1 bit-time can be made up of between 8..25 time quanta (TQ). The total TQ in a bit is SYNC + TSEG1 + TSEG2 with SYNC always being 1. The sample point is (SYNC + TSEG1) / (SYNC + TSEG1 + SEG2) * 100%. This array contains possible and valid time quanta configurations with a sample point between 68..78%.

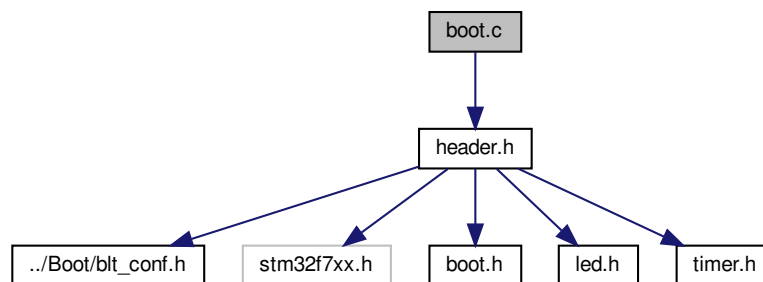
Referenced by CanGetSpeedConfig().

7.220 boot.c File Reference

Demo program bootloader interface source file.

```
#include "header.h"
```

Include dependency graph for Demo/ARMCM7_STM32F7_Nucleo_F746ZG_GCC/Prog/boot.c:



Data Structures

- struct [tCanBusTiming](#)
Structure type for grouping CAN bus timing related information.

Macros

- #define [RS232_CTO_RX_PACKET_TIMEOUT_MS](#) (100u)
Timeout time for the reception of a CTO packet. The timer is started upon reception of the first packet byte.

Functions

- static void [BootComRs232Init](#) (void)
Initializes the UART communication interface.
- static void [BootComRs232CheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- static void [BootComCanInit](#) (void)
Initializes the CAN communication interface.
- static void [BootComCanCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootComInit](#) (void)
Initializes the communication interface.
- void [BootComCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootActivate](#) (void)
Bootloader activation function.
- static unsigned char [Rs232ReceiveByte](#) (unsigned char *data)
Receives a communication interface byte if one is present.
- static unsigned char [CanGetSpeedConfig](#) (unsigned short baud, unsigned short *prescaler, unsigned char *tseg1, unsigned char *tseg2)
Search algorithm to match the desired baudrate to a possible bus timing configuration.

Variables

- static UART_HandleTypeDef [rs232Handle](#)
UART handle to be used in API calls.
- static const [tCanBusTiming](#) [canTiming](#) []
CAN bittiming table for dynamically calculating the bittiming settings.
- static CAN_HandleTypeDef [canHandle](#)
CAN handle to be used in API calls.

7.220.1 Detailed Description

Demo program bootloader interface source file.

7.220.2 Function Documentation

7.220.2.1 BootActivate()

```
void BootActivate (
    void )
```

Bootloader activation function.

Returns

none.

Referenced by [BootComCanCheckActivationRequest\(\)](#), and [BootComRs232CheckActivationRequest\(\)](#).

7.220.2.2 BootComCanCheckActivationRequest()

```
static void BootComCanCheckActivationRequest (  
    void ) [static]
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

Referenced by BootComCheckActivationRequest().

7.220.2.3 BootComCanInit()

```
static void BootComCanInit (  
    void ) [static]
```

Initializes the CAN communication interface.

Returns

none.

Referenced by BootComInit().

7.220.2.4 BootComCheckActivationRequest()

```
void BootComCheckActivationRequest (  
    void )
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.220.2.5 BootComInit()

```
void BootComInit (
    void )
```

Initializes the communication interface.

Returns

none.

7.220.2.6 BootComRs232CheckActivationRequest()

```
static void BootComRs232CheckActivationRequest (
    void ) [static]
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

Referenced by BootComCheckActivationRequest().

7.220.2.7 BootComRs232Init()

```
static void BootComRs232Init (
    void ) [static]
```

Initializes the UART communication interface.

Returns

none.

Referenced by BootComInit().

7.220.2.8 CanGetSpeedConfig()

```
static unsigned char CanGetSpeedConfig (
    unsigned short baud,
    unsigned short * prescaler,
    unsigned char * tseg1,
    unsigned char * tseg2 ) [static]
```

Search algorithm to match the desired baudrate to a possible bus timing configuration.

Parameters

<i>baud</i>	The desired baudrate in kbps. Valid values are 10..1000.
<i>prescaler</i>	Pointer to where the value for the prescaler will be stored.
<i>tseg1</i>	Pointer to where the value for TSEG2 will be stored.
<i>tseg2</i>	Pointer to where the value for TSEG2 will be stored.

Returns

1 if the CAN bustiming register values were found, 0 otherwise.

Referenced by BootComCanInit().

7.220.2.9 Rs232ReceiveByte()

```
static unsigned char Rs232ReceiveByte (
    unsigned char * data ) [static]
```

Receives a communication interface byte if one is present.

Parameters

<i>data</i>	Pointer to byte where the data is to be stored.
-------------	---

Returns

1 if a byte was received, 0 otherwise.

Referenced by BootComRs232CheckActivationRequest().

7.220.3 Variable Documentation**7.220.3.1 canTiming**

```
const tCanBusTiming canTiming[] [static]
```

Initial value:

```
=
{
    { 5, 2 },
    { 6, 2 },
    { 6, 3 },
    { 7, 3 },
    { 8, 3 },
```

```

    { 9, 3 },
    { 9, 4 },
    { 10, 4 },
    { 11, 4 },
    { 12, 4 },
    { 12, 5 },
    { 13, 5 },
    { 14, 5 },
    { 15, 5 },
    { 15, 6 },
    { 16, 6 },
    { 16, 7 },
    { 16, 8 }
}

```

CAN bittiming table for dynamically calculating the bittiming settings.

According to the CAN protocol 1 bit-time can be made up of between 8..25 time quanta (TQ). The total TQ in a bit is SYNC + TSEG1 + TSEG2 with SYNC always being 1. The sample point is (SYNC + TSEG1) / (SYNC + TSEG1 + SEG2) * 100%. This array contains possible and valid time quanta configurations with a sample point between 68..78%.

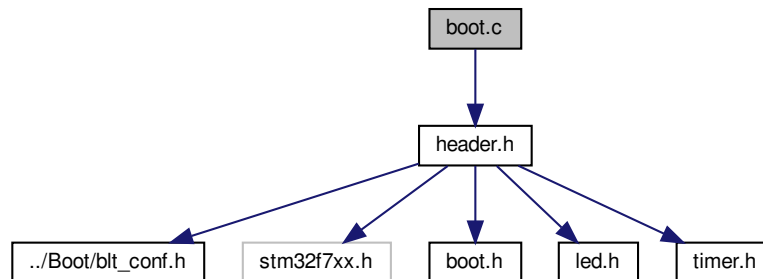
Referenced by CanGetSpeedConfig().

7.221 boot.c File Reference

Demo program bootloader interface source file.

```
#include "header.h"
```

Include dependency graph for Demo/ARMCM7_STM32F7_Nucleo_F746ZG_IAR/Prog/boot.c:



Data Structures

- struct [tCanBusTiming](#)
Structure type for grouping CAN bus timing related information.

Macros

- #define [RS232_CTO_RX_PACKET_TIMEOUT_MS](#) (100u)
Timeout time for the reception of a CTO packet. The timer is started upon reception of the first packet byte.

Functions

- static void [BootComRs232Init](#) (void)
Initializes the UART communication interface.
- static void [BootComRs232CheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- static void [BootComCanInit](#) (void)
Initializes the CAN communication interface.
- static void [BootComCanCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootComInit](#) (void)
Initializes the communication interface.
- void [BootComCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootActivate](#) (void)
Bootloader activation function.
- static unsigned char [Rs232ReceiveByte](#) (unsigned char *data)
Receives a communication interface byte if one is present.
- static unsigned char [CanGetSpeedConfig](#) (unsigned short baud, unsigned short *prescaler, unsigned char *tseg1, unsigned char *tseg2)
Search algorithm to match the desired baudrate to a possible bus timing configuration.

Variables

- static UART_HandleTypeDef [rs232Handle](#)
UART handle to be used in API calls.
- static const [tCanBusTiming](#) [canTiming](#) []
CAN bittiming table for dynamically calculating the bittiming settings.
- static CAN_HandleTypeDef [canHandle](#)
CAN handle to be used in API calls.

7.221.1 Detailed Description

Demo program bootloader interface source file.

7.221.2 Function Documentation

7.221.2.1 BootActivate()

```
void BootActivate (
    void )
```

Bootloader activation function.

Returns

none.

Referenced by [BootComCanCheckActivationRequest\(\)](#), and [BootComRs232CheckActivationRequest\(\)](#).

7.221.2.2 BootComCanCheckActivationRequest()

```
static void BootComCanCheckActivationRequest (  
    void ) [static]
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

Referenced by BootComCheckActivationRequest().

7.221.2.3 BootComCanInit()

```
static void BootComCanInit (  
    void ) [static]
```

Initializes the CAN communication interface.

Returns

none.

Referenced by BootComInit().

7.221.2.4 BootComCheckActivationRequest()

```
void BootComCheckActivationRequest (  
    void )
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.221.2.5 BootComInit()

```
void BootComInit (
    void )
```

Initializes the communication interface.

Returns

none.

7.221.2.6 BootComRs232CheckActivationRequest()

```
static void BootComRs232CheckActivationRequest (
    void ) [static]
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

Referenced by BootComCheckActivationRequest().

7.221.2.7 BootComRs232Init()

```
static void BootComRs232Init (
    void ) [static]
```

Initializes the UART communication interface.

Returns

none.

Referenced by BootComInit().

7.221.2.8 CanGetSpeedConfig()

```
static unsigned char CanGetSpeedConfig (
    unsigned short baud,
    unsigned short * prescaler,
    unsigned char * tseg1,
    unsigned char * tseg2 ) [static]
```

Search algorithm to match the desired baudrate to a possible bus timing configuration.

Parameters

<i>baud</i>	The desired baudrate in kbps. Valid values are 10..1000.
<i>prescaler</i>	Pointer to where the value for the prescaler will be stored.
<i>tseg1</i>	Pointer to where the value for TSEG2 will be stored.
<i>tseg2</i>	Pointer to where the value for TSEG2 will be stored.

Returns

1 if the CAN bustiming register values were found, 0 otherwise.

Referenced by BootComCanInit().

7.221.2.9 Rs232ReceiveByte()

```
static unsigned char Rs232ReceiveByte (  
    unsigned char * data ) [static]
```

Receives a communication interface byte if one is present.

Parameters

<i>data</i>	Pointer to byte where the data is to be stored.
-------------	---

Returns

1 if a byte was received, 0 otherwise.

Referenced by BootComRs232CheckActivationRequest().

7.221.3 Variable Documentation**7.221.3.1 canTiming**

```
const tCanBusTiming canTiming[] [static]
```

Initial value:

```
=  
{  
    { 5, 2 },  
    { 6, 2 },  
    { 6, 3 },  
    { 7, 3 },  
    { 8, 3 },  
}
```

```

    { 9, 3 },
    { 9, 4 },
    { 10, 4 },
    { 11, 4 },
    { 12, 4 },
    { 12, 5 },
    { 13, 5 },
    { 14, 5 },
    { 15, 5 },
    { 15, 6 },
    { 16, 6 },
    { 16, 7 },
    { 16, 8 }
}

```

CAN bittiming table for dynamically calculating the bittiming settings.

According to the CAN protocol 1 bit-time can be made up of between 8..25 time quanta (TQ). The total TQ in a bit is SYNC + TSEG1 + TSEG2 with SYNC always being 1. The sample point is (SYNC + TSEG1) / (SYNC + TSEG1 + SEG2) * 100%. This array contains possible and valid time quanta configurations with a sample point between 68..78%.

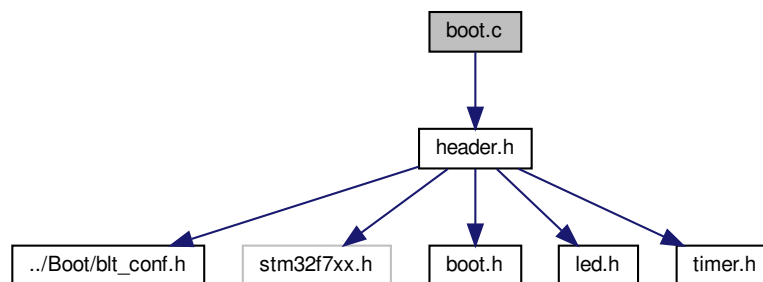
Referenced by CanGetSpeedConfig().

7.222 boot.c File Reference

Demo program bootloader interface source file.

```
#include "header.h"
```

Include dependency graph for Demo/ARMCM7_STM32F7_Nucleo_F746ZG_Keil/Prog/boot.c:



Data Structures

- struct [tCanBusTiming](#)
Structure type for grouping CAN bus timing related information.

Macros

- #define [RS232_CTO_RX_PACKET_TIMEOUT_MS](#) (100u)
Timeout time for the reception of a CTO packet. The timer is started upon reception of the first packet byte.

Functions

- static void [BootComRs232Init](#) (void)
Initializes the UART communication interface.
- static void [BootComRs232CheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- static void [BootComCanInit](#) (void)
Initializes the CAN communication interface.
- static void [BootComCanCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootComInit](#) (void)
Initializes the communication interface.
- void [BootComCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootActivate](#) (void)
Bootloader activation function.
- static unsigned char [Rs232ReceiveByte](#) (unsigned char *data)
Receives a communication interface byte if one is present.
- static unsigned char [CanGetSpeedConfig](#) (unsigned short baud, unsigned short *prescaler, unsigned char *tseg1, unsigned char *tseg2)
Search algorithm to match the desired baudrate to a possible bus timing configuration.

Variables

- static UART_HandleTypeDef [rs232Handle](#)
UART handle to be used in API calls.
- static const [tCanBusTiming](#) [canTiming](#) []
CAN bittiming table for dynamically calculating the bittiming settings.
- static CAN_HandleTypeDef [canHandle](#)
CAN handle to be used in API calls.

7.222.1 Detailed Description

Demo program bootloader interface source file.

7.222.2 Function Documentation

7.222.2.1 BootActivate()

```
void BootActivate (
    void )
```

Bootloader activation function.

Returns

none.

Referenced by [BootComCanCheckActivationRequest\(\)](#), and [BootComRs232CheckActivationRequest\(\)](#).

7.222.2.2 BootComCanCheckActivationRequest()

```
static void BootComCanCheckActivationRequest (  
    void ) [static]
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

Referenced by BootComCheckActivationRequest().

7.222.2.3 BootComCanInit()

```
static void BootComCanInit (  
    void ) [static]
```

Initializes the CAN communication interface.

Returns

none.

Referenced by BootComInit().

7.222.2.4 BootComCheckActivationRequest()

```
void BootComCheckActivationRequest (  
    void )
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.222.2.5 BootComInit()

```
void BootComInit (
    void )
```

Initializes the communication interface.

Returns

none.

7.222.2.6 BootComRs232CheckActivationRequest()

```
static void BootComRs232CheckActivationRequest (
    void ) [static]
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

Referenced by BootComCheckActivationRequest().

7.222.2.7 BootComRs232Init()

```
static void BootComRs232Init (
    void ) [static]
```

Initializes the UART communication interface.

Returns

none.

Referenced by BootComInit().

7.222.2.8 CanGetSpeedConfig()

```
static unsigned char CanGetSpeedConfig (
    unsigned short baud,
    unsigned short * prescaler,
    unsigned char * tseg1,
    unsigned char * tseg2 ) [static]
```

Search algorithm to match the desired baudrate to a possible bus timing configuration.

Parameters

<i>baud</i>	The desired baudrate in kbps. Valid values are 10..1000.
<i>prescaler</i>	Pointer to where the value for the prescaler will be stored.
<i>tseg1</i>	Pointer to where the value for TSEG2 will be stored.
<i>tseg2</i>	Pointer to where the value for TSEG2 will be stored.

Returns

1 if the CAN bustiming register values were found, 0 otherwise.

Referenced by BootComCanInit().

7.222.2.9 Rs232ReceiveByte()

```
static unsigned char Rs232ReceiveByte (
    unsigned char * data ) [static]
```

Receives a communication interface byte if one is present.

Parameters

<i>data</i>	Pointer to byte where the data is to be stored.
-------------	---

Returns

1 if a byte was received, 0 otherwise.

Referenced by BootComRs232CheckActivationRequest().

7.222.3 Variable Documentation**7.222.3.1 canTiming**

```
const tCanBusTiming canTiming[] [static]
```

Initial value:

```
=
{
    { 5, 2 },
    { 6, 2 },
    { 6, 3 },
    { 7, 3 },
    { 8, 3 },
```

```

{ 9, 3 },
{ 9, 4 },
{ 10, 4 },
{ 11, 4 },
{ 12, 4 },
{ 12, 5 },
{ 13, 5 },
{ 14, 5 },
{ 15, 5 },
{ 15, 6 },
{ 16, 6 },
{ 16, 7 },
{ 16, 8 }
}

```

CAN bittiming table for dynamically calculating the bittiming settings.

According to the CAN protocol 1 bit-time can be made up of between 8..25 time quanta (TQ). The total TQ in a bit is SYNC + TSEG1 + TSEG2 with SYNC always being 1. The sample point is (SYNC + TSEG1) / (SYNC + TSEG1 + SEG2) * 100%. This array contains possible and valid time quanta configurations with a sample point between 68..78%.

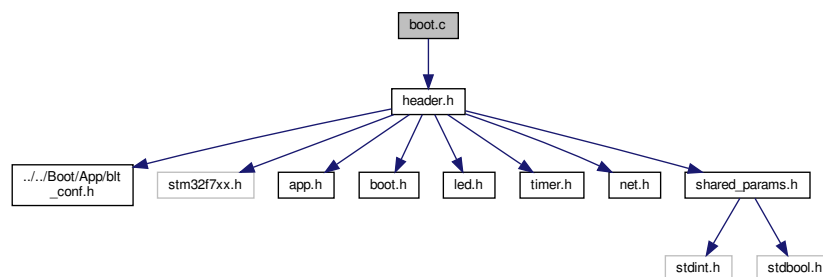
Referenced by CanGetSpeedConfig().

7.223 boot.c File Reference

Demo program bootloader interface source file.

```
#include "header.h"
```

Include dependency graph for Demo/ARMCM7_STM32F7_Nucleo_F767ZI_CubeIDE/Prog/App/boot.c:



Macros

- `#define RS232_CTO_RX_PACKET_TIMEOUT_MS (100u)`

Timeout time for the reception of a CTO packet. The timer is started upon reception of the first packet byte.

Functions

- static void [BootComRs232Init](#) (void)
Initializes the UART communication interface.
- static void [BootComRs232CheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootComInit](#) (void)
Initializes the communication interface.
- void [BootComCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootActivate](#) (void)
Bootloader activation function.
- static unsigned char [Rs232ReceiveByte](#) (unsigned char *data)
Receives a communication interface byte if one is present.

Variables

- static UART_HandleTypeDef [rs232Handle](#)
UART handle to be used in API calls.

7.223.1 Detailed Description

Demo program bootloader interface source file.

7.223.2 Function Documentation

7.223.2.1 BootActivate()

```
void BootActivate (  
    void )
```

Bootloader activation function.

Returns

none.

Referenced by [BootComRs232CheckActivationRequest\(\)](#).

7.223.2.2 BootComCheckActivationRequest()

```
void BootComCheckActivationRequest (  
    void )
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.223.2.3 BootComInit()

```
void BootComInit (  
    void )
```

Initializes the communication interface.

Returns

none.

7.223.2.4 BootComRs232CheckActivationRequest()

```
static void BootComRs232CheckActivationRequest (  
    void ) [static]
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

Referenced by BootComCheckActivationRequest().

7.223.2.5 BootComRs232Init()

```
static void BootComRs232Init (  
    void ) [static]
```

Initializes the UART communication interface.

Returns

none.

Referenced by BootComInit().

7.223.2.6 Rs232ReceiveByte()

```
static unsigned char Rs232ReceiveByte (  
    unsigned char * data ) [static]
```

Receives a communication interface byte if one is present.

Parameters

<i>data</i>	Pointer to byte where the data is to be stored.
-------------	---

Returns

1 if a byte was received, 0 otherwise.

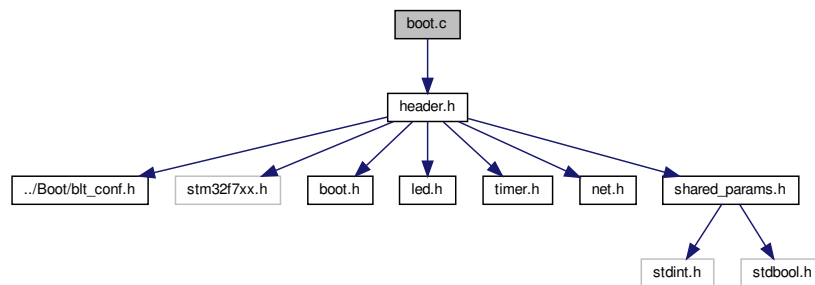
Referenced by `BootComRs232CheckActivationRequest()`.

7.224 boot.c File Reference

Demo program bootloader interface source file.

```
#include "header.h"
```

Include dependency graph for Demo/ARMCM7_STM32F7_Nucleo_F767ZI_GCC/Prog/boot.c:



Macros

- `#define RS232_CTO_RX_PACKET_TIMEOUT_MS (100u)`
Timeout time for the reception of a CTO packet. The timer is started upon reception of the first packet byte.

Functions

- static void `BootComRs232Init` (void)
Initializes the UART communication interface.
- static void `BootComRs232CheckActivationRequest` (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void `BootComInit` (void)
Initializes the communication interface.
- void `BootComCheckActivationRequest` (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void `BootActivate` (void)
Bootloader activation function.
- static unsigned char `Rs232ReceiveByte` (unsigned char *data)
Receives a communication interface byte if one is present.

Variables

- static UART_HandleTypeDef [rs232Handle](#)
UART handle to be used in API calls.

7.224.1 Detailed Description

Demo program bootloader interface source file.

7.224.2 Function Documentation

7.224.2.1 BootActivate()

```
void BootActivate (
    void )
```

Bootloader activation function.

Returns

none.

Referenced by `BootComRs232CheckActivationRequest()`.

7.224.2.2 BootComCheckActivationRequest()

```
void BootComCheckActivationRequest (
    void )
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.224.2.3 BootComInit()

```
void BootComInit (
    void )
```

Initializes the communication interface.

Returns

none.

7.224.2.4 BootComRs232CheckActivationRequest()

```
static void BootComRs232CheckActivationRequest (
    void ) [static]
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

Referenced by BootComCheckActivationRequest().

7.224.2.5 BootComRs232Init()

```
static void BootComRs232Init (
    void ) [static]
```

Initializes the UART communication interface.

Returns

none.

Referenced by BootComInit().

7.224.2.6 Rs232ReceiveByte()

```
static unsigned char Rs232ReceiveByte (
    unsigned char * data ) [static]
```

Receives a communication interface byte if one is present.

Parameters

<i>data</i>	Pointer to byte where the data is to be stored.
-------------	---

Returns

1 if a byte was received, 0 otherwise.

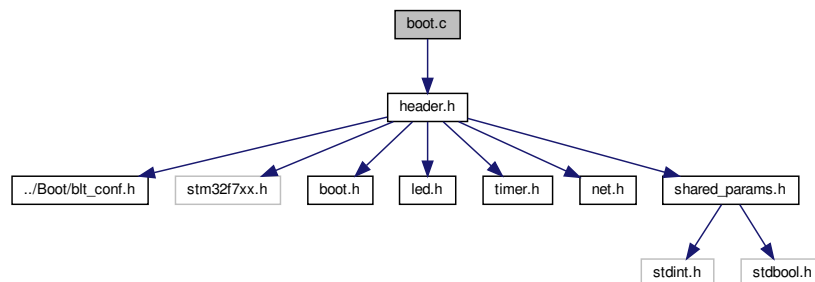
Referenced by `BootComRs232CheckActivationRequest()`.

7.225 boot.c File Reference

Demo program bootloader interface source file.

```
#include "header.h"
```

Include dependency graph for Demo/ARMCM7_STM32F7_Nucleo_F767ZI_IAR/Prog/boot.c:



Macros

- `#define RS232_CTO_RX_PACKET_TIMEOUT_MS (100u)`
Timeout time for the reception of a CTO packet. The timer is started upon reception of the first packet byte.

Functions

- static void `BootComRs232Init` (void)
Initializes the UART communication interface.
- static void `BootComRs232CheckActivationRequest` (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void `BootComInit` (void)
Initializes the communication interface.
- void `BootComCheckActivationRequest` (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void `BootActivate` (void)
Bootloader activation function.
- static unsigned char `Rs232ReceiveByte` (unsigned char *data)
Receives a communication interface byte if one is present.

Variables

- static UART_HandleTypeDef [rs232Handle](#)
UART handle to be used in API calls.

7.225.1 Detailed Description

Demo program bootloader interface source file.

7.225.2 Function Documentation

7.225.2.1 BootActivate()

```
void BootActivate (  
    void )
```

Bootloader activation function.

Returns

none.

Referenced by BootComRs232CheckActivationRequest().

7.225.2.2 BootComCheckActivationRequest()

```
void BootComCheckActivationRequest (  
    void )
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.225.2.3 BootComInit()

```
void BootComInit (
    void )
```

Initializes the communication interface.

Returns

none.

7.225.2.4 BootComRs232CheckActivationRequest()

```
static void BootComRs232CheckActivationRequest (
    void ) [static]
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

Referenced by BootComCheckActivationRequest().

7.225.2.5 BootComRs232Init()

```
static void BootComRs232Init (
    void ) [static]
```

Initializes the UART communication interface.

Returns

none.

Referenced by BootComInit().

7.225.2.6 Rs232ReceiveByte()

```
static unsigned char Rs232ReceiveByte (
    unsigned char * data ) [static]
```

Receives a communication interface byte if one is present.

Parameters

<i>data</i>	Pointer to byte where the data is to be stored.
-------------	---

Returns

1 if a byte was received, 0 otherwise.

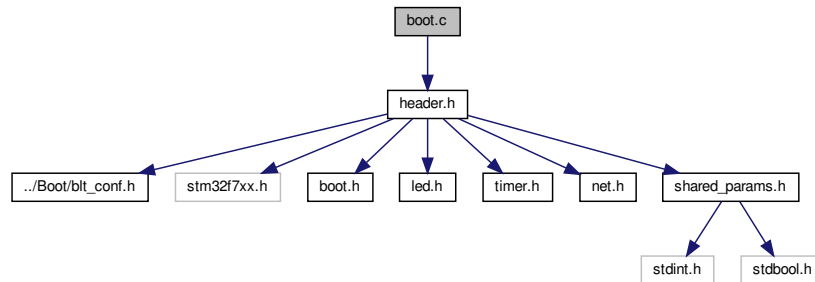
Referenced by `BootComRs232CheckActivationRequest()`.

7.226 boot.c File Reference

Demo program bootloader interface source file.

```
#include "header.h"
```

Include dependency graph for Demo/ARMCM7_STM32F7_Nucleo_F767ZI_Keil/Prog/boot.c:



Macros

- `#define RS232_CTO_RX_PACKET_TIMEOUT_MS (100u)`
Timeout time for the reception of a CTO packet. The timer is started upon reception of the first packet byte.

Functions

- static void `BootComRs232Init` (void)
Initializes the UART communication interface.
- static void `BootComRs232CheckActivationRequest` (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void `BootComInit` (void)
Initializes the communication interface.
- void `BootComCheckActivationRequest` (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void `BootActivate` (void)
Bootloader activation function.
- static unsigned char `Rs232ReceiveByte` (unsigned char *data)
Receives a communication interface byte if one is present.

Variables

- static UART_HandleTypeDef [rs232Handle](#)
UART handle to be used in API calls.

7.226.1 Detailed Description

Demo program bootloader interface source file.

7.226.2 Function Documentation

7.226.2.1 BootActivate()

```
void BootActivate (
    void )
```

Bootloader activation function.

Returns

none.

Referenced by `BootComRs232CheckActivationRequest()`.

7.226.2.2 BootComCheckActivationRequest()

```
void BootComCheckActivationRequest (
    void )
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.226.2.3 BootComInit()

```
void BootComInit (
    void )
```

Initializes the communication interface.

Returns

none.

7.226.2.4 BootComRs232CheckActivationRequest()

```
static void BootComRs232CheckActivationRequest (
    void ) [static]
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

Referenced by BootComCheckActivationRequest().

7.226.2.5 BootComRs232Init()

```
static void BootComRs232Init (
    void ) [static]
```

Initializes the UART communication interface.

Returns

none.

Referenced by BootComInit().

7.226.2.6 Rs232ReceiveByte()

```
static unsigned char Rs232ReceiveByte (
    unsigned char * data ) [static]
```

Receives a communication interface byte if one is present.

Parameters

<code>data</code>	Pointer to byte where the data is to be stored.
-------------------	---

Returns

1 if a byte was received, 0 otherwise.

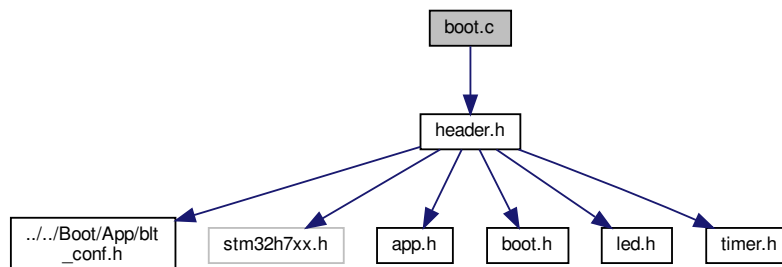
Referenced by `BootComRs232CheckActivationRequest()`.

7.227 boot.c File Reference

Demo program bootloader interface source file.

```
#include "header.h"
```

Include dependency graph for Demo/ARMCM7_STM32H7_Nucleo_H743ZI_CubeIDE/Prog/App/boot.c:



Data Structures

- struct [tCanBusTiming](#)

Structure type for grouping CAN bus timing related information.

Macros

- #define [RS232_CTO_RX_PACKET_TIMEOUT_MS](#) (100u)

Timeout time for the reception of a CTO packet. The timer is started upon reception of the first packet byte.

Functions

- static void [BootComRs232Init](#) (void)
Initializes the UART communication interface.
- static void [BootComRs232CheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- static void [BootComCanInit](#) (void)
Initializes the CAN communication interface.
- static void [BootComCanCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootComInit](#) (void)
Initializes the communication interface.
- void [BootComCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootActivate](#) (void)
Bootloader activation function.
- static unsigned char [Rs232ReceiveByte](#) (unsigned char *data)
Receives a communication interface byte if one is present.
- static unsigned char [CanGetSpeedConfig](#) (unsigned short baud, unsigned short *prescaler, unsigned char *tseg1, unsigned char *tseg2)
Search algorithm to match the desired baudrate to a possible bus timing configuration.

Variables

- static UART_HandleTypeDef [rs232Handle](#)
UART handle to be used in API calls.
- static const [tCanBusTiming](#) [canTiming](#) []
CAN bittiming table for dynamically calculating the bittiming settings.
- static FDCAN_HandleTypeDef [canHandle](#)
CAN handle to be used in API calls.

7.227.1 Detailed Description

Demo program bootloader interface source file.

7.227.2 Function Documentation

7.227.2.1 BootActivate()

```
void BootActivate (
    void )
```

Bootloader activation function.

Returns

none.

Referenced by [BootComCanCheckActivationRequest\(\)](#), and [BootComRs232CheckActivationRequest\(\)](#).

7.227.2.2 BootComCanCheckActivationRequest()

```
static void BootComCanCheckActivationRequest (  
    void ) [static]
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

Referenced by BootComCheckActivationRequest().

7.227.2.3 BootComCanInit()

```
static void BootComCanInit (  
    void ) [static]
```

Initializes the CAN communication interface.

Returns

none.

Referenced by BootComInit().

7.227.2.4 BootComCheckActivationRequest()

```
void BootComCheckActivationRequest (  
    void )
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.227.2.5 BootComInit()

```
void BootComInit (
    void )
```

Initializes the communication interface.

Returns

none.

7.227.2.6 BootComRs232CheckActivationRequest()

```
static void BootComRs232CheckActivationRequest (
    void ) [static]
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

Referenced by BootComCheckActivationRequest().

7.227.2.7 BootComRs232Init()

```
static void BootComRs232Init (
    void ) [static]
```

Initializes the UART communication interface.

Returns

none.

Referenced by BootComInit().

7.227.2.8 CanGetSpeedConfig()

```
static unsigned char CanGetSpeedConfig (
    unsigned short baud,
    unsigned short * prescaler,
    unsigned char * tseg1,
    unsigned char * tseg2 ) [static]
```

Search algorithm to match the desired baudrate to a possible bus timing configuration.

Parameters

<i>baud</i>	The desired baudrate in kbps. Valid values are 10..1000.
<i>prescaler</i>	Pointer to where the value for the prescaler will be stored.
<i>tseg1</i>	Pointer to where the value for TSEG2 will be stored.
<i>tseg2</i>	Pointer to where the value for TSEG2 will be stored.

Returns

1 if the CAN bustiming register values were found, 0 otherwise.

Referenced by BootComCanInit().

7.227.2.9 Rs232ReceiveByte()

```
static unsigned char Rs232ReceiveByte (  
    unsigned char * data ) [static]
```

Receives a communication interface byte if one is present.

Parameters

<i>data</i>	Pointer to byte where the data is to be stored.
-------------	---

Returns

1 if a byte was received, 0 otherwise.

Referenced by BootComRs232CheckActivationRequest().

7.227.3 Variable Documentation**7.227.3.1 canTiming**

```
const tCanBusTiming canTiming[] [static]
```

Initial value:

```
=  
{  
    { 5, 2 },  
    { 6, 2 },  
    { 6, 3 },  
    { 7, 3 },  
    { 8, 3 },  
}
```

```

{ 9, 3 },
{ 9, 4 },
{ 10, 4 },
{ 11, 4 },
{ 12, 4 },
{ 12, 5 },
{ 13, 5 },
{ 14, 5 },
{ 15, 5 },
{ 15, 6 },
{ 16, 6 },
{ 16, 7 },
{ 16, 8 }
}

```

CAN bittiming table for dynamically calculating the bittiming settings.

According to the CAN protocol 1 bit-time can be made up of between 8..25 time quanta (TQ). The total TQ in a bit is SYNC + TSEG1 + TSEG2 with SYNC always being 1. The sample point is (SYNC + TSEG1) / (SYNC + TSEG1 + SEG2) * 100%. This array contains possible and valid time quanta configurations with a sample point between 68..78%.

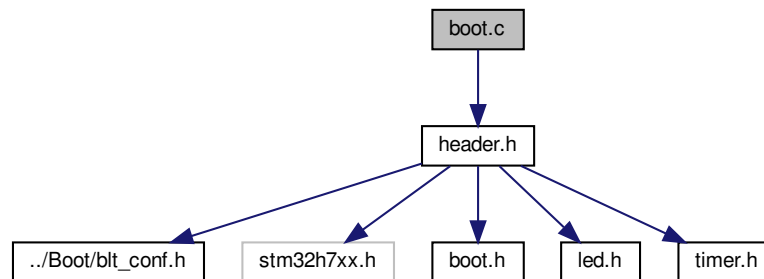
Referenced by CanGetSpeedConfig().

7.228 boot.c File Reference

Demo program bootloader interface source file.

```
#include "header.h"
```

Include dependency graph for Demo/ARMCM7_STM32H7_Nucleo_H743ZI_GCC/Prog/boot.c:



Data Structures

- struct [tCanBusTiming](#)
Structure type for grouping CAN bus timing related information.

Macros

- #define [RS232_CTO_RX_PACKET_TIMEOUT_MS](#) (100u)
Timeout time for the reception of a CTO packet. The timer is started upon reception of the first packet byte.

Functions

- static void [BootComRs232Init](#) (void)
Initializes the UART communication interface.
- static void [BootComRs232CheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- static void [BootComCanInit](#) (void)
Initializes the CAN communication interface.
- static void [BootComCanCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootComInit](#) (void)
Initializes the communication interface.
- void [BootComCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootActivate](#) (void)
Bootloader activation function.
- static unsigned char [Rs232ReceiveByte](#) (unsigned char *data)
Receives a communication interface byte if one is present.
- static unsigned char [CanGetSpeedConfig](#) (unsigned short baud, unsigned short *prescaler, unsigned char *tseg1, unsigned char *tseg2)
Search algorithm to match the desired baudrate to a possible bus timing configuration.

Variables

- static UART_HandleTypeDef [rs232Handle](#)
UART handle to be used in API calls.
- static const [tCanBusTiming](#) [canTiming](#) []
CAN bittiming table for dynamically calculating the bittiming settings.
- static FDCAN_HandleTypeDef [canHandle](#)
CAN handle to be used in API calls.

7.228.1 Detailed Description

Demo program bootloader interface source file.

7.228.2 Function Documentation

7.228.2.1 [BootActivate\(\)](#)

```
void BootActivate (
    void )
```

Bootloader activation function.

Returns

none.

Referenced by [BootComCanCheckActivationRequest\(\)](#), and [BootComRs232CheckActivationRequest\(\)](#).

7.228.2.2 BootComCanCheckActivationRequest()

```
static void BootComCanCheckActivationRequest (  
    void ) [static]
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

Referenced by BootComCheckActivationRequest().

7.228.2.3 BootComCanInit()

```
static void BootComCanInit (  
    void ) [static]
```

Initializes the CAN communication interface.

Returns

none.

Referenced by BootComInit().

7.228.2.4 BootComCheckActivationRequest()

```
void BootComCheckActivationRequest (  
    void )
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.228.2.5 BootComInit()

```
void BootComInit (
    void )
```

Initializes the communication interface.

Returns

none.

7.228.2.6 BootComRs232CheckActivationRequest()

```
static void BootComRs232CheckActivationRequest (
    void ) [static]
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

Referenced by BootComCheckActivationRequest().

7.228.2.7 BootComRs232Init()

```
static void BootComRs232Init (
    void ) [static]
```

Initializes the UART communication interface.

Returns

none.

Referenced by BootComInit().

7.228.2.8 CanGetSpeedConfig()

```
static unsigned char CanGetSpeedConfig (
    unsigned short baud,
    unsigned short * prescaler,
    unsigned char * tseg1,
    unsigned char * tseg2 ) [static]
```

Search algorithm to match the desired baudrate to a possible bus timing configuration.

Parameters

<i>baud</i>	The desired baudrate in kbps. Valid values are 10..1000.
<i>prescaler</i>	Pointer to where the value for the prescaler will be stored.
<i>tseg1</i>	Pointer to where the value for TSEG2 will be stored.
<i>tseg2</i>	Pointer to where the value for TSEG2 will be stored.

Returns

1 if the CAN bustiming register values were found, 0 otherwise.

Referenced by BootComCanInit().

7.228.2.9 Rs232ReceiveByte()

```
static unsigned char Rs232ReceiveByte (
    unsigned char * data ) [static]
```

Receives a communication interface byte if one is present.

Parameters

<i>data</i>	Pointer to byte where the data is to be stored.
-------------	---

Returns

1 if a byte was received, 0 otherwise.

Referenced by BootComRs232CheckActivationRequest().

7.228.3 Variable Documentation**7.228.3.1 canTiming**

```
const tCanBusTiming canTiming[] [static]
```

Initial value:

```
=
{
    { 5, 2 },
    { 6, 2 },
    { 6, 3 },
    { 7, 3 },
    { 8, 3 },
```

```

{ 9, 3 },
{ 9, 4 },
{ 10, 4 },
{ 11, 4 },
{ 12, 4 },
{ 12, 5 },
{ 13, 5 },
{ 14, 5 },
{ 15, 5 },
{ 15, 6 },
{ 16, 6 },
{ 16, 7 },
{ 16, 8 }
}

```

CAN bittiming table for dynamically calculating the bittiming settings.

According to the CAN protocol 1 bit-time can be made up of between 8..25 time quanta (TQ). The total TQ in a bit is SYNC + TSEG1 + TSEG2 with SYNC always being 1. The sample point is (SYNC + TSEG1) / (SYNC + TSEG1 + SEG2) * 100%. This array contains possible and valid time quanta configurations with a sample point between 68..78%.

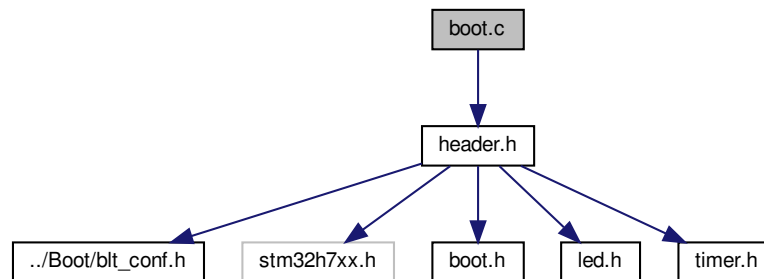
Referenced by CanGetSpeedConfig().

7.229 boot.c File Reference

Demo program bootloader interface source file.

```
#include "header.h"
```

Include dependency graph for Demo/ARMCM7_STM32H7_Nucleo_H743ZI_IAR/Prog/boot.c:



Data Structures

- struct [tCanBusTiming](#)
Structure type for grouping CAN bus timing related information.

Macros

- #define [RS232_CTO_RX_PACKET_TIMEOUT_MS](#) (100u)
Timeout time for the reception of a CTO packet. The timer is started upon reception of the first packet byte.

Functions

- static void [BootComRs232Init](#) (void)
Initializes the UART communication interface.
- static void [BootComRs232CheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- static void [BootComCanInit](#) (void)
Initializes the CAN communication interface.
- static void [BootComCanCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootComInit](#) (void)
Initializes the communication interface.
- void [BootComCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootActivate](#) (void)
Bootloader activation function.
- static unsigned char [Rs232ReceiveByte](#) (unsigned char *data)
Receives a communication interface byte if one is present.
- static unsigned char [CanGetSpeedConfig](#) (unsigned short baud, unsigned short *prescaler, unsigned char *tseg1, unsigned char *tseg2)
Search algorithm to match the desired baudrate to a possible bus timing configuration.

Variables

- static UART_HandleTypeDef [rs232Handle](#)
UART handle to be used in API calls.
- static const [tCanBusTiming](#) [canTiming](#) []
CAN bittiming table for dynamically calculating the bittiming settings.
- static FDCAN_HandleTypeDef [canHandle](#)
CAN handle to be used in API calls.

7.229.1 Detailed Description

Demo program bootloader interface source file.

7.229.2 Function Documentation

7.229.2.1 BootActivate()

```
void BootActivate (
    void )
```

Bootloader activation function.

Returns

none.

Referenced by [BootComCanCheckActivationRequest\(\)](#), and [BootComRs232CheckActivationRequest\(\)](#).

7.229.2.2 BootComCanCheckActivationRequest()

```
static void BootComCanCheckActivationRequest (  
    void ) [static]
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

Referenced by BootComCheckActivationRequest().

7.229.2.3 BootComCanInit()

```
static void BootComCanInit (  
    void ) [static]
```

Initializes the CAN communication interface.

Returns

none.

Referenced by BootComInit().

7.229.2.4 BootComCheckActivationRequest()

```
void BootComCheckActivationRequest (  
    void )
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.229.2.5 BootComInit()

```
void BootComInit (
    void )
```

Initializes the communication interface.

Returns

none.

7.229.2.6 BootComRs232CheckActivationRequest()

```
static void BootComRs232CheckActivationRequest (
    void ) [static]
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

Referenced by BootComCheckActivationRequest().

7.229.2.7 BootComRs232Init()

```
static void BootComRs232Init (
    void ) [static]
```

Initializes the UART communication interface.

Returns

none.

Referenced by BootComInit().

7.229.2.8 CanGetSpeedConfig()

```
static unsigned char CanGetSpeedConfig (
    unsigned short baud,
    unsigned short * prescaler,
    unsigned char * tseg1,
    unsigned char * tseg2 ) [static]
```

Search algorithm to match the desired baudrate to a possible bus timing configuration.

Parameters

<i>baud</i>	The desired baudrate in kbps. Valid values are 10..1000.
<i>prescaler</i>	Pointer to where the value for the prescaler will be stored.
<i>tseg1</i>	Pointer to where the value for TSEG2 will be stored.
<i>tseg2</i>	Pointer to where the value for TSEG2 will be stored.

Returns

1 if the CAN bustiming register values were found, 0 otherwise.

Referenced by BootComCanInit().

7.229.2.9 Rs232ReceiveByte()

```
static unsigned char Rs232ReceiveByte (  
    unsigned char * data ) [static]
```

Receives a communication interface byte if one is present.

Parameters

<i>data</i>	Pointer to byte where the data is to be stored.
-------------	---

Returns

1 if a byte was received, 0 otherwise.

Referenced by BootComRs232CheckActivationRequest().

7.229.3 Variable Documentation**7.229.3.1 canTiming**

```
const tCanBusTiming canTiming[] [static]
```

Initial value:

```
=  
{  
    { 5, 2 },  
    { 6, 2 },  
    { 6, 3 },  
    { 7, 3 },  
    { 8, 3 },  
}
```

```

{ 9, 3 },
{ 9, 4 },
{ 10, 4 },
{ 11, 4 },
{ 12, 4 },
{ 12, 5 },
{ 13, 5 },
{ 14, 5 },
{ 15, 5 },
{ 15, 6 },
{ 16, 6 },
{ 16, 7 },
{ 16, 8 }
}

```

CAN bittiming table for dynamically calculating the bittiming settings.

According to the CAN protocol 1 bit-time can be made up of between 8..25 time quanta (TQ). The total TQ in a bit is SYNC + TSEG1 + TSEG2 with SYNC always being 1. The sample point is (SYNC + TSEG1) / (SYNC + TSEG1 + SEG2) * 100%. This array contains possible and valid time quanta configurations with a sample point between 68..78%.

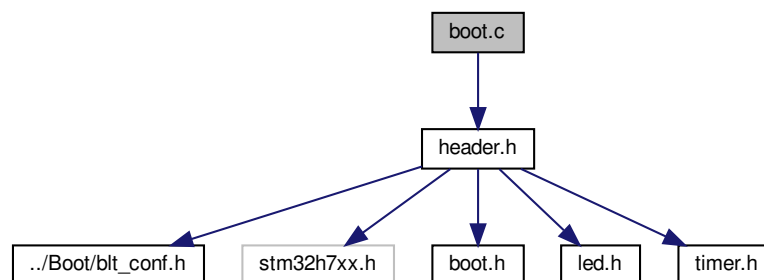
Referenced by CanGetSpeedConfig().

7.230 boot.c File Reference

Demo program bootloader interface source file.

```
#include "header.h"
```

Include dependency graph for Demo/ARMCM7_STM32H7_Nucleo_H743ZI_Keil/Prog/boot.c:



Data Structures

- struct [tCanBusTiming](#)
Structure type for grouping CAN bus timing related information.

Macros

- #define [RS232_CTO_RX_PACKET_TIMEOUT_MS](#) (100u)
Timeout time for the reception of a CTO packet. The timer is started upon reception of the first packet byte.

Functions

- static void [BootComRs232Init](#) (void)
Initializes the UART communication interface.
- static void [BootComRs232CheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- static void [BootComCanInit](#) (void)
Initializes the CAN communication interface.
- static void [BootComCanCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootComInit](#) (void)
Initializes the communication interface.
- void [BootComCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootActivate](#) (void)
Bootloader activation function.
- static unsigned char [Rs232ReceiveByte](#) (unsigned char *data)
Receives a communication interface byte if one is present.
- static unsigned char [CanGetSpeedConfig](#) (unsigned short baud, unsigned short *prescaler, unsigned char *tseg1, unsigned char *tseg2)
Search algorithm to match the desired baudrate to a possible bus timing configuration.

Variables

- static UART_HandleTypeDef [rs232Handle](#)
UART handle to be used in API calls.
- static const [tCanBusTiming](#) [canTiming](#) []
CAN bittiming table for dynamically calculating the bittiming settings.
- static FDCAN_HandleTypeDef [canHandle](#)
CAN handle to be used in API calls.

7.230.1 Detailed Description

Demo program bootloader interface source file.

7.230.2 Function Documentation

7.230.2.1 BootActivate()

```
void BootActivate (
    void )
```

Bootloader activation function.

Returns

none.

Referenced by [BootComCanCheckActivationRequest\(\)](#), and [BootComRs232CheckActivationRequest\(\)](#).

7.230.2.2 BootComCanCheckActivationRequest()

```
static void BootComCanCheckActivationRequest (  
    void ) [static]
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

Referenced by BootComCheckActivationRequest().

7.230.2.3 BootComCanInit()

```
static void BootComCanInit (  
    void ) [static]
```

Initializes the CAN communication interface.

Returns

none.

Referenced by BootComInit().

7.230.2.4 BootComCheckActivationRequest()

```
void BootComCheckActivationRequest (  
    void )
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.230.2.5 BootComInit()

```
void BootComInit (
    void )
```

Initializes the communication interface.

Returns

none.

7.230.2.6 BootComRs232CheckActivationRequest()

```
static void BootComRs232CheckActivationRequest (
    void ) [static]
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

Referenced by BootComCheckActivationRequest().

7.230.2.7 BootComRs232Init()

```
static void BootComRs232Init (
    void ) [static]
```

Initializes the UART communication interface.

Returns

none.

Referenced by BootComInit().

7.230.2.8 CanGetSpeedConfig()

```
static unsigned char CanGetSpeedConfig (
    unsigned short baud,
    unsigned short * prescaler,
    unsigned char * tseg1,
    unsigned char * tseg2 ) [static]
```

Search algorithm to match the desired baudrate to a possible bus timing configuration.

Parameters

<i>baud</i>	The desired baudrate in kbps. Valid values are 10..1000.
<i>prescaler</i>	Pointer to where the value for the prescaler will be stored.
<i>tseg1</i>	Pointer to where the value for TSEG2 will be stored.
<i>tseg2</i>	Pointer to where the value for TSEG2 will be stored.

Returns

1 if the CAN bustiming register values were found, 0 otherwise.

Referenced by BootComCanInit().

7.230.2.9 Rs232ReceiveByte()

```
static unsigned char Rs232ReceiveByte (
    unsigned char * data ) [static]
```

Receives a communication interface byte if one is present.

Parameters

<i>data</i>	Pointer to byte where the data is to be stored.
-------------	---

Returns

1 if a byte was received, 0 otherwise.

Referenced by BootComRs232CheckActivationRequest().

7.230.3 Variable Documentation**7.230.3.1 canTiming**

```
const tCanBusTiming canTiming[] [static]
```

Initial value:

```
=
{
    { 5, 2 },
    { 6, 2 },
    { 6, 3 },
    { 7, 3 },
    { 8, 3 },
```

```

{ 9, 3 },
{ 9, 4 },
{ 10, 4 },
{ 11, 4 },
{ 12, 4 },
{ 12, 5 },
{ 13, 5 },
{ 14, 5 },
{ 15, 5 },
{ 15, 6 },
{ 16, 6 },
{ 16, 7 },
{ 16, 8 }
}

```

CAN bittiming table for dynamically calculating the bittiming settings.

According to the CAN protocol 1 bit-time can be made up of between 8..25 time quanta (TQ). The total TQ in a bit is SYNC + TSEG1 + TSEG2 with SYNC always being 1. The sample point is (SYNC + TSEG1) / (SYNC + TSEG1 + SEG2) * 100%. This array contains possible and valid time quanta configurations with a sample point between 68..78%.

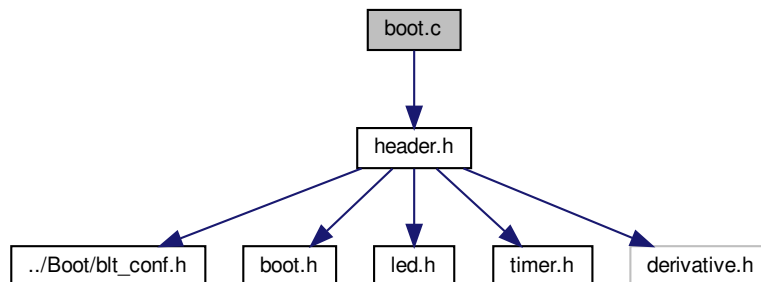
Referenced by CanGetSpeedConfig().

7.231 boot.c File Reference

Demo program bootloader interface source file.

```
#include "header.h"
```

Include dependency graph for Demo/HCS12_Evbplus_Dragon12p_CodeWarrior/Prog/boot.c:



Data Structures

- struct [tCanBusTiming](#)
Structure type for grouping CAN bus timing related information.

Macros

- #define [RS232_CTO_RX_PACKET_TIMEOUT_MS](#) (100u)
Timeout time for the reception of a CTO packet. The timer is started upon reception of the first packet byte.

Functions

- static void [BootComRs232Init](#) (void)
Initializes the UART communication interface.
- static void [BootComRs232CheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- static void [BootComCanInit](#) (void)
Initializes the CAN communication interface.
- static void [BootComCanCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootComInit](#) (void)
Initializes the communication interface.
- void [BootComCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootActivate](#) (void)
Bootloader activation function.
- static unsigned char [Rs232ReceiveByte](#) (unsigned char *data)
Receives a communication interface byte if one is present.
- static unsigned char [CanGetSpeedConfig](#) (unsigned short baud, unsigned char *btr0, unsigned char *btr1)
Search algorithm to match the desired baudrate to a possible bus timing configuration.

Variables

- static const [tCanBusTiming](#) [canTiming](#) []
Array with possible time quanta configurations.

7.231.1 Detailed Description

Demo program bootloader interface source file.

7.231.2 Function Documentation

7.231.2.1 [BootActivate\(\)](#)

```
void BootActivate (
    void )
```

Bootloader activation function.

Returns

none.

Referenced by [BootComCanCheckActivationRequest\(\)](#), and [BootComRs232CheckActivationRequest\(\)](#).

7.231.2.2 BootComCanCheckActivationRequest()

```
static void BootComCanCheckActivationRequest (  
    void ) [static]
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

Referenced by BootComCheckActivationRequest().

7.231.2.3 BootComCanInit()

```
static void BootComCanInit (  
    void ) [static]
```

Initializes the CAN communication interface.

Returns

none.

Referenced by BootComInit().

7.231.2.4 BootComCheckActivationRequest()

```
void BootComCheckActivationRequest (  
    void )
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.231.2.5 BootComInit()

```
void BootComInit (
    void )
```

Initializes the communication interface.

Returns

none.

7.231.2.6 BootComRs232CheckActivationRequest()

```
static void BootComRs232CheckActivationRequest (
    void ) [static]
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

Referenced by BootComCheckActivationRequest().

7.231.2.7 BootComRs232Init()

```
static void BootComRs232Init (
    void ) [static]
```

Initializes the UART communication interface.

Returns

none.

Referenced by BootComInit().

7.231.2.8 CanGetSpeedConfig()

```
static unsigned char CanGetSpeedConfig (
    unsigned short baud,
    unsigned char * btr0,
    unsigned char * btr1 ) [static]
```

Search algorithm to match the desired baudrate to a possible bus timing configuration.

Parameters

<i>baud</i>	The desired baudrate in kbps. Valid values are 10..1000.
<i>btr0</i>	Pointer to where the value for register CANxBTR0 will be stored.
<i>btr1</i>	Pointer to where the value for register CANxBTR1 will be stored.

Returns

1 if the CAN bustiming register values were found, 0 otherwise.

Referenced by BootComCanInit().

7.231.2.9 Rs232ReceiveByte()

```
static unsigned char Rs232ReceiveByte (  
    unsigned char * data ) [static]
```

Receives a communication interface byte if one is present.

Parameters

<i>data</i>	Pointer to byte where the data is to be stored.
-------------	---

Returns

1 if a byte was received, 0 otherwise.

Referenced by BootComRs232CheckActivationRequest().

7.231.3 Variable Documentation

7.231.3.1 canTiming

```
const tCanBusTiming canTiming[] [static]
```

Initial value:

```
=  
{  
    { 5, 2 },  
    { 6, 2 },  
    { 6, 3 },  
    { 7, 3 },  
    { 8, 3 },  
    { 9, 3 },  
}
```

```

{ 9, 4 },
{ 10, 4 },
{ 11, 4 },
{ 12, 4 },
{ 12, 5 },
{ 13, 5 },
{ 14, 5 },
{ 15, 5 },
{ 15, 6 },
{ 16, 6 },
{ 16, 7 },
{ 16, 8 }
}

```

Array with possible time quanta configurations.

According to the CAN protocol 1 bit-time can be made up of between 8..25 time quanta (TQ). The total TQ in a bit is SYNC + TSEG1 + TSEG2 with SYNC always being 1. The sample point is (SYNC + TSEG1) / (SYNC + TSEG1 + SEG2)

- 100%. This array contains possible and valid time quanta configurations with a sample point between 68..78%.

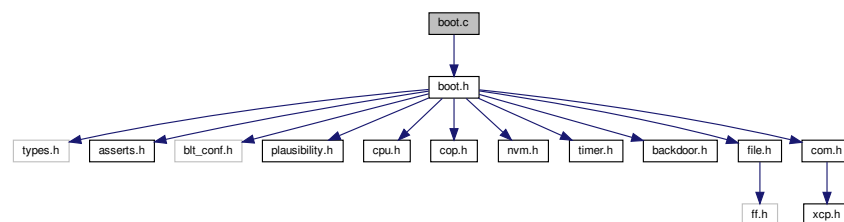
Referenced by CanGetSpeedConfig().

7.232 boot.c File Reference

Bootloader core module source file.

```
#include "boot.h"
```

Include dependency graph for Source/boot.c:



Functions

- void **BootInit** (void)
Initializes the bootloader core.
- void **BootTask** (void)
Task function of the bootloader core that drives the program.

7.232.1 Detailed Description

Bootloader core module source file.

7.232.2 Function Documentation

7.232.2.1 BootInit()

```
void BootInit (
    void )
```

Initializes the bootloader core.

Returns

none

Referenced by AppInit(), and main().

7.232.2.2 BootTask()

```
void BootTask (
    void )
```

Task function of the bootloader core that drives the program.

Returns

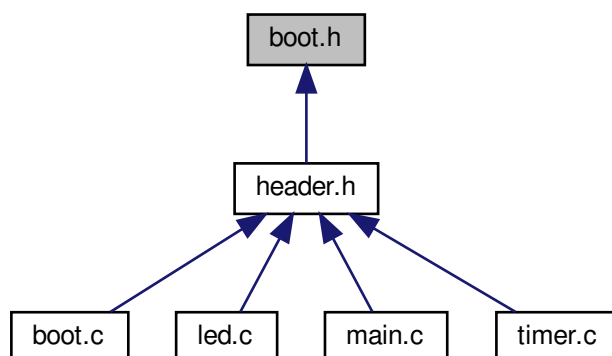
none

Referenced by AppTask(), and main().

7.233 boot.h File Reference

Demo program bootloader interface header file.

This graph shows which files directly or indirectly include this file:



Functions

- void [BootComInit](#) (void)
Initializes the communication interface.
- void [BootComCheckActivationRequest](#) (void)
Receives the `CONNECT` request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootActivate](#) (void)
Bootloader activation function.

7.233.1 Detailed Description

Demo program bootloader interface header file.

7.233.2 Function Documentation

7.233.2.1 [BootActivate\(\)](#)

```
void BootActivate (  
    void )
```

Bootloader activation function.

Returns

none.

7.233.2.2 [BootComCheckActivationRequest\(\)](#)

```
void BootComCheckActivationRequest (  
    void )
```

Receives the `CONNECT` request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.233.2.3 BootComInit()

```
void BootComInit (
    void )
```

Initializes the communication interface.

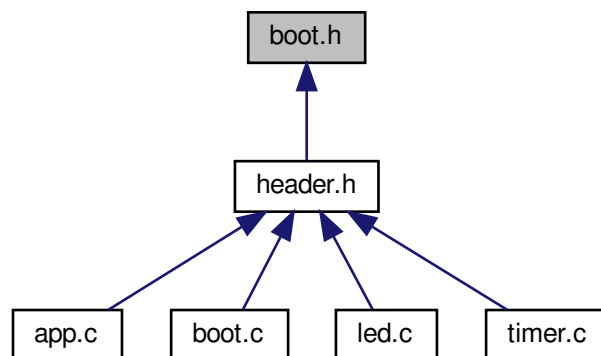
Returns

none.

7.234 boot.h File Reference

Demo program bootloader interface header file.

This graph shows which files directly or indirectly include this file:



Functions

- void [BootComInit](#) (void)
Initializes the communication interface.
- void [BootComCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootActivate](#) (void)
Bootloader activation function.

7.234.1 Detailed Description

Demo program bootloader interface header file.

7.234.2 Function Documentation

7.234.2.1 BootActivate()

```
void BootActivate (
    void )
```

Bootloader activation function.

Returns

none.

7.234.2.2 BootComCheckActivationRequest()

```
void BootComCheckActivationRequest (
    void )
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.234.2.3 BootComInit()

```
void BootComInit (
    void )
```

Initializes the communication interface.

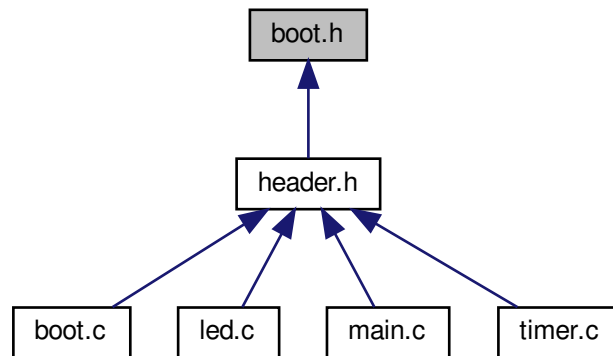
Returns

none.

7.235 boot.h File Reference

Demo program bootloader interface header file.

This graph shows which files directly or indirectly include this file:



Functions

- void [BootComInit](#) (void)
Initializes the communication interface.
- void [BootComCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootActivate](#) (void)
Bootloader activation function.

7.235.1 Detailed Description

Demo program bootloader interface header file.

7.235.2 Function Documentation

7.235.2.1 BootActivate()

```
void BootActivate (  
    void )
```

Bootloader activation function.

Returns

none.

7.235.2.2 BootComCheckActivationRequest()

```
void BootComCheckActivationRequest (
    void )
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.235.2.3 BootComInit()

```
void BootComInit (
    void )
```

Initializes the communication interface.

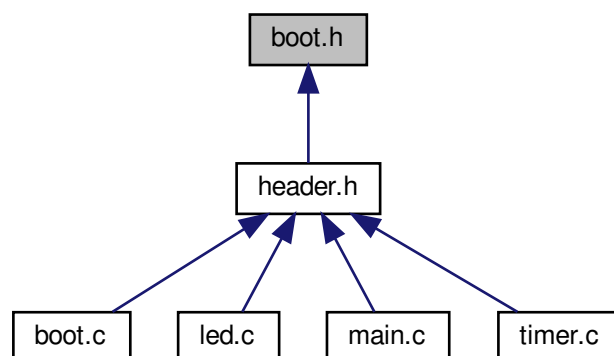
Returns

none.

7.236 boot.h File Reference

Demo program bootloader interface header file.

This graph shows which files directly or indirectly include this file:



Functions

- void [BootComInit](#) (void)
Initializes the communication interface.
- void [BootComCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootActivate](#) (void)
Bootloader activation function.

7.236.1 Detailed Description

Demo program bootloader interface header file.

7.236.2 Function Documentation

7.236.2.1 [BootActivate\(\)](#)

```
void BootActivate (  
    void )
```

Bootloader activation function.

Returns

none.

7.236.2.2 [BootComCheckActivationRequest\(\)](#)

```
void BootComCheckActivationRequest (  
    void )
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.236.2.3 BootComInit()

```
void BootComInit (
    void )
```

Initializes the communication interface.

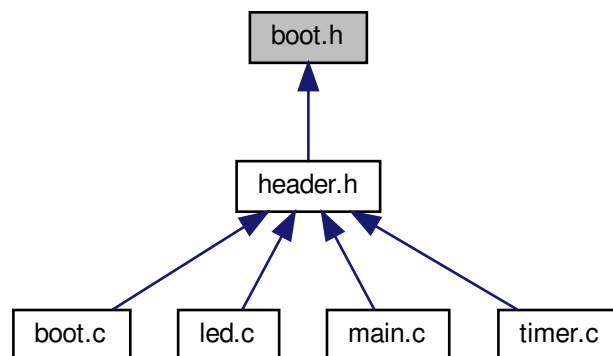
Returns

none.

7.237 boot.h File Reference

Demo program bootloader interface header file.

This graph shows which files directly or indirectly include this file:



Functions

- void [BootComInit](#) (void)
Initializes the communication interface.
- void [BootComCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootActivate](#) (void)
Bootloader activation function.

7.237.1 Detailed Description

Demo program bootloader interface header file.

7.237.2 Function Documentation

7.237.2.1 BootActivate()

```
void BootActivate (
    void )
```

Bootloader activation function.

Returns

none.

7.237.2.2 BootComCheckActivationRequest()

```
void BootComCheckActivationRequest (
    void )
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.237.2.3 BootComInit()

```
void BootComInit (
    void )
```

Initializes the communication interface.

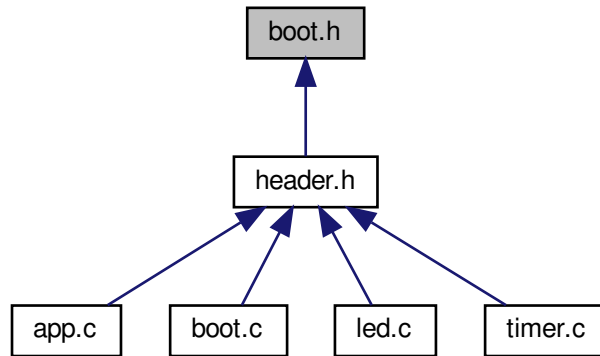
Returns

none.

7.238 boot.h File Reference

Demo program bootloader interface header file.

This graph shows which files directly or indirectly include this file:



Functions

- void [BootComInit](#) (void)
Initializes the communication interface.
- void [BootComCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootActivate](#) (void)
Bootloader activation function.

7.238.1 Detailed Description

Demo program bootloader interface header file.

7.238.2 Function Documentation

7.238.2.1 BootActivate()

```
void BootActivate (  
    void )
```

Bootloader activation function.

Returns

none.

7.238.2.2 BootComCheckActivationRequest()

```
void BootComCheckActivationRequest (
    void )
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.238.2.3 BootComInit()

```
void BootComInit (
    void )
```

Initializes the communication interface.

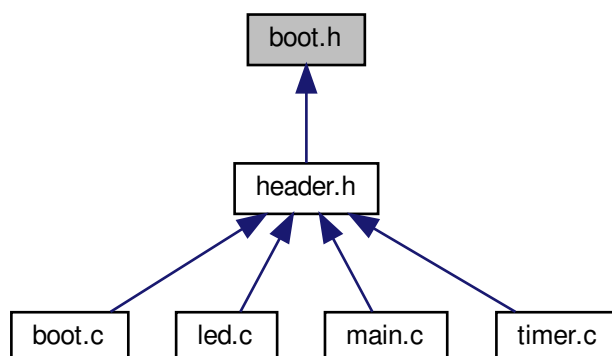
Returns

none.

7.239 boot.h File Reference

Demo program bootloader interface header file.

This graph shows which files directly or indirectly include this file:



Functions

- void [BootComInit](#) (void)
Initializes the communication interface.
- void [BootComCheckActivationRequest](#) (void)
Receives the `CONNECT` request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootActivate](#) (void)
Bootloader activation function.

7.239.1 Detailed Description

Demo program bootloader interface header file.

7.239.2 Function Documentation

7.239.2.1 [BootActivate\(\)](#)

```
void BootActivate (  
    void )
```

Bootloader activation function.

Returns

none.

7.239.2.2 [BootComCheckActivationRequest\(\)](#)

```
void BootComCheckActivationRequest (  
    void )
```

Receives the `CONNECT` request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.239.2.3 BootComInit()

```
void BootComInit (
    void )
```

Initializes the communication interface.

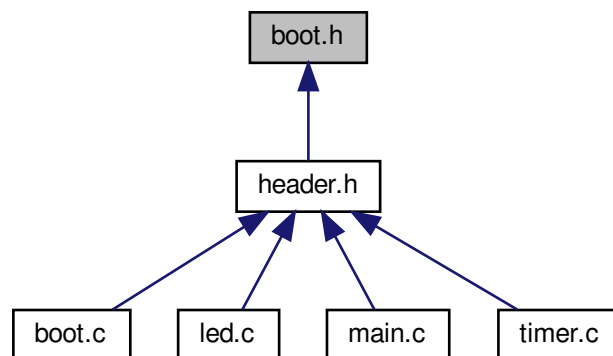
Returns

none.

7.240 boot.h File Reference

Demo program bootloader interface header file.

This graph shows which files directly or indirectly include this file:



Functions

- void [BootComInit](#) (void)
Initializes the communication interface.
- void [BootComCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootActivate](#) (void)
Bootloader activation function.

7.240.1 Detailed Description

Demo program bootloader interface header file.

7.240.2 Function Documentation

7.240.2.1 BootActivate()

```
void BootActivate (
    void )
```

Bootloader activation function.

Returns

none.

7.240.2.2 BootComCheckActivationRequest()

```
void BootComCheckActivationRequest (
    void )
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.240.2.3 BootComInit()

```
void BootComInit (
    void )
```

Initializes the communication interface.

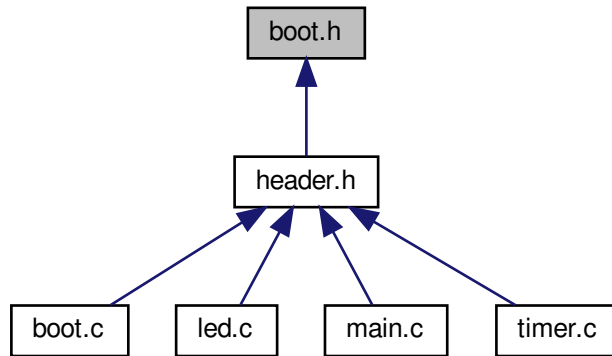
Returns

none.

7.241 boot.h File Reference

Demo program bootloader interface header file.

This graph shows which files directly or indirectly include this file:



Functions

- void [BootComInit](#) (void)
Initializes the communication interface.
- void [BootComCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootActivate](#) (void)
Bootloader activation function.

7.241.1 Detailed Description

Demo program bootloader interface header file.

7.241.2 Function Documentation

7.241.2.1 BootActivate()

```
void BootActivate (  
    void )
```

Bootloader activation function.

Returns

none.

7.241.2.2 BootComCheckActivationRequest()

```
void BootComCheckActivationRequest (
    void )
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.241.2.3 BootComInit()

```
void BootComInit (
    void )
```

Initializes the communication interface.

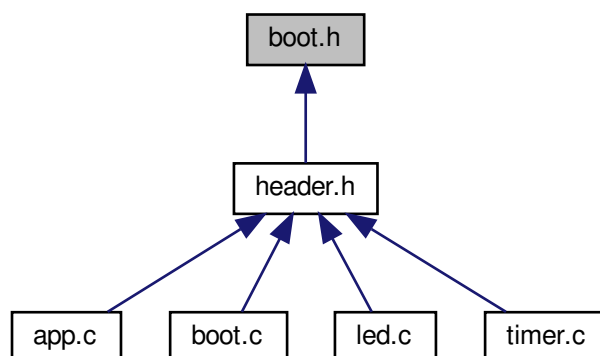
Returns

none.

7.242 boot.h File Reference

Demo program bootloader interface header file.

This graph shows which files directly or indirectly include this file:



Functions

- void [BootComInit](#) (void)
Initializes the communication interface.
- void [BootComCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootActivate](#) (void)
Bootloader activation function.

7.242.1 Detailed Description

Demo program bootloader interface header file.

7.242.2 Function Documentation

7.242.2.1 [BootActivate\(\)](#)

```
void BootActivate (  
    void )
```

Bootloader activation function.

Returns

none.

7.242.2.2 [BootComCheckActivationRequest\(\)](#)

```
void BootComCheckActivationRequest (  
    void )
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.242.2.3 BootComInit()

```
void BootComInit (
    void )
```

Initializes the communication interface.

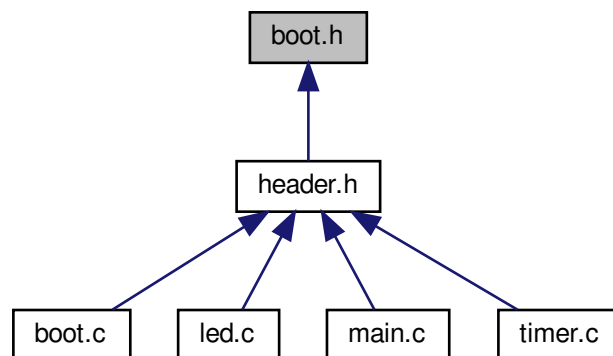
Returns

none.

7.243 boot.h File Reference

Demo program bootloader interface header file.

This graph shows which files directly or indirectly include this file:



Functions

- void [BootComInit](#) (void)
Initializes the communication interface.
- void [BootComCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootActivate](#) (void)
Bootloader activation function.

7.243.1 Detailed Description

Demo program bootloader interface header file.

7.243.2 Function Documentation

7.243.2.1 BootActivate()

```
void BootActivate (
    void )
```

Bootloader activation function.

Returns

none.

7.243.2.2 BootComCheckActivationRequest()

```
void BootComCheckActivationRequest (
    void )
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.243.2.3 BootComInit()

```
void BootComInit (
    void )
```

Initializes the communication interface.

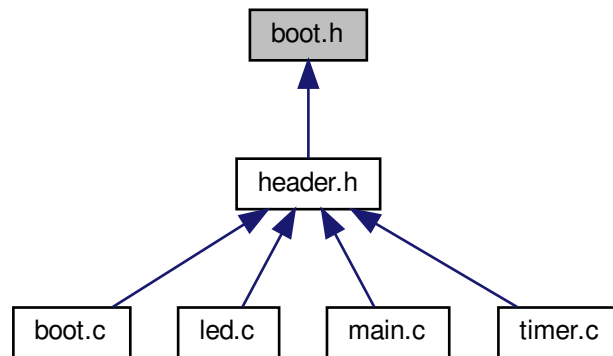
Returns

none.

7.244 boot.h File Reference

Demo program bootloader interface header file.

This graph shows which files directly or indirectly include this file:



Functions

- void [BootComInit](#) (void)
Initializes the communication interface.
- void [BootComCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootActivate](#) (void)
Bootloader activation function.

7.244.1 Detailed Description

Demo program bootloader interface header file.

7.244.2 Function Documentation

7.244.2.1 BootActivate()

```
void BootActivate (  
    void )
```

Bootloader activation function.

Returns

none.

7.244.2.2 BootComCheckActivationRequest()

```
void BootComCheckActivationRequest (  
    void )
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.244.2.3 BootComInit()

```
void BootComInit (  
    void )
```

Initializes the communication interface.

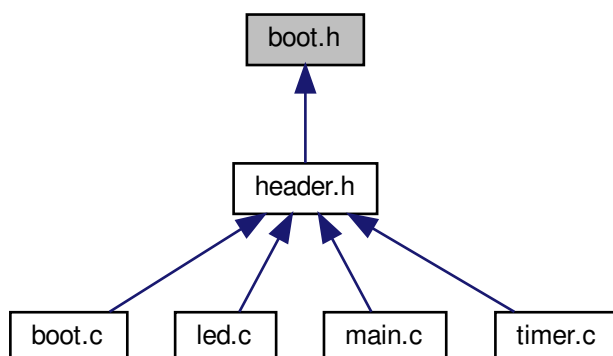
Returns

none.

7.245 boot.h File Reference

Demo program bootloader interface header file.

This graph shows which files directly or indirectly include this file:



Functions

- void [BootComInit](#) (void)
Initializes the communication interface.
- void [BootComCheckActivationRequest](#) (void)
Receives the `CONNECT` request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootActivate](#) (void)
Bootloader activation function.

7.245.1 Detailed Description

Demo program bootloader interface header file.

7.245.2 Function Documentation

7.245.2.1 [BootActivate\(\)](#)

```
void BootActivate (  
    void )
```

Bootloader activation function.

Returns

none.

7.245.2.2 [BootComCheckActivationRequest\(\)](#)

```
void BootComCheckActivationRequest (  
    void )
```

Receives the `CONNECT` request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.245.2.3 BootComInit()

```
void BootComInit (
    void )
```

Initializes the communication interface.

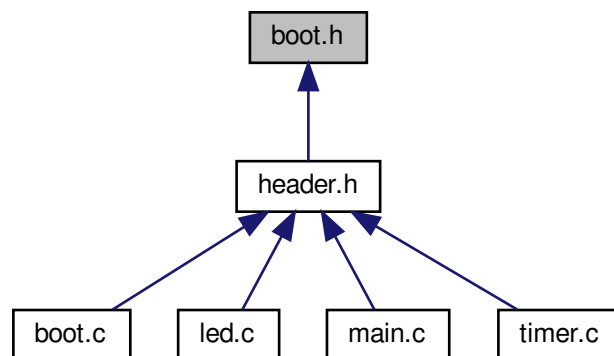
Returns

none.

7.246 boot.h File Reference

Demo program bootloader interface header file.

This graph shows which files directly or indirectly include this file:



Functions

- void [BootComInit](#) (void)
Initializes the communication interface.
- void [BootComCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootActivate](#) (void)
Bootloader activation function.

7.246.1 Detailed Description

Demo program bootloader interface header file.

7.246.2 Function Documentation

7.246.2.1 BootActivate()

```
void BootActivate (
    void )
```

Bootloader activation function.

Returns

none.

7.246.2.2 BootComCheckActivationRequest()

```
void BootComCheckActivationRequest (
    void )
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.246.2.3 BootComInit()

```
void BootComInit (
    void )
```

Initializes the communication interface.

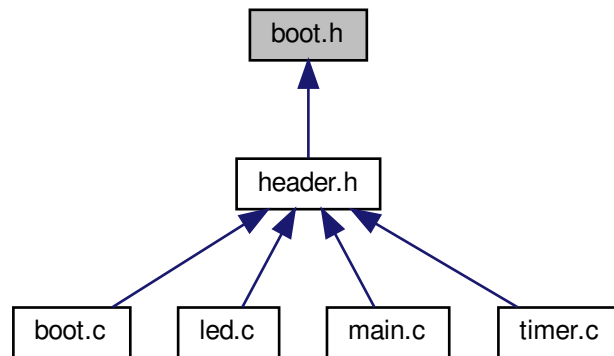
Returns

none.

7.247 boot.h File Reference

Demo program bootloader interface header file.

This graph shows which files directly or indirectly include this file:



Functions

- void [BootComInit](#) (void)
Initializes the communication interface.
- void [BootComCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootActivate](#) (void)
Bootloader activation function.

7.247.1 Detailed Description

Demo program bootloader interface header file.

7.247.2 Function Documentation

7.247.2.1 BootActivate()

```
void BootActivate (  
    void )
```

Bootloader activation function.

Returns

none.

7.247.2.2 BootComCheckActivationRequest()

```
void BootComCheckActivationRequest (  
    void )
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.247.2.3 BootComInit()

```
void BootComInit (  
    void )
```

Initializes the communication interface.

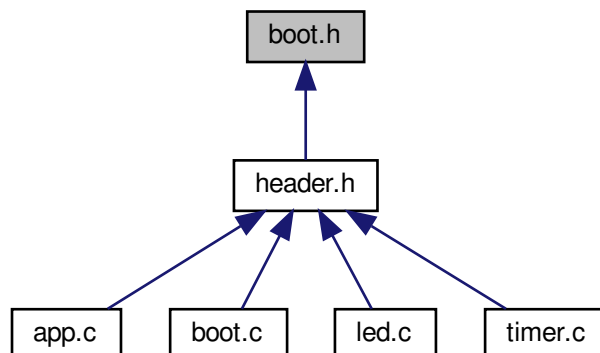
Returns

none.

7.248 boot.h File Reference

Demo program bootloader interface header file.

This graph shows which files directly or indirectly include this file:



Functions

- void [BootComInit](#) (void)
Initializes the communication interface.
- void [BootComCheckActivationRequest](#) (void)
Receives the `CONNECT` request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootActivate](#) (void)
Bootloader activation function.

7.248.1 Detailed Description

Demo program bootloader interface header file.

7.248.2 Function Documentation

7.248.2.1 [BootActivate\(\)](#)

```
void BootActivate (  
    void )
```

Bootloader activation function.

Returns

none.

7.248.2.2 [BootComCheckActivationRequest\(\)](#)

```
void BootComCheckActivationRequest (  
    void )
```

Receives the `CONNECT` request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.248.2.3 BootComInit()

```
void BootComInit (
    void )
```

Initializes the communication interface.

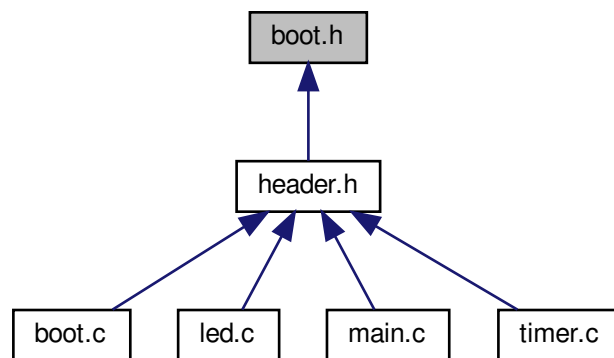
Returns

none.

7.249 boot.h File Reference

Demo program bootloader interface header file.

This graph shows which files directly or indirectly include this file:



Functions

- void [BootComInit](#) (void)
Initializes the communication interface.
- void [BootComCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootActivate](#) (void)
Bootloader activation function.

7.249.1 Detailed Description

Demo program bootloader interface header file.

7.249.2 Function Documentation

7.249.2.1 BootActivate()

```
void BootActivate (
    void )
```

Bootloader activation function.

Returns

none.

7.249.2.2 BootComCheckActivationRequest()

```
void BootComCheckActivationRequest (
    void )
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.249.2.3 BootComInit()

```
void BootComInit (
    void )
```

Initializes the communication interface.

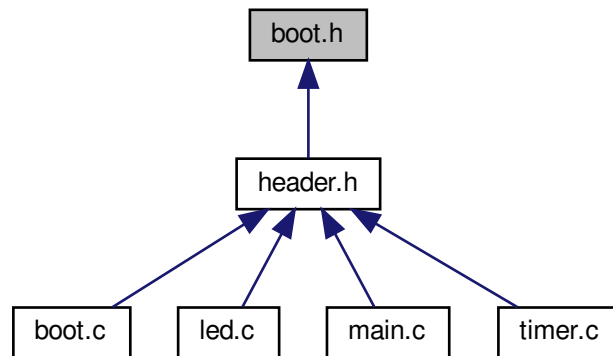
Returns

none.

7.250 boot.h File Reference

Demo program bootloader interface header file.

This graph shows which files directly or indirectly include this file:



Functions

- void [BootComInit](#) (void)
Initializes the communication interface.
- void [BootComCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootActivate](#) (void)
Bootloader activation function.

7.250.1 Detailed Description

Demo program bootloader interface header file.

7.250.2 Function Documentation

7.250.2.1 BootActivate()

```
void BootActivate (  
    void )
```

Bootloader activation function.

Returns

none.

7.250.2.2 BootComCheckActivationRequest()

```
void BootComCheckActivationRequest (  
    void )
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.250.2.3 BootComInit()

```
void BootComInit (  
    void )
```

Initializes the communication interface.

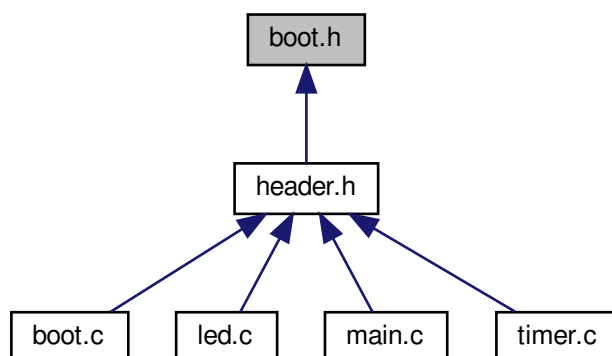
Returns

none.

7.251 boot.h File Reference

Demo program bootloader interface header file.

This graph shows which files directly or indirectly include this file:



Functions

- void [BootComInit](#) (void)
Initializes the communication interface.
- void [BootComCheckActivationRequest](#) (void)
Receives the `CONNECT` request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootActivate](#) (void)
Bootloader activation function.

7.251.1 Detailed Description

Demo program bootloader interface header file.

7.251.2 Function Documentation

7.251.2.1 [BootActivate\(\)](#)

```
void BootActivate (  
    void )
```

Bootloader activation function.

Returns

none.

7.251.2.2 [BootComCheckActivationRequest\(\)](#)

```
void BootComCheckActivationRequest (  
    void )
```

Receives the `CONNECT` request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.251.2.3 BootComInit()

```
void BootComInit (
    void )
```

Initializes the communication interface.

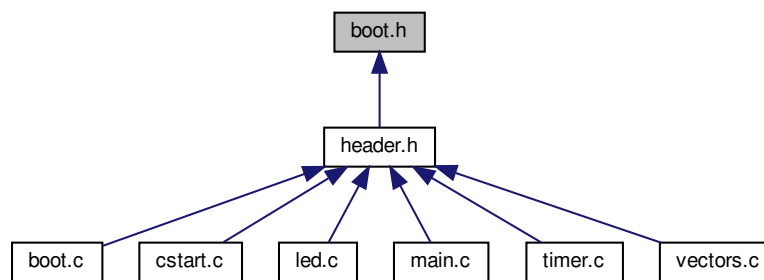
Returns

none.

7.252 boot.h File Reference

Demo program bootloader interface header file.

This graph shows which files directly or indirectly include this file:



Functions

- void [BootComInit](#) (void)
Initializes the communication interface.
- void [BootComCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootActivate](#) (void)
Bootloader activation function.

7.252.1 Detailed Description

Demo program bootloader interface header file.

7.252.2 Function Documentation

7.252.2.1 BootActivate()

```
void BootActivate (
    void )
```

Bootloader activation function.

Returns

none.

7.252.2.2 BootComCheckActivationRequest()

```
void BootComCheckActivationRequest (
    void )
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.252.2.3 BootComInit()

```
void BootComInit (
    void )
```

Initializes the communication interface.

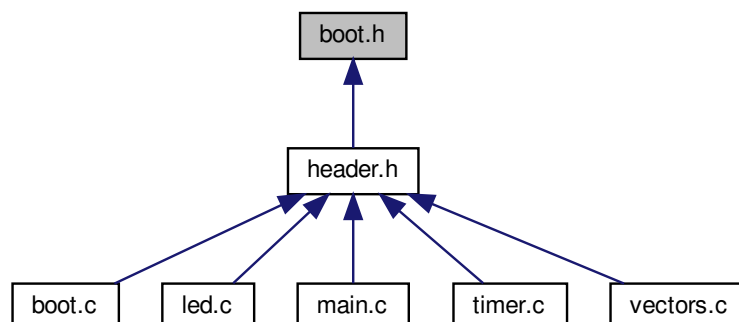
Returns

none.

7.253 boot.h File Reference

Demo program bootloader interface header file.

This graph shows which files directly or indirectly include this file:



Functions

- void [BootComInit](#) (void)
Initializes the communication interface.
- void [BootComCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootActivate](#) (void)
Bootloader activation function.

7.253.1 Detailed Description

Demo program bootloader interface header file.

7.253.2 Function Documentation

7.253.2.1 BootActivate()

```
void BootActivate (  
    void )
```

Bootloader activation function.

Returns

none.

7.253.2.2 BootComCheckActivationRequest()

```
void BootComCheckActivationRequest (  
    void )
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.253.2.3 BootComInit()

```
void BootComInit (
    void )
```

Initializes the communication interface.

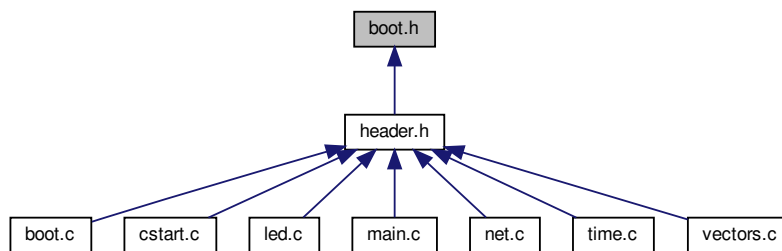
Returns

none.

7.254 boot.h File Reference

Demo program bootloader interface header file.

This graph shows which files directly or indirectly include this file:



Functions

- void [BootComInit](#) (void)
Initializes the communication interface.
- void [BootComCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootActivate](#) (void)
Bootloader activation function.

7.254.1 Detailed Description

Demo program bootloader interface header file.

7.254.2 Function Documentation

7.254.2.1 BootActivate()

```
void BootActivate (
    void )
```

Bootloader activation function.

Returns

none.

7.254.2.2 BootComCheckActivationRequest()

```
void BootComCheckActivationRequest (
    void )
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.254.2.3 BootComInit()

```
void BootComInit (
    void )
```

Initializes the communication interface.

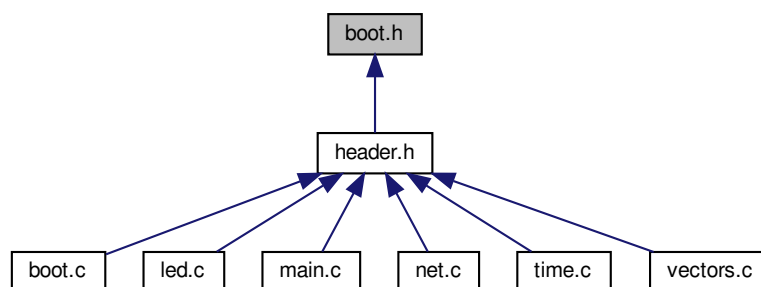
Returns

none.

7.255 boot.h File Reference

Demo program bootloader interface header file.

This graph shows which files directly or indirectly include this file:



Functions

- void [BootComInit](#) (void)
Initializes the communication interface.
- void [BootComCheckActivationRequest](#) (void)
Receives the `CONNECT` request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootActivate](#) (void)
Bootloader activation function.

7.255.1 Detailed Description

Demo program bootloader interface header file.

7.255.2 Function Documentation

7.255.2.1 [BootActivate\(\)](#)

```
void BootActivate (  
    void )
```

Bootloader activation function.

Returns

none.

7.255.2.2 [BootComCheckActivationRequest\(\)](#)

```
void BootComCheckActivationRequest (  
    void )
```

Receives the `CONNECT` request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.255.2.3 BootComInit()

```
void BootComInit (
    void )
```

Initializes the communication interface.

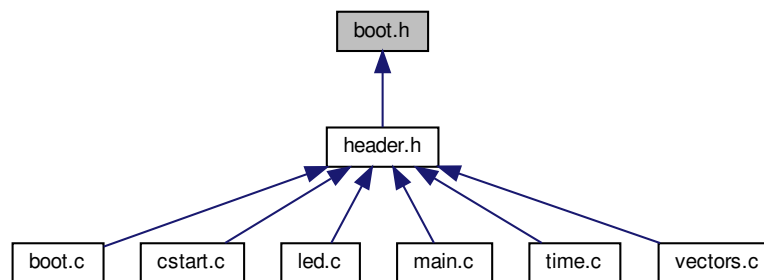
Returns

none.

7.256 boot.h File Reference

Demo program bootloader interface header file.

This graph shows which files directly or indirectly include this file:



Functions

- void [BootComInit](#) (void)
Initializes the communication interface.
- void [BootComCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootActivate](#) (void)
Bootloader activation function.

7.256.1 Detailed Description

Demo program bootloader interface header file.

7.256.2 Function Documentation

7.256.2.1 BootActivate()

```
void BootActivate (
    void )
```

Bootloader activation function.

Returns

none.

7.256.2.2 BootComCheckActivationRequest()

```
void BootComCheckActivationRequest (
    void )
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.256.2.3 BootComInit()

```
void BootComInit (
    void )
```

Initializes the communication interface.

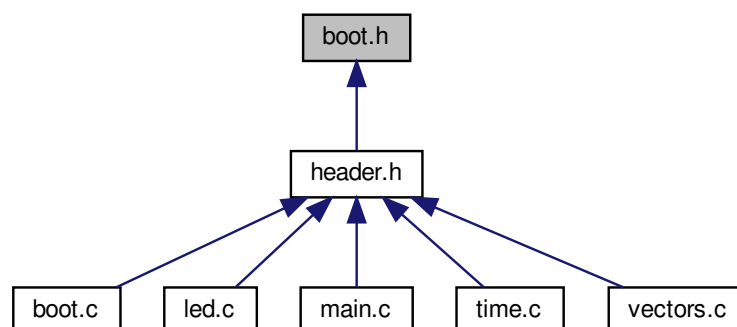
Returns

none.

7.257 boot.h File Reference

Demo program bootloader interface header file.

This graph shows which files directly or indirectly include this file:



Functions

- void [BootComInit](#) (void)
Initializes the communication interface.
- void [BootComCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootActivate](#) (void)
Bootloader activation function.

7.257.1 Detailed Description

Demo program bootloader interface header file.

7.257.2 Function Documentation

7.257.2.1 [BootActivate\(\)](#)

```
void BootActivate (  
    void )
```

Bootloader activation function.

Returns

none.

7.257.2.2 [BootComCheckActivationRequest\(\)](#)

```
void BootComCheckActivationRequest (  
    void )
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.257.2.3 BootComInit()

```
void BootComInit (
    void )
```

Initializes the communication interface.

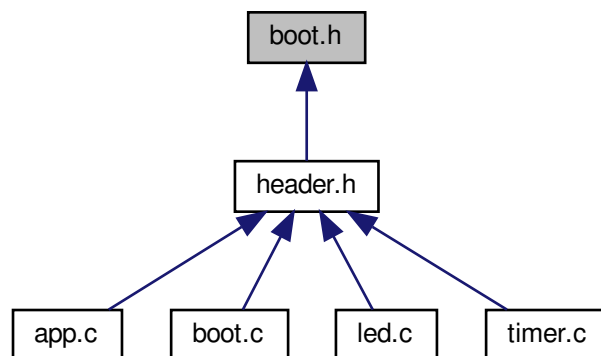
Returns

none.

7.258 boot.h File Reference

Demo program bootloader interface header file.

This graph shows which files directly or indirectly include this file:



Functions

- void [BootComInit](#) (void)
Initializes the communication interface.
- void [BootComCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootActivate](#) (void)
Bootloader activation function.

7.258.1 Detailed Description

Demo program bootloader interface header file.

7.258.2 Function Documentation

7.258.2.1 BootActivate()

```
void BootActivate (
    void )
```

Bootloader activation function.

Returns

none.

7.258.2.2 BootComCheckActivationRequest()

```
void BootComCheckActivationRequest (
    void )
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.258.2.3 BootComInit()

```
void BootComInit (
    void )
```

Initializes the communication interface.

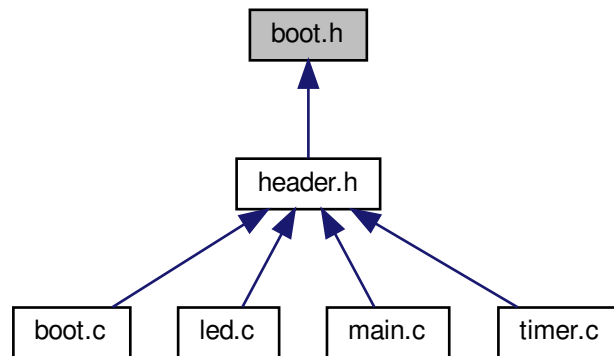
Returns

none.

7.259 boot.h File Reference

Demo program bootloader interface header file.

This graph shows which files directly or indirectly include this file:



Functions

- void [BootComInit](#) (void)
Initializes the communication interface.
- void [BootComCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootActivate](#) (void)
Bootloader activation function.

7.259.1 Detailed Description

Demo program bootloader interface header file.

7.259.2 Function Documentation

7.259.2.1 BootActivate()

```
void BootActivate (  
    void )
```

Bootloader activation function.

Returns

none.

7.259.2.2 BootComCheckActivationRequest()

```
void BootComCheckActivationRequest (  
    void )
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.259.2.3 BootComInit()

```
void BootComInit (  
    void )
```

Initializes the communication interface.

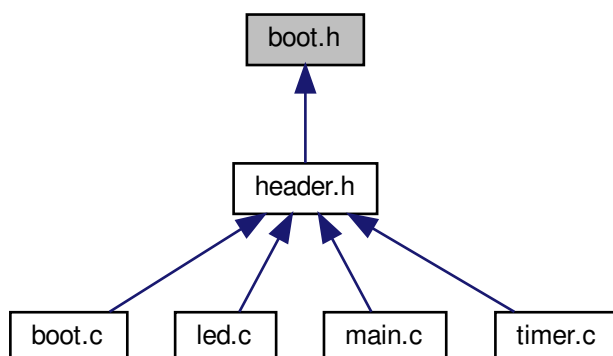
Returns

none.

7.260 boot.h File Reference

Demo program bootloader interface header file.

This graph shows which files directly or indirectly include this file:



Functions

- void [BootComInit](#) (void)
Initializes the communication interface.
- void [BootComCheckActivationRequest](#) (void)
Receives the `CONNECT` request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootActivate](#) (void)
Bootloader activation function.

7.260.1 Detailed Description

Demo program bootloader interface header file.

7.260.2 Function Documentation

7.260.2.1 [BootActivate\(\)](#)

```
void BootActivate (  
    void )
```

Bootloader activation function.

Returns

none.

7.260.2.2 [BootComCheckActivationRequest\(\)](#)

```
void BootComCheckActivationRequest (  
    void )
```

Receives the `CONNECT` request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.260.2.3 BootComInit()

```
void BootComInit (
    void )
```

Initializes the communication interface.

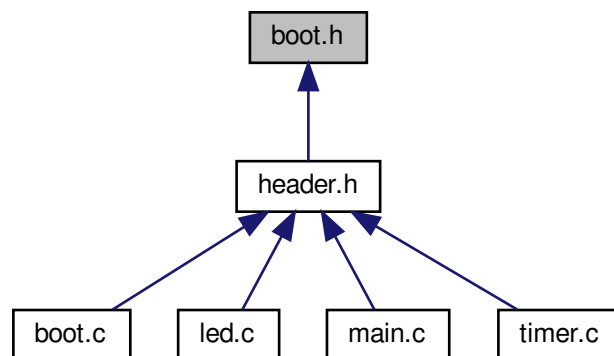
Returns

none.

7.261 boot.h File Reference

Demo program bootloader interface header file.

This graph shows which files directly or indirectly include this file:



Functions

- void [BootComInit](#) (void)
Initializes the communication interface.
- void [BootComCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootActivate](#) (void)
Bootloader activation function.

7.261.1 Detailed Description

Demo program bootloader interface header file.

7.261.2 Function Documentation

7.261.2.1 BootActivate()

```
void BootActivate (
    void )
```

Bootloader activation function.

Returns

none.

7.261.2.2 BootComCheckActivationRequest()

```
void BootComCheckActivationRequest (
    void )
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.261.2.3 BootComInit()

```
void BootComInit (
    void )
```

Initializes the communication interface.

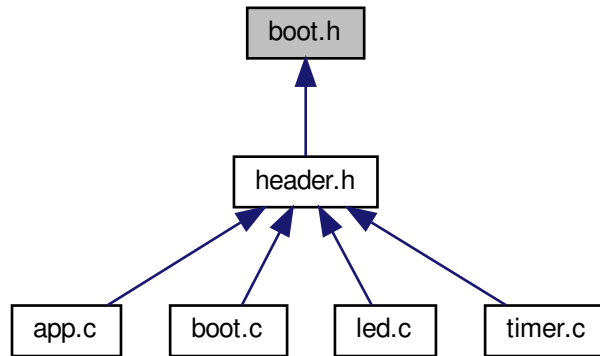
Returns

none.

7.262 boot.h File Reference

Demo program bootloader interface header file.

This graph shows which files directly or indirectly include this file:



Functions

- void [BootComInit](#) (void)
Initializes the communication interface.
- void [BootComCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootActivate](#) (void)
Bootloader activation function.

7.262.1 Detailed Description

Demo program bootloader interface header file.

7.262.2 Function Documentation

7.262.2.1 BootActivate()

```
void BootActivate (  
    void )
```

Bootloader activation function.

Returns

none.

7.262.2.2 BootComCheckActivationRequest()

```
void BootComCheckActivationRequest (
    void )
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.262.2.3 BootComInit()

```
void BootComInit (
    void )
```

Initializes the communication interface.

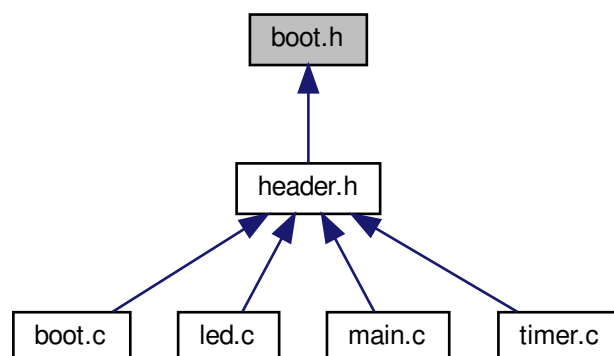
Returns

none.

7.263 boot.h File Reference

Demo program bootloader interface header file.

This graph shows which files directly or indirectly include this file:



Functions

- void [BootComInit](#) (void)
Initializes the communication interface.
- void [BootComCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootActivate](#) (void)
Bootloader activation function.

7.263.1 Detailed Description

Demo program bootloader interface header file.

7.263.2 Function Documentation

7.263.2.1 [BootActivate\(\)](#)

```
void BootActivate (  
    void )
```

Bootloader activation function.

Returns

none.

7.263.2.2 [BootComCheckActivationRequest\(\)](#)

```
void BootComCheckActivationRequest (  
    void )
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.263.2.3 BootComInit()

```
void BootComInit (
    void )
```

Initializes the communication interface.

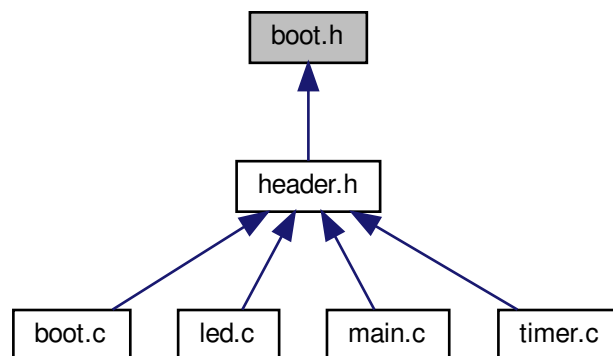
Returns

none.

7.264 boot.h File Reference

Demo program bootloader interface header file.

This graph shows which files directly or indirectly include this file:



Functions

- void [BootComInit](#) (void)
Initializes the communication interface.
- void [BootComCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootActivate](#) (void)
Bootloader activation function.

7.264.1 Detailed Description

Demo program bootloader interface header file.

7.264.2 Function Documentation

7.264.2.1 BootActivate()

```
void BootActivate (
    void )
```

Bootloader activation function.

Returns

none.

7.264.2.2 BootComCheckActivationRequest()

```
void BootComCheckActivationRequest (
    void )
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.264.2.3 BootComInit()

```
void BootComInit (
    void )
```

Initializes the communication interface.

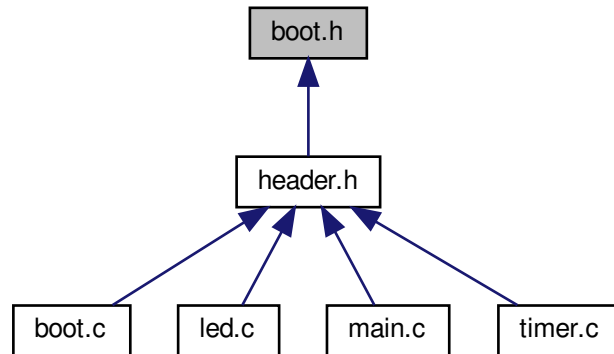
Returns

none.

7.265 boot.h File Reference

Demo program bootloader interface header file.

This graph shows which files directly or indirectly include this file:



Functions

- void [BootComInit](#) (void)
Initializes the communication interface.
- void [BootComCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootActivate](#) (void)
Bootloader activation function.

7.265.1 Detailed Description

Demo program bootloader interface header file.

7.265.2 Function Documentation

7.265.2.1 BootActivate()

```
void BootActivate (  
    void )
```

Bootloader activation function.

Returns

none.

7.265.2.2 BootComCheckActivationRequest()

```
void BootComCheckActivationRequest (  
    void )
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.265.2.3 BootComInit()

```
void BootComInit (  
    void )
```

Initializes the communication interface.

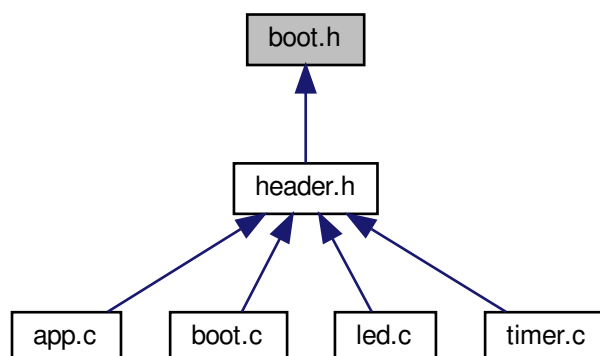
Returns

none.

7.266 boot.h File Reference

Demo program bootloader interface header file.

This graph shows which files directly or indirectly include this file:



Functions

- void [BootComInit](#) (void)
Initializes the communication interface.
- void [BootComCheckActivationRequest](#) (void)
Receives the `CONNECT` request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootActivate](#) (void)
Bootloader activation function.

7.266.1 Detailed Description

Demo program bootloader interface header file.

7.266.2 Function Documentation

7.266.2.1 [BootActivate\(\)](#)

```
void BootActivate (  
    void )
```

Bootloader activation function.

Returns

none.

7.266.2.2 [BootComCheckActivationRequest\(\)](#)

```
void BootComCheckActivationRequest (  
    void )
```

Receives the `CONNECT` request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.266.2.3 BootComInit()

```
void BootComInit (
    void )
```

Initializes the communication interface.

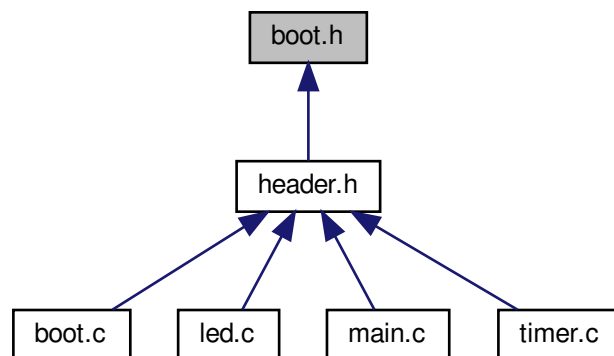
Returns

none.

7.267 boot.h File Reference

Demo program bootloader interface header file.

This graph shows which files directly or indirectly include this file:



Functions

- void [BootComInit](#) (void)
Initializes the communication interface.
- void [BootComCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootActivate](#) (void)
Bootloader activation function.

7.267.1 Detailed Description

Demo program bootloader interface header file.

7.267.2 Function Documentation

7.267.2.1 BootActivate()

```
void BootActivate (
    void )
```

Bootloader activation function.

Returns

none.

7.267.2.2 BootComCheckActivationRequest()

```
void BootComCheckActivationRequest (
    void )
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.267.2.3 BootComInit()

```
void BootComInit (
    void )
```

Initializes the communication interface.

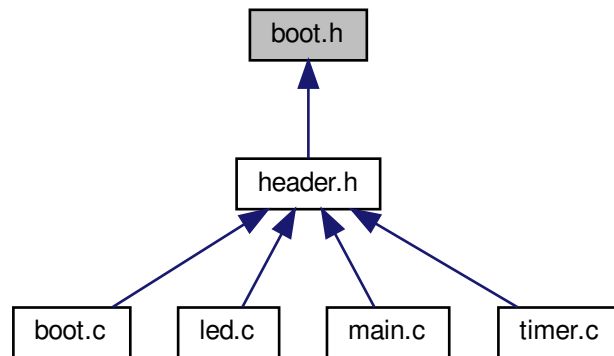
Returns

none.

7.268 boot.h File Reference

Demo program bootloader interface header file.

This graph shows which files directly or indirectly include this file:



Functions

- void [BootComInit](#) (void)
Initializes the communication interface.
- void [BootComCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootActivate](#) (void)
Bootloader activation function.

7.268.1 Detailed Description

Demo program bootloader interface header file.

7.268.2 Function Documentation

7.268.2.1 BootActivate()

```
void BootActivate (  
    void )
```

Bootloader activation function.

Returns

none.

7.268.2.2 BootComCheckActivationRequest()

```
void BootComCheckActivationRequest (  
    void )
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.268.2.3 BootComInit()

```
void BootComInit (  
    void )
```

Initializes the communication interface.

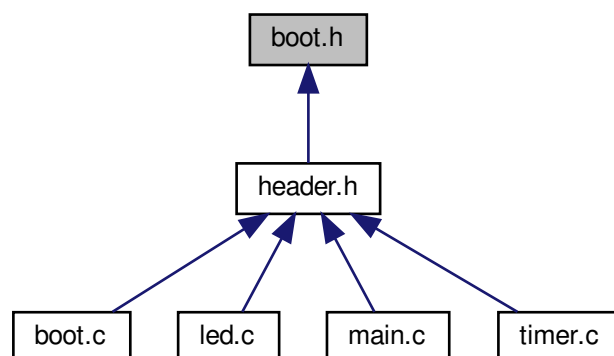
Returns

none.

7.269 boot.h File Reference

Demo program bootloader interface header file.

This graph shows which files directly or indirectly include this file:



Functions

- void [BootComInit](#) (void)
Initializes the communication interface.
- void [BootComCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootActivate](#) (void)
Bootloader activation function.

7.269.1 Detailed Description

Demo program bootloader interface header file.

7.269.2 Function Documentation

7.269.2.1 [BootActivate\(\)](#)

```
void BootActivate (  
    void )
```

Bootloader activation function.

Returns

none.

7.269.2.2 [BootComCheckActivationRequest\(\)](#)

```
void BootComCheckActivationRequest (  
    void )
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.269.2.3 BootComInit()

```
void BootComInit (
    void )
```

Initializes the communication interface.

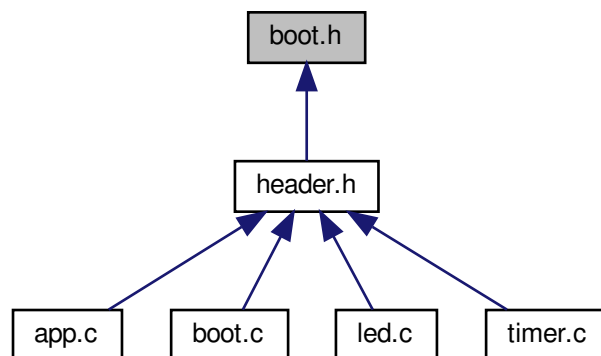
Returns

none.

7.270 boot.h File Reference

Demo program bootloader interface header file.

This graph shows which files directly or indirectly include this file:



Functions

- void [BootComInit](#) (void)
Initializes the communication interface.
- void [BootComCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootActivate](#) (void)
Bootloader activation function.

7.270.1 Detailed Description

Demo program bootloader interface header file.

7.270.2 Function Documentation

7.270.2.1 BootActivate()

```
void BootActivate (
    void )
```

Bootloader activation function.

Returns

none.

7.270.2.2 BootComCheckActivationRequest()

```
void BootComCheckActivationRequest (
    void )
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.270.2.3 BootComInit()

```
void BootComInit (
    void )
```

Initializes the communication interface.

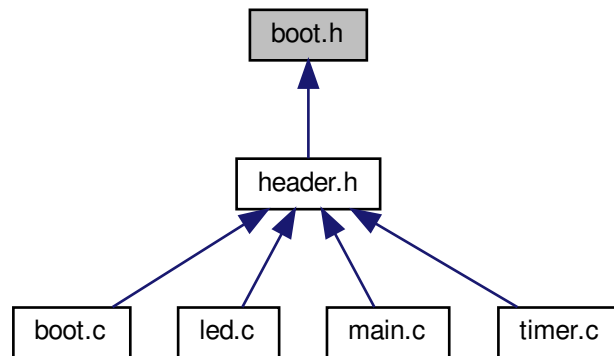
Returns

none.

7.271 boot.h File Reference

Demo program bootloader interface header file.

This graph shows which files directly or indirectly include this file:



Functions

- void [BootComInit](#) (void)
Initializes the communication interface.
- void [BootComCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootActivate](#) (void)
Bootloader activation function.

7.271.1 Detailed Description

Demo program bootloader interface header file.

7.271.2 Function Documentation

7.271.2.1 BootActivate()

```
void BootActivate (  
    void )
```

Bootloader activation function.

Returns

none.

7.271.2.2 BootComCheckActivationRequest()

```
void BootComCheckActivationRequest (  
    void )
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.271.2.3 BootComInit()

```
void BootComInit (  
    void )
```

Initializes the communication interface.

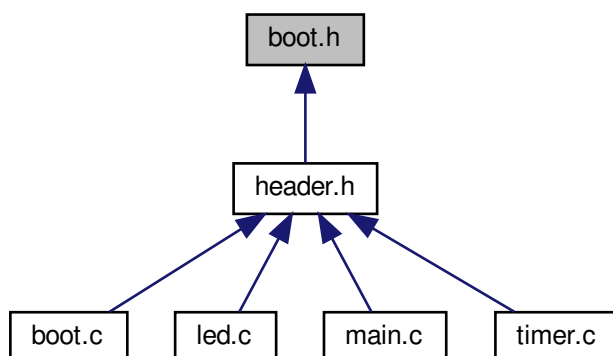
Returns

none.

7.272 boot.h File Reference

Demo program bootloader interface header file.

This graph shows which files directly or indirectly include this file:



Functions

- void [BootComInit](#) (void)
Initializes the communication interface.
- void [BootComCheckActivationRequest](#) (void)
Receives the `CONNECT` request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootActivate](#) (void)
Bootloader activation function.

7.272.1 Detailed Description

Demo program bootloader interface header file.

7.272.2 Function Documentation

7.272.2.1 [BootActivate\(\)](#)

```
void BootActivate (  
    void )
```

Bootloader activation function.

Returns

none.

7.272.2.2 [BootComCheckActivationRequest\(\)](#)

```
void BootComCheckActivationRequest (  
    void )
```

Receives the `CONNECT` request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.272.2.3 BootComInit()

```
void BootComInit (
    void )
```

Initializes the communication interface.

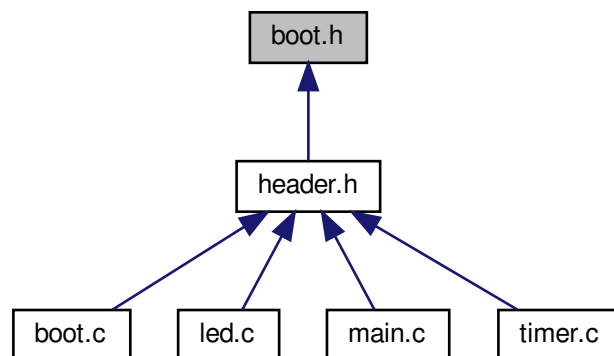
Returns

none.

7.273 boot.h File Reference

Demo program bootloader interface header file.

This graph shows which files directly or indirectly include this file:



Functions

- void [BootComInit](#) (void)
Initializes the communication interface.
- void [BootComCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootActivate](#) (void)
Bootloader activation function.

7.273.1 Detailed Description

Demo program bootloader interface header file.

7.273.2 Function Documentation

7.273.2.1 BootActivate()

```
void BootActivate (
    void )
```

Bootloader activation function.

Returns

none.

7.273.2.2 BootComCheckActivationRequest()

```
void BootComCheckActivationRequest (
    void )
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.273.2.3 BootComInit()

```
void BootComInit (
    void )
```

Initializes the communication interface.

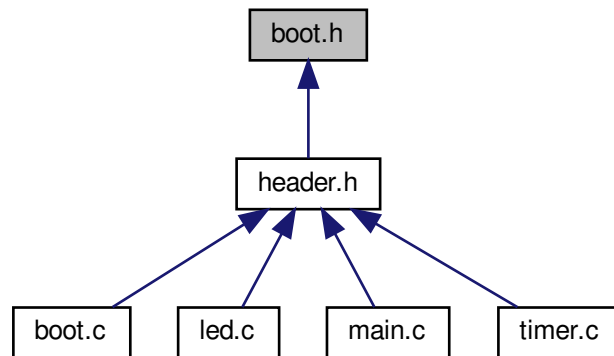
Returns

none.

7.274 boot.h File Reference

Demo program bootloader interface header file.

This graph shows which files directly or indirectly include this file:



Functions

- void [BootComInit](#) (void)
Initializes the communication interface.
- void [BootComCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootActivate](#) (void)
Bootloader activation function.

7.274.1 Detailed Description

Demo program bootloader interface header file.

7.274.2 Function Documentation

7.274.2.1 BootActivate()

```
void BootActivate (  
    void )
```

Bootloader activation function.

Returns

none.

7.274.2.2 BootComCheckActivationRequest()

```
void BootComCheckActivationRequest (  
    void )
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.274.2.3 BootComInit()

```
void BootComInit (  
    void )
```

Initializes the communication interface.

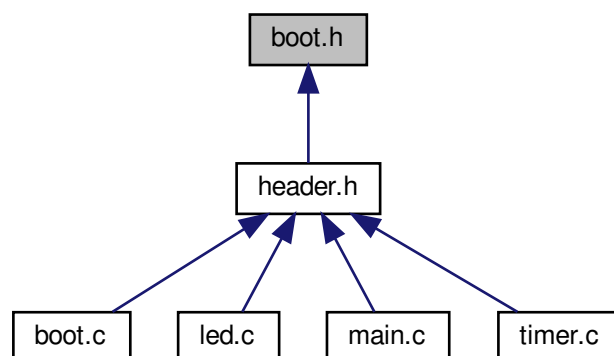
Returns

none.

7.275 boot.h File Reference

Demo program bootloader interface header file.

This graph shows which files directly or indirectly include this file:



Functions

- void [BootComInit](#) (void)
Initializes the communication interface.
- void [BootComCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootActivate](#) (void)
Bootloader activation function.

7.275.1 Detailed Description

Demo program bootloader interface header file.

7.275.2 Function Documentation

7.275.2.1 [BootActivate\(\)](#)

```
void BootActivate (  
    void )
```

Bootloader activation function.

Returns

none.

7.275.2.2 [BootComCheckActivationRequest\(\)](#)

```
void BootComCheckActivationRequest (  
    void )
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.275.2.3 BootComInit()

```
void BootComInit (
    void )
```

Initializes the communication interface.

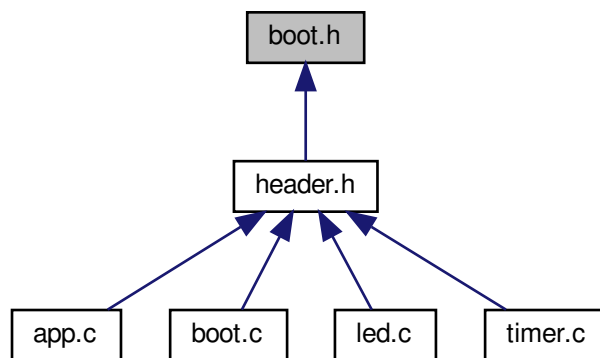
Returns

none.

7.276 boot.h File Reference

Demo program bootloader interface header file.

This graph shows which files directly or indirectly include this file:



Functions

- void [BootComInit](#) (void)
Initializes the communication interface.
- void [BootComCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootActivate](#) (void)
Bootloader activation function.

7.276.1 Detailed Description

Demo program bootloader interface header file.

7.276.2 Function Documentation

7.276.2.1 BootActivate()

```
void BootActivate (
    void )
```

Bootloader activation function.

Returns

none.

7.276.2.2 BootComCheckActivationRequest()

```
void BootComCheckActivationRequest (
    void )
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.276.2.3 BootComInit()

```
void BootComInit (
    void )
```

Initializes the communication interface.

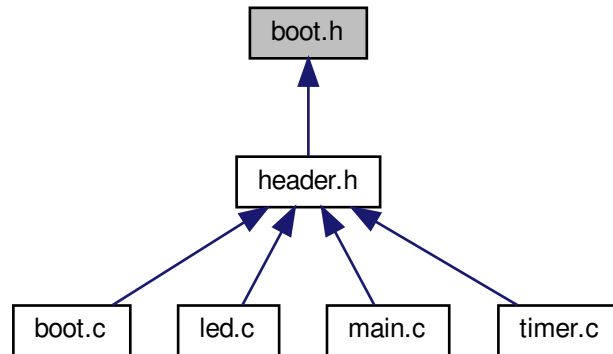
Returns

none.

7.277 boot.h File Reference

Demo program bootloader interface header file.

This graph shows which files directly or indirectly include this file:



Functions

- void [BootComInit](#) (void)
Initializes the communication interface.
- void [BootComCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootActivate](#) (void)
Bootloader activation function.

7.277.1 Detailed Description

Demo program bootloader interface header file.

7.277.2 Function Documentation

7.277.2.1 BootActivate()

```
void BootActivate (  
    void )
```

Bootloader activation function.

Returns

none.

7.277.2.2 BootComCheckActivationRequest()

```
void BootComCheckActivationRequest (
    void )
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.277.2.3 BootComInit()

```
void BootComInit (
    void )
```

Initializes the communication interface.

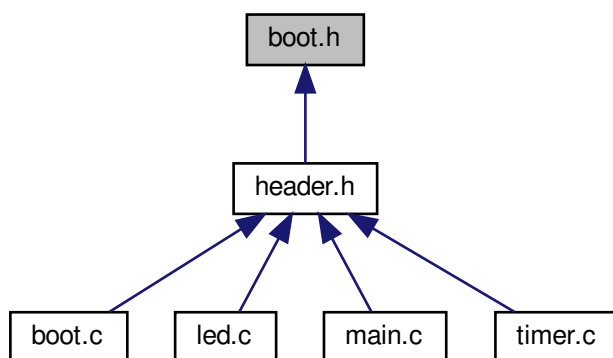
Returns

none.

7.278 boot.h File Reference

Demo program bootloader interface header file.

This graph shows which files directly or indirectly include this file:



Functions

- void [BootComInit](#) (void)
Initializes the communication interface.
- void [BootComCheckActivationRequest](#) (void)
Receives the `CONNECT` request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootActivate](#) (void)
Bootloader activation function.

7.278.1 Detailed Description

Demo program bootloader interface header file.

7.278.2 Function Documentation

7.278.2.1 [BootActivate\(\)](#)

```
void BootActivate (  
    void )
```

Bootloader activation function.

Returns

none.

7.278.2.2 [BootComCheckActivationRequest\(\)](#)

```
void BootComCheckActivationRequest (  
    void )
```

Receives the `CONNECT` request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.278.2.3 BootComInit()

```
void BootComInit (
    void )
```

Initializes the communication interface.

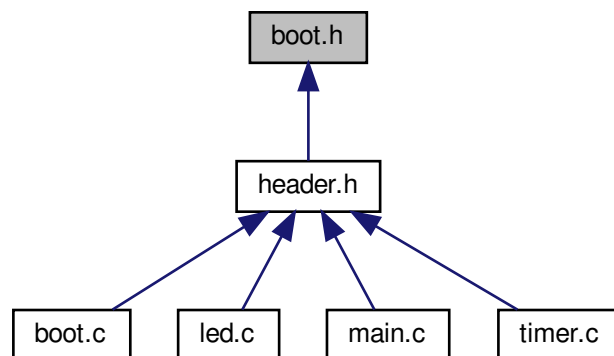
Returns

none.

7.279 boot.h File Reference

Demo program bootloader interface header file.

This graph shows which files directly or indirectly include this file:



Functions

- void [BootComInit](#) (void)
Initializes the communication interface.
- void [BootComCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootActivate](#) (void)
Bootloader activation function.

7.279.1 Detailed Description

Demo program bootloader interface header file.

7.279.2 Function Documentation

7.279.2.1 BootActivate()

```
void BootActivate (
    void )
```

Bootloader activation function.

Returns

none.

7.279.2.2 BootComCheckActivationRequest()

```
void BootComCheckActivationRequest (
    void )
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.279.2.3 BootComInit()

```
void BootComInit (
    void )
```

Initializes the communication interface.

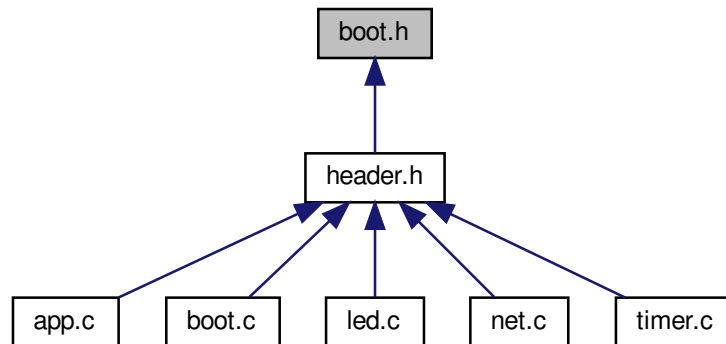
Returns

none.

7.280 boot.h File Reference

Demo program bootloader interface header file.

This graph shows which files directly or indirectly include this file:



Functions

- void [BootComInit](#) (void)
Initializes the communication interface.
- void [BootComCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootActivate](#) (void)
Bootloader activation function.

7.280.1 Detailed Description

Demo program bootloader interface header file.

7.280.2 Function Documentation

7.280.2.1 BootActivate()

```
void BootActivate (  
    void )
```

Bootloader activation function.

Returns

none.

7.280.2.2 BootComCheckActivationRequest()

```
void BootComCheckActivationRequest (  
    void )
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.280.2.3 BootComInit()

```
void BootComInit (  
    void )
```

Initializes the communication interface.

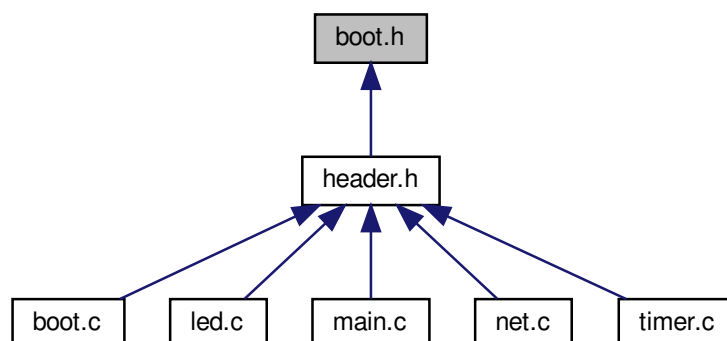
Returns

none.

7.281 boot.h File Reference

Demo program bootloader interface header file.

This graph shows which files directly or indirectly include this file:



Functions

- void [BootComInit](#) (void)
Initializes the communication interface.
- void [BootComCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootActivate](#) (void)
Bootloader activation function.

7.281.1 Detailed Description

Demo program bootloader interface header file.

7.281.2 Function Documentation

7.281.2.1 BootActivate()

```
void BootActivate (  
    void )
```

Bootloader activation function.

Returns

none.

7.281.2.2 BootComCheckActivationRequest()

```
void BootComCheckActivationRequest (  
    void )
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.281.2.3 BootComInit()

```
void BootComInit (
    void )
```

Initializes the communication interface.

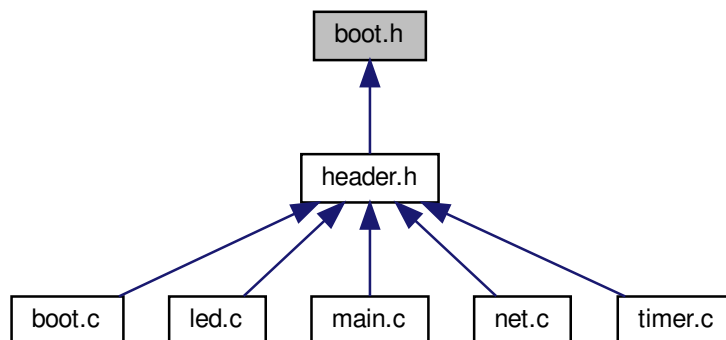
Returns

none.

7.282 boot.h File Reference

Demo program bootloader interface header file.

This graph shows which files directly or indirectly include this file:



Functions

- void [BootComInit](#) (void)
Initializes the communication interface.
- void [BootComCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootActivate](#) (void)
Bootloader activation function.

7.282.1 Detailed Description

Demo program bootloader interface header file.

7.282.2 Function Documentation

7.282.2.1 BootActivate()

```
void BootActivate (
    void )
```

Bootloader activation function.

Returns

none.

7.282.2.2 BootComCheckActivationRequest()

```
void BootComCheckActivationRequest (
    void )
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.282.2.3 BootComInit()

```
void BootComInit (
    void )
```

Initializes the communication interface.

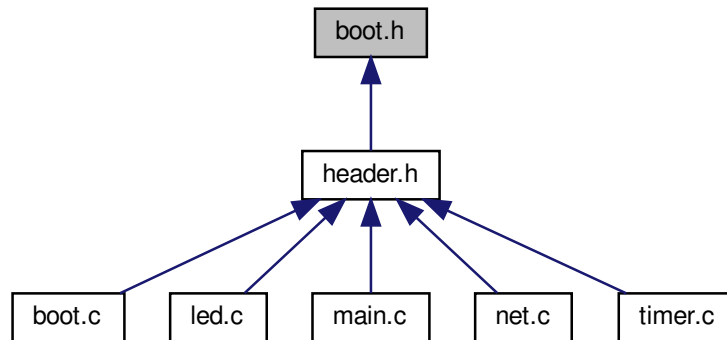
Returns

none.

7.283 boot.h File Reference

Demo program bootloader interface header file.

This graph shows which files directly or indirectly include this file:



Functions

- void [BootComInit](#) (void)
Initializes the communication interface.
- void [BootComCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootActivate](#) (void)
Bootloader activation function.

7.283.1 Detailed Description

Demo program bootloader interface header file.

7.283.2 Function Documentation

7.283.2.1 BootActivate()

```
void BootActivate (  
    void )
```

Bootloader activation function.

Returns

none.

7.283.2.2 BootComCheckActivationRequest()

```
void BootComCheckActivationRequest (  
    void )
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.283.2.3 BootComInit()

```
void BootComInit (  
    void )
```

Initializes the communication interface.

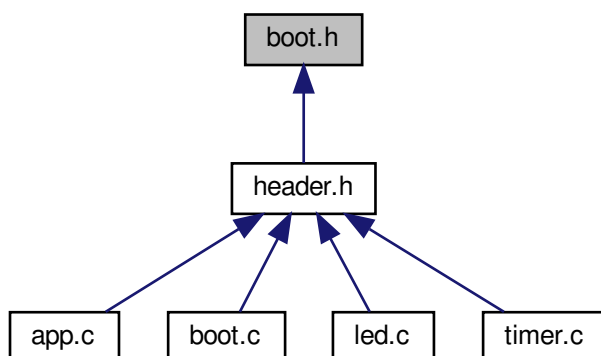
Returns

none.

7.284 boot.h File Reference

Demo program bootloader interface header file.

This graph shows which files directly or indirectly include this file:



Functions

- void [BootComInit](#) (void)
Initializes the communication interface.
- void [BootComCheckActivationRequest](#) (void)
Receives the `CONNECT` request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootActivate](#) (void)
Bootloader activation function.

7.284.1 Detailed Description

Demo program bootloader interface header file.

7.284.2 Function Documentation

7.284.2.1 [BootActivate\(\)](#)

```
void BootActivate (  
    void )
```

Bootloader activation function.

Returns

none.

7.284.2.2 [BootComCheckActivationRequest\(\)](#)

```
void BootComCheckActivationRequest (  
    void )
```

Receives the `CONNECT` request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.284.2.3 BootComInit()

```
void BootComInit (
    void )
```

Initializes the communication interface.

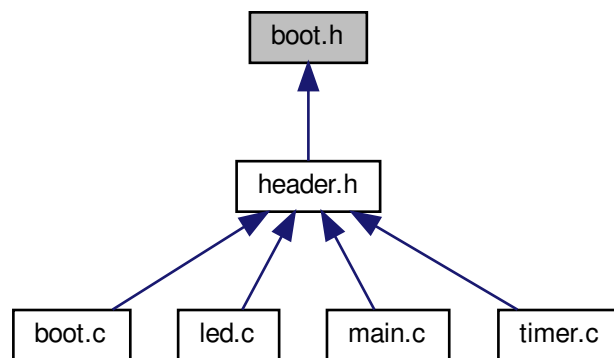
Returns

none.

7.285 boot.h File Reference

Demo program bootloader interface header file.

This graph shows which files directly or indirectly include this file:



Functions

- void [BootComInit](#) (void)
Initializes the communication interface.
- void [BootComCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootActivate](#) (void)
Bootloader activation function.

7.285.1 Detailed Description

Demo program bootloader interface header file.

7.285.2 Function Documentation

7.285.2.1 BootActivate()

```
void BootActivate (
    void )
```

Bootloader activation function.

Returns

none.

7.285.2.2 BootComCheckActivationRequest()

```
void BootComCheckActivationRequest (
    void )
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.285.2.3 BootComInit()

```
void BootComInit (
    void )
```

Initializes the communication interface.

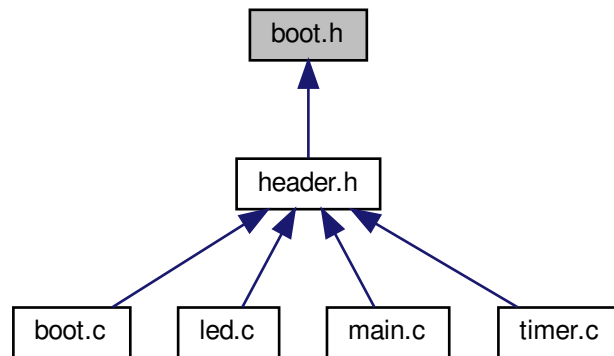
Returns

none.

7.286 boot.h File Reference

Demo program bootloader interface header file.

This graph shows which files directly or indirectly include this file:



Functions

- void [BootComInit](#) (void)
Initializes the communication interface.
- void [BootComCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootActivate](#) (void)
Bootloader activation function.

7.286.1 Detailed Description

Demo program bootloader interface header file.

7.286.2 Function Documentation

7.286.2.1 BootActivate()

```
void BootActivate (  
    void )
```

Bootloader activation function.

Returns

none.

7.286.2.2 BootComCheckActivationRequest()

```
void BootComCheckActivationRequest (
    void )
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.286.2.3 BootComInit()

```
void BootComInit (
    void )
```

Initializes the communication interface.

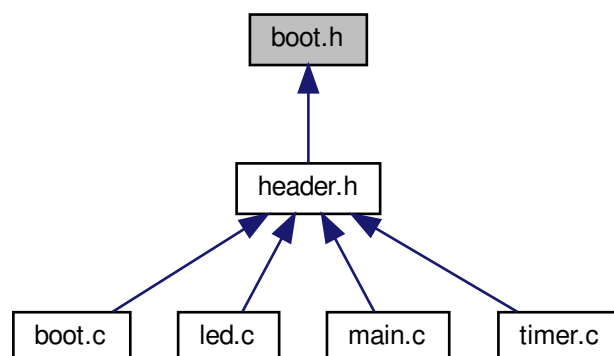
Returns

none.

7.287 boot.h File Reference

Demo program bootloader interface header file.

This graph shows which files directly or indirectly include this file:



Functions

- void [BootComInit](#) (void)
Initializes the communication interface.
- void [BootComCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootActivate](#) (void)
Bootloader activation function.

7.287.1 Detailed Description

Demo program bootloader interface header file.

7.287.2 Function Documentation

7.287.2.1 [BootActivate\(\)](#)

```
void BootActivate (  
    void )
```

Bootloader activation function.

Returns

none.

7.287.2.2 [BootComCheckActivationRequest\(\)](#)

```
void BootComCheckActivationRequest (  
    void )
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.287.2.3 BootComInit()

```
void BootComInit (
    void )
```

Initializes the communication interface.

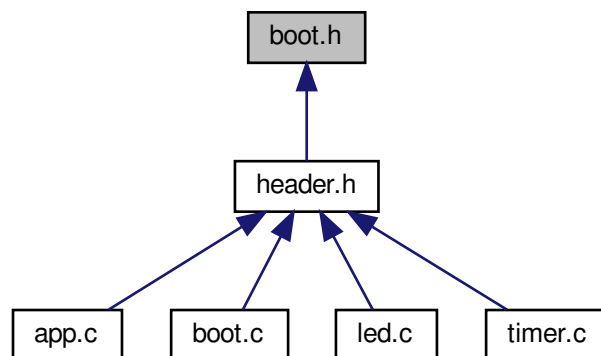
Returns

none.

7.288 boot.h File Reference

Demo program bootloader interface header file.

This graph shows which files directly or indirectly include this file:



Functions

- void [BootComInit](#) (void)
Initializes the communication interface.
- void [BootComCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootActivate](#) (void)
Bootloader activation function.

7.288.1 Detailed Description

Demo program bootloader interface header file.

7.288.2 Function Documentation

7.288.2.1 BootActivate()

```
void BootActivate (
    void )
```

Bootloader activation function.

Returns

none.

7.288.2.2 BootComCheckActivationRequest()

```
void BootComCheckActivationRequest (
    void )
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.288.2.3 BootComInit()

```
void BootComInit (
    void )
```

Initializes the communication interface.

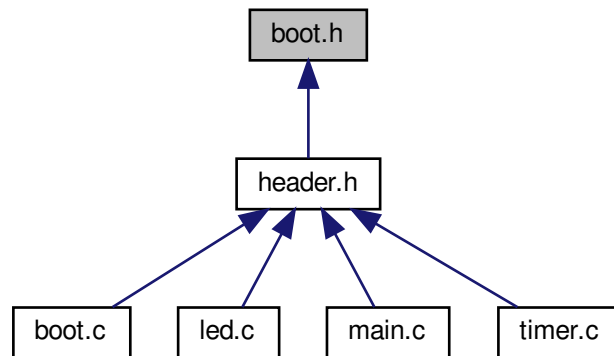
Returns

none.

7.289 boot.h File Reference

Demo program bootloader interface header file.

This graph shows which files directly or indirectly include this file:



Functions

- void [BootComInit](#) (void)
Initializes the communication interface.
- void [BootComCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootActivate](#) (void)
Bootloader activation function.

7.289.1 Detailed Description

Demo program bootloader interface header file.

7.289.2 Function Documentation

7.289.2.1 BootActivate()

```
void BootActivate (  
    void )
```

Bootloader activation function.

Returns

none.

7.289.2.2 BootComCheckActivationRequest()

```
void BootComCheckActivationRequest (  
    void )
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.289.2.3 BootComInit()

```
void BootComInit (  
    void )
```

Initializes the communication interface.

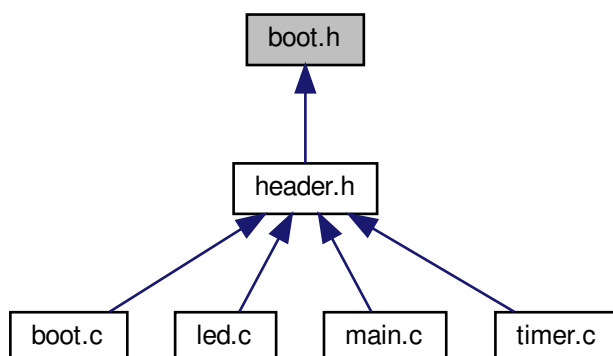
Returns

none.

7.290 boot.h File Reference

Demo program bootloader interface header file.

This graph shows which files directly or indirectly include this file:



Functions

- void [BootComInit](#) (void)
Initializes the communication interface.
- void [BootComCheckActivationRequest](#) (void)
Receives the `CONNECT` request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootActivate](#) (void)
Bootloader activation function.

7.290.1 Detailed Description

Demo program bootloader interface header file.

7.290.2 Function Documentation

7.290.2.1 [BootActivate\(\)](#)

```
void BootActivate (  
    void )
```

Bootloader activation function.

Returns

none.

7.290.2.2 [BootComCheckActivationRequest\(\)](#)

```
void BootComCheckActivationRequest (  
    void )
```

Receives the `CONNECT` request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.290.2.3 BootComInit()

```
void BootComInit (
    void )
```

Initializes the communication interface.

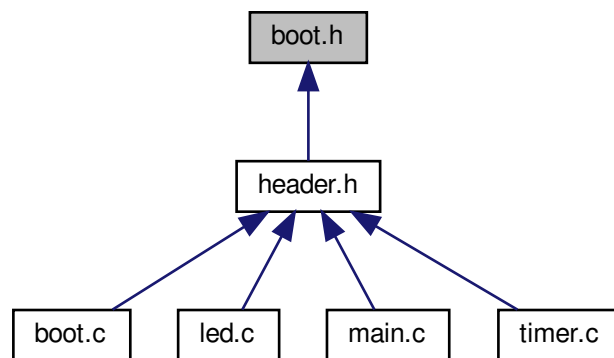
Returns

none.

7.291 boot.h File Reference

Demo program bootloader interface header file.

This graph shows which files directly or indirectly include this file:



Functions

- void [BootComInit](#) (void)
Initializes the communication interface.
- void [BootComCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootActivate](#) (void)
Bootloader activation function.

7.291.1 Detailed Description

Demo program bootloader interface header file.

7.291.2 Function Documentation

7.291.2.1 BootActivate()

```
void BootActivate (
    void )
```

Bootloader activation function.

Returns

none.

7.291.2.2 BootComCheckActivationRequest()

```
void BootComCheckActivationRequest (
    void )
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.291.2.3 BootComInit()

```
void BootComInit (
    void )
```

Initializes the communication interface.

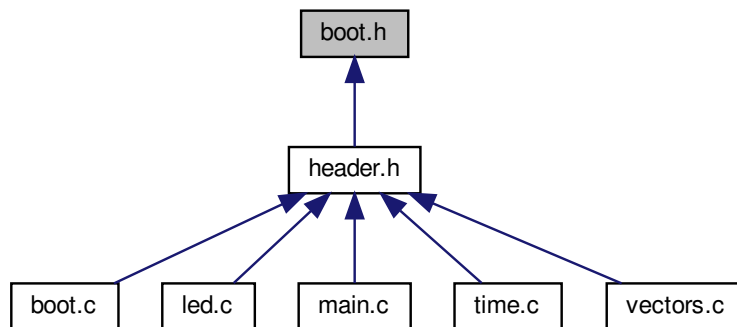
Returns

none.

7.292 boot.h File Reference

Demo program bootloader interface header file.

This graph shows which files directly or indirectly include this file:



Functions

- void [BootComInit](#) (void)
Initializes the communication interface.
- void [BootComCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootActivate](#) (void)
Bootloader activation function.

7.292.1 Detailed Description

Demo program bootloader interface header file.

7.292.2 Function Documentation

7.292.2.1 BootActivate()

```
void BootActivate (  
    void )
```

Bootloader activation function.

Returns

none.

7.292.2.2 BootComCheckActivationRequest()

```
void BootComCheckActivationRequest (  
    void )
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.292.2.3 BootComInit()

```
void BootComInit (  
    void )
```

Initializes the communication interface.

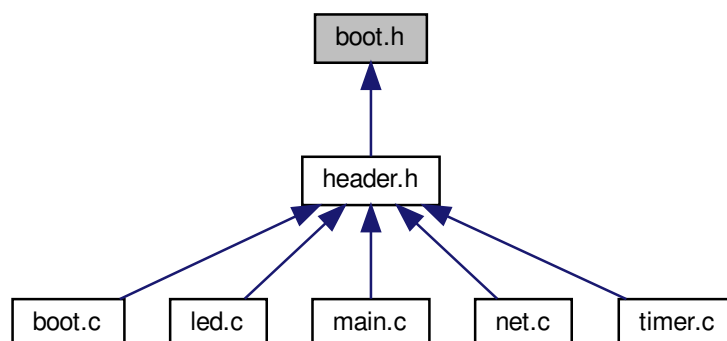
Returns

none.

7.293 boot.h File Reference

Demo program bootloader interface header file.

This graph shows which files directly or indirectly include this file:



Functions

- void [BootComInit](#) (void)
Initializes the communication interface.
- void [BootComCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootActivate](#) (void)
Bootloader activation function.

7.293.1 Detailed Description

Demo program bootloader interface header file.

7.293.2 Function Documentation

7.293.2.1 [BootActivate\(\)](#)

```
void BootActivate (  
    void )
```

Bootloader activation function.

Returns

none.

7.293.2.2 [BootComCheckActivationRequest\(\)](#)

```
void BootComCheckActivationRequest (  
    void )
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.293.2.3 BootComInit()

```
void BootComInit (
    void )
```

Initializes the communication interface.

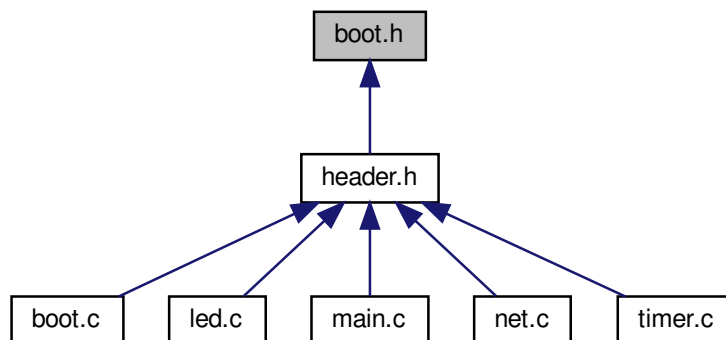
Returns

none.

7.294 boot.h File Reference

Demo program bootloader interface header file.

This graph shows which files directly or indirectly include this file:



Functions

- void [BootComInit](#) (void)
Initializes the communication interface.
- void [BootComCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootActivate](#) (void)
Bootloader activation function.

7.294.1 Detailed Description

Demo program bootloader interface header file.

7.294.2 Function Documentation

7.294.2.1 BootActivate()

```
void BootActivate (
    void )
```

Bootloader activation function.

Returns

none.

7.294.2.2 BootComCheckActivationRequest()

```
void BootComCheckActivationRequest (
    void )
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.294.2.3 BootComInit()

```
void BootComInit (
    void )
```

Initializes the communication interface.

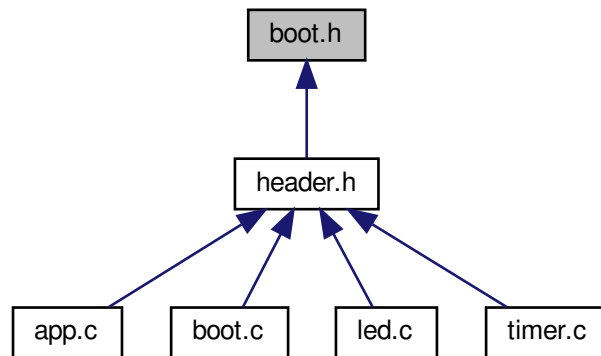
Returns

none.

7.295 boot.h File Reference

Demo program bootloader interface header file.

This graph shows which files directly or indirectly include this file:



Functions

- void [BootComInit](#) (void)
Initializes the communication interface.
- void [BootComCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootActivate](#) (void)
Bootloader activation function.

7.295.1 Detailed Description

Demo program bootloader interface header file.

7.295.2 Function Documentation

7.295.2.1 BootActivate()

```
void BootActivate (  
    void )
```

Bootloader activation function.

Returns

none.

7.295.2.2 BootComCheckActivationRequest()

```
void BootComCheckActivationRequest (  
    void )
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.295.2.3 BootComInit()

```
void BootComInit (  
    void )
```

Initializes the communication interface.

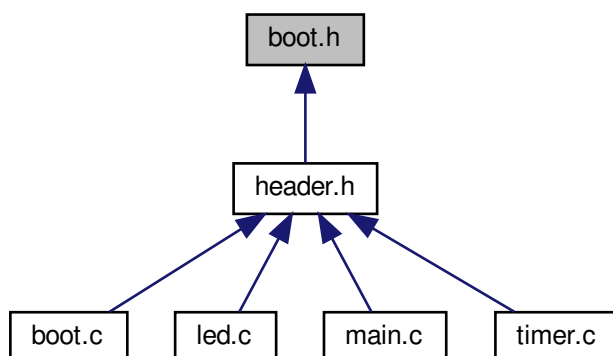
Returns

none.

7.296 boot.h File Reference

Demo program bootloader interface header file.

This graph shows which files directly or indirectly include this file:



Functions

- void [BootComInit](#) (void)
Initializes the communication interface.
- void [BootComCheckActivationRequest](#) (void)
Receives the `CONNECT` request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootActivate](#) (void)
Bootloader activation function.

7.296.1 Detailed Description

Demo program bootloader interface header file.

7.296.2 Function Documentation

7.296.2.1 [BootActivate\(\)](#)

```
void BootActivate (  
    void )
```

Bootloader activation function.

Returns

none.

7.296.2.2 [BootComCheckActivationRequest\(\)](#)

```
void BootComCheckActivationRequest (  
    void )
```

Receives the `CONNECT` request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.296.2.3 BootComInit()

```
void BootComInit (
    void )
```

Initializes the communication interface.

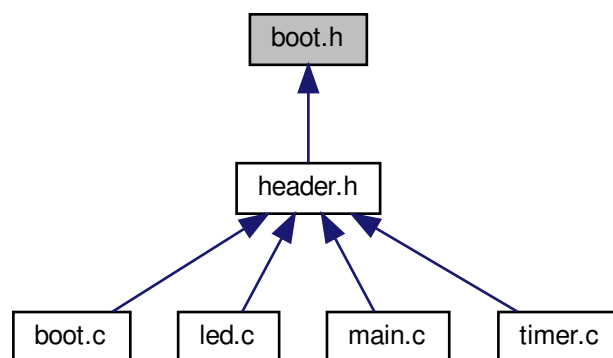
Returns

none.

7.297 boot.h File Reference

Demo program bootloader interface header file.

This graph shows which files directly or indirectly include this file:



Functions

- void [BootComInit](#) (void)
Initializes the communication interface.
- void [BootComCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootActivate](#) (void)
Bootloader activation function.

7.297.1 Detailed Description

Demo program bootloader interface header file.

7.297.2 Function Documentation

7.297.2.1 BootActivate()

```
void BootActivate (
    void )
```

Bootloader activation function.

Returns

none.

7.297.2.2 BootComCheckActivationRequest()

```
void BootComCheckActivationRequest (
    void )
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.297.2.3 BootComInit()

```
void BootComInit (
    void )
```

Initializes the communication interface.

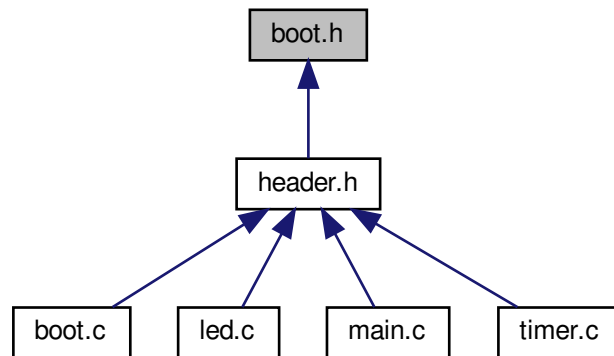
Returns

none.

7.298 boot.h File Reference

Demo program bootloader interface header file.

This graph shows which files directly or indirectly include this file:



Functions

- void [BootComInit](#) (void)
Initializes the communication interface.
- void [BootComCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootActivate](#) (void)
Bootloader activation function.

7.298.1 Detailed Description

Demo program bootloader interface header file.

7.298.2 Function Documentation

7.298.2.1 BootActivate()

```
void BootActivate (  
    void )
```

Bootloader activation function.

Returns

none.

7.298.2.2 BootComCheckActivationRequest()

```
void BootComCheckActivationRequest (  
    void )
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.298.2.3 BootComInit()

```
void BootComInit (  
    void )
```

Initializes the communication interface.

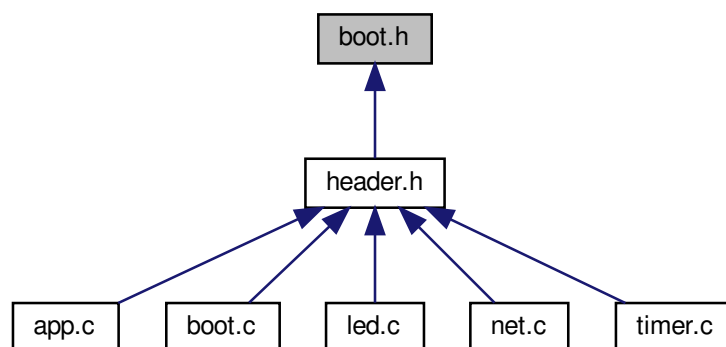
Returns

none.

7.299 boot.h File Reference

Demo program bootloader interface header file.

This graph shows which files directly or indirectly include this file:



Functions

- void [BootComInit](#) (void)
Initializes the communication interface.
- void [BootComCheckActivationRequest](#) (void)
Receives the `CONNECT` request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootActivate](#) (void)
Bootloader activation function.

7.299.1 Detailed Description

Demo program bootloader interface header file.

7.299.2 Function Documentation

7.299.2.1 [BootActivate\(\)](#)

```
void BootActivate (  
    void )
```

Bootloader activation function.

Returns

none.

7.299.2.2 [BootComCheckActivationRequest\(\)](#)

```
void BootComCheckActivationRequest (  
    void )
```

Receives the `CONNECT` request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.299.2.3 BootComInit()

```
void BootComInit (
    void )
```

Initializes the communication interface.

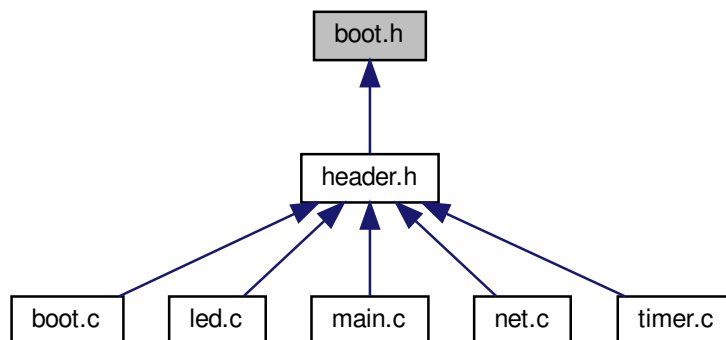
Returns

none.

7.300 boot.h File Reference

Demo program bootloader interface header file.

This graph shows which files directly or indirectly include this file:



Functions

- void [BootComInit](#) (void)
Initializes the communication interface.
- void [BootComCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootActivate](#) (void)
Bootloader activation function.

7.300.1 Detailed Description

Demo program bootloader interface header file.

7.300.2 Function Documentation

7.300.2.1 BootActivate()

```
void BootActivate (
    void )
```

Bootloader activation function.

Returns

none.

7.300.2.2 BootComCheckActivationRequest()

```
void BootComCheckActivationRequest (
    void )
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.300.2.3 BootComInit()

```
void BootComInit (
    void )
```

Initializes the communication interface.

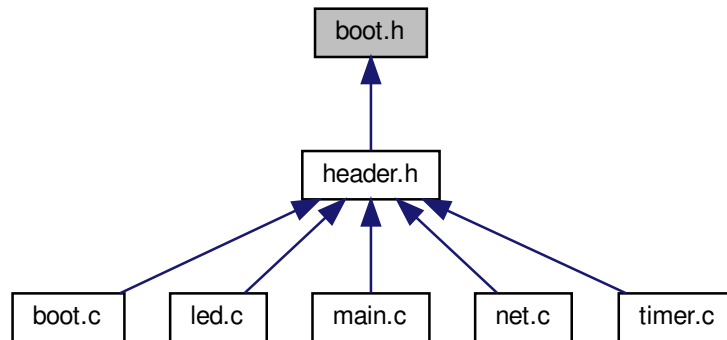
Returns

none.

7.301 boot.h File Reference

Demo program bootloader interface header file.

This graph shows which files directly or indirectly include this file:



Functions

- void [BootComInit](#) (void)
Initializes the communication interface.
- void [BootComCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootActivate](#) (void)
Bootloader activation function.

7.301.1 Detailed Description

Demo program bootloader interface header file.

7.301.2 Function Documentation

7.301.2.1 BootActivate()

```
void BootActivate (  
    void )
```

Bootloader activation function.

Returns

none.

7.301.2.2 BootComCheckActivationRequest()

```
void BootComCheckActivationRequest (  
    void )
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.301.2.3 BootComInit()

```
void BootComInit (  
    void )
```

Initializes the communication interface.

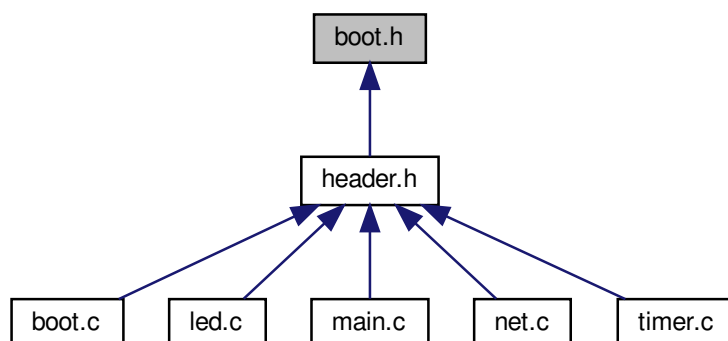
Returns

none.

7.302 boot.h File Reference

Demo program bootloader interface header file.

This graph shows which files directly or indirectly include this file:



Functions

- void [BootComInit](#) (void)
Initializes the communication interface.
- void [BootComCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootActivate](#) (void)
Bootloader activation function.

7.302.1 Detailed Description

Demo program bootloader interface header file.

7.302.2 Function Documentation

7.302.2.1 [BootActivate\(\)](#)

```
void BootActivate (  
    void )
```

Bootloader activation function.

Returns

none.

7.302.2.2 [BootComCheckActivationRequest\(\)](#)

```
void BootComCheckActivationRequest (  
    void )
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.302.2.3 BootComInit()

```
void BootComInit (
    void )
```

Initializes the communication interface.

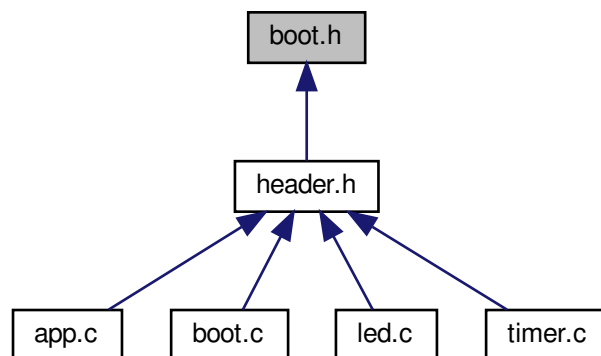
Returns

none.

7.303 boot.h File Reference

Demo program bootloader interface header file.

This graph shows which files directly or indirectly include this file:



Functions

- void [BootComInit](#) (void)
Initializes the communication interface.
- void [BootComCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootActivate](#) (void)
Bootloader activation function.

7.303.1 Detailed Description

Demo program bootloader interface header file.

7.303.2 Function Documentation

7.303.2.1 BootActivate()

```
void BootActivate (
    void )
```

Bootloader activation function.

Returns

none.

7.303.2.2 BootComCheckActivationRequest()

```
void BootComCheckActivationRequest (
    void )
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.303.2.3 BootComInit()

```
void BootComInit (
    void )
```

Initializes the communication interface.

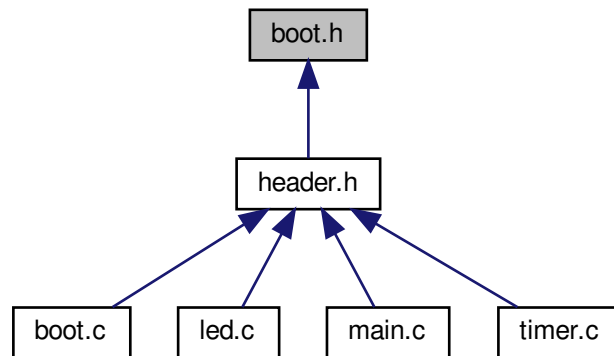
Returns

none.

7.304 boot.h File Reference

Demo program bootloader interface header file.

This graph shows which files directly or indirectly include this file:



Functions

- void [BootComInit](#) (void)
Initializes the communication interface.
- void [BootComCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootActivate](#) (void)
Bootloader activation function.

7.304.1 Detailed Description

Demo program bootloader interface header file.

7.304.2 Function Documentation

7.304.2.1 BootActivate()

```
void BootActivate (  
    void )
```

Bootloader activation function.

Returns

none.

7.304.2.2 BootComCheckActivationRequest()

```
void BootComCheckActivationRequest (
    void )
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.304.2.3 BootComInit()

```
void BootComInit (
    void )
```

Initializes the communication interface.

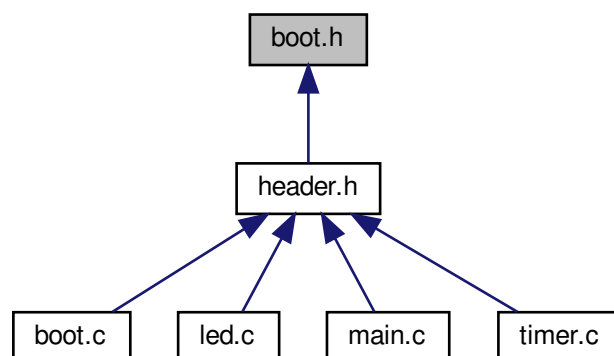
Returns

none.

7.305 boot.h File Reference

Demo program bootloader interface header file.

This graph shows which files directly or indirectly include this file:



Functions

- void [BootComInit](#) (void)
Initializes the communication interface.
- void [BootComCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootActivate](#) (void)
Bootloader activation function.

7.305.1 Detailed Description

Demo program bootloader interface header file.

7.305.2 Function Documentation

7.305.2.1 BootActivate()

```
void BootActivate (  
    void )
```

Bootloader activation function.

Returns

none.

7.305.2.2 BootComCheckActivationRequest()

```
void BootComCheckActivationRequest (  
    void )
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.305.2.3 BootComInit()

```
void BootComInit (
    void )
```

Initializes the communication interface.

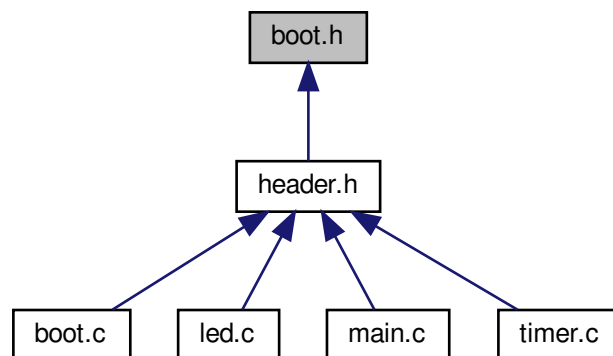
Returns

none.

7.306 boot.h File Reference

Demo program bootloader interface header file.

This graph shows which files directly or indirectly include this file:



Functions

- void [BootComInit](#) (void)
Initializes the communication interface.
- void [BootComCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootActivate](#) (void)
Bootloader activation function.

7.306.1 Detailed Description

Demo program bootloader interface header file.

7.306.2 Function Documentation

7.306.2.1 BootActivate()

```
void BootActivate (
    void )
```

Bootloader activation function.

Returns

none.

7.306.2.2 BootComCheckActivationRequest()

```
void BootComCheckActivationRequest (
    void )
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

7.306.2.3 BootComInit()

```
void BootComInit (
    void )
```

Initializes the communication interface.

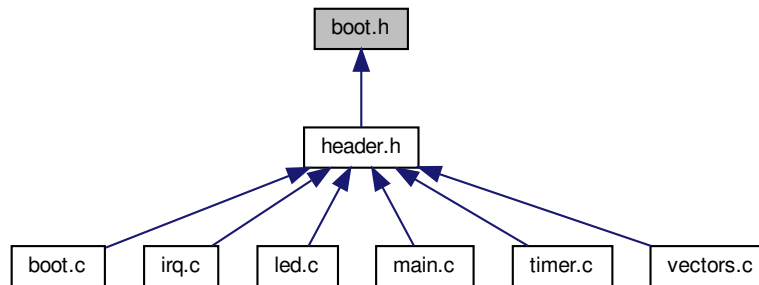
Returns

none.

7.307 boot.h File Reference

Demo program bootloader interface header file.

This graph shows which files directly or indirectly include this file:



Functions

- void [BootComInit](#) (void)
Initializes the communication interface.
- void [BootComCheckActivationRequest](#) (void)
Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.
- void [BootActivate](#) (void)
Bootloader activation function.

7.307.1 Detailed Description

Demo program bootloader interface header file.

7.307.2 Function Documentation

7.307.2.1 BootActivate()

```
void BootActivate (
    void )
```

Bootloader activation function.

Returns

none.

Referenced by [BootComCanCheckActivationRequest\(\)](#), [BootComRs232CheckActivationRequest\(\)](#), and [NetApp\(\)](#).

7.307.2.2 BootComCheckActivationRequest()

```
void BootComCheckActivationRequest (  
    void )
```

Receives the CONNECT request from the host, which indicates that the bootloader should be activated and, if so, activates it.

Returns

none.

Referenced by AppTask(), and main().

7.307.2.3 BootComInit()

```
void BootComInit (  
    void )
```

Initializes the communication interface.

Returns

none.

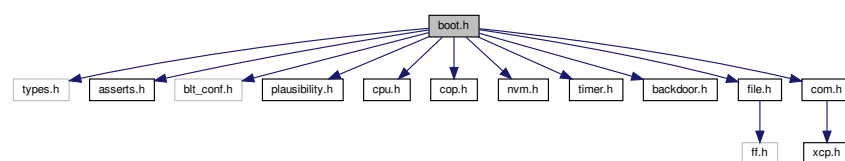
Referenced by Applnit(), and main().

7.308 boot.h File Reference

Bootloader core module header file.

```
#include "types.h"  
#include "asserts.h"  
#include "blt_conf.h"  
#include "plausibility.h"  
#include "cpu.h"  
#include "cop.h"  
#include "nvm.h"  
#include "timer.h"  
#include "backdoor.h"  
#include "file.h"  
#include "com.h"
```

Include dependency graph for Source/boot.h:



Macros

- `#define BOOT_VERSION_CORE_MAIN (1u)`
Main version number of the bootloader core.
- `#define BOOT_VERSION_CORE_MINOR (12u)`
Minor version number of the bootloader core.
- `#define BOOT_VERSION_CORE_PATCH (1u)`
Patch number of the bootloader core.

Functions

- `void BootInit (void)`
Initializes the bootloader core.
- `void BootTask (void)`
Task function of the bootloader core that drives the program.

7.308.1 Detailed Description

Bootloader core module header file.

7.308.2 Function Documentation

7.308.2.1 BootInit()

```
void BootInit (  
    void )
```

Initializes the bootloader core.

Returns

none

Referenced by `Applnit()`, and `main()`.

7.308.2.2 BootTask()

```
void BootTask (  
    void )
```

Task function of the bootloader core that drives the program.

Returns

none

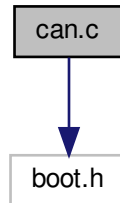
Referenced by `AppTask()`, and `main()`.

7.309 can.c File Reference

Bootloader CAN communication interface source file.

```
#include "boot.h"
```

Include dependency graph for _template/can.c:



Data Structures

- struct [tCanBusTiming](#)
Structure type for grouping CAN bus timing related information.

Macros

- #define [CAN_MSG_TX_TIMEOUT_MS](#) (50u)
Timeout for transmitting a CAN message in milliseconds.

Functions

- static [blt_bool](#) [CanGetSpeedConfig](#) ([blt_int16u](#) baud, [blt_int16u](#) *prescaler, [blt_int8u](#) *tseg1, [blt_int8u](#) *tseg2)
Search algorithm to match the desired baudrate to a possible bus timing configuration.
- void [CanInit](#) (void)
Initializes the CAN controller and synchronizes it to the CAN bus.
- void [CanTransmitPacket](#) ([blt_int8u](#) *data, [blt_int8u](#) len)
Transmits a packet formatted for the communication interface.
- [blt_bool](#) [CanReceivePacket](#) ([blt_int8u](#) *data, [blt_int8u](#) *len)
Receives a communication interface packet if one is present.

Variables

- static const [tCanBusTiming](#) [canTiming](#) []
CAN bittiming table for dynamically calculating the bittiming settings.

7.309.1 Detailed Description

Bootloader CAN communication interface source file.

7.309.2 Function Documentation

7.309.2.1 CanGetSpeedConfig()

```
static blt_bool CanGetSpeedConfig (
    blt_int16u baud,
    blt_int16u * prescaler,
    blt_int8u * tseg1,
    blt_int8u * tseg2 ) [static]
```

Search algorithm to match the desired baudrate to a possible bus timing configuration.

Parameters

<i>baud</i>	The desired baudrate in kbps. Valid values are 10..1000.
<i>prescaler</i>	Pointer to where the value for the prescaler will be stored.
<i>tseg1</i>	Pointer to where the value for TSEG2 will be stored.
<i>tseg2</i>	Pointer to where the value for TSEG2 will be stored.

Returns

BLT_TRUE if the CAN bustiming register values were found, BLT_FALSE otherwise.

Referenced by CanInit().

7.309.2.2 CanInit()

```
void CanInit (
    void )
```

Initializes the CAN controller and synchronizes it to the CAN bus.

Returns

none.

Referenced by ComInit().

7.309.2.3 CanReceivePacket()

```
blt_bool CanReceivePacket (
    blt_int8u * data,
    blt_int8u * len )
```

Receives a communication interface packet if one is present.

Parameters

<i>data</i>	Pointer to byte array where the data is to be stored.
<i>len</i>	Pointer where the length of the packet is to be stored.

Returns

BLT_TRUE is a packet was received, BLT_FALSE otherwise.

Referenced by ComTask().

7.309.2.4 CanTransmitPacket()

```
void CanTransmitPacket (
    blt_int8u * data,
    blt_int8u len )
```

Transmits a packet formatted for the communication interface.

Parameters

<i>data</i>	Pointer to byte array with data that it to be transmitted.
<i>len</i>	Number of bytes that are to be transmitted.

Returns

none.

Referenced by ComTransmitPacket().

7.309.3 Variable Documentation**7.309.3.1 canTiming**

```
const tCanBusTiming canTiming[] [static]
```

Initial value:

```
=
{
    { 5, 2 },
    { 6, 2 },
    { 6, 3 },
    { 7, 3 },
```

```

{ 8, 3 },
{ 9, 3 },
{ 9, 4 },
{ 10, 4 },
{ 11, 4 },
{ 12, 4 },
{ 12, 5 },
{ 13, 5 },
{ 14, 5 },
{ 15, 5 },
{ 15, 6 },
{ 16, 6 },
{ 16, 7 },
{ 16, 8 }
}

```

CAN bittiming table for dynamically calculating the bittiming settings.

According to the CAN protocol 1 bit-time can be made up of between 8..25 time quanta (TQ). The total TQ in a bit is SYNC + TSEG1 + TSEG2 with SYNC always being 1. The sample point is (SYNC + TSEG1) / (SYNC + TSEG1 + SEG2) * 100%. This array contains possible and valid time quanta configurations with a sample point between 68..78%.

Referenced by CanGetSpeedConfig().

7.310 can.c File Reference

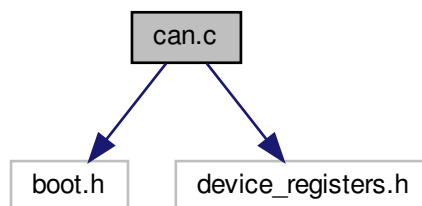
Bootloader CAN communication interface source file.

```

#include "boot.h"
#include "device_registers.h"

```

Include dependency graph for ARMCM0_S32K11/can.c:



Data Structures

- struct [tCanBusTiming](#)

Structure type for grouping CAN bus timing related information.

Macros

- `#define CAN_INIT_TIMEOUT_MS (250U)`
Timeout for entering/leaving CAN initialization mode in milliseconds.
- `#define CAN_MSG_TX_TIMEOUT_MS (50U)`
Timeout for transmitting a CAN message in milliseconds.
- `#define CANx (CAN0)`
Set the peripheral CAN0 base pointer.
- `#define PCC_FlexCANx_INDEX (PCC_FlexCAN0_INDEX)`
Set the PCC index offset for CAN0.
- `#define CANx_MAX_MB_NUM (FEATURE_CAN0_MAX_MB_NUM)`
Set the number of message boxes supported by CAN0.
- `#define CAN_TX_MSGBOX_NUM (8U)`
The mailbox used for transmitting the XCP respond message.
- `#define CAN_RX_MSGBOX_NUM (9U)`
The mailbox used for receiving the XCP command message.

Functions

- static `blt_bool CanGetSpeedConfig (blt_int16u baud, blt_int16u *prescaler, tCanBusTiming *busTimingCfg)`
Search algorithm to match the desired baudrate to a possible bus timing configuration.
- static void `CanFreezeModeEnter (void)`
Places the CAN controller in freeze mode. Note that the CAN controller can only be placed in freeze mode, if it is actually enabled.
- static void `CanFreezeModeExit (void)`
Leaves the CAN controller's freeze mode. Note that this operation can only be done, if it is actually enabled.
- static void `CanDisabledModeEnter (void)`
Places the CAN controller in disabled mode.
- static void `CanDisabledModeExit (void)`
Places the CAN controller in enabled mode.
- void `CanInit (void)`
Initializes the CAN controller and synchronizes it to the CAN bus.
- void `CanTransmitPacket (blt_int8u *data, blt_int8u len)`
Transmits a packet formatted for the communication interface.
- `blt_bool CanReceivePacket (blt_int8u *data, blt_int8u *len)`
Receives a communication interface packet if one is present.

Variables

- static const `tCanBusTiming canTiming []`
CAN bit timing table for dynamically calculating the bittiming settings.
- static volatile `blt_int32u dummyTimerVal`
Dummy variable to store the CAN controller's free running timer value in. This is needed at the end of a CAN message reception to unlock the mailbox again. If this variable is declared locally within the function, it generates an unwanted compiler warning about assigning a value and not using it. For this reason this dummy variable is declared here as a module global.

7.310.1 Detailed Description

Bootloader CAN communication interface source file.

7.310.2 Function Documentation

7.310.2.1 CanDisabledModeEnter()

```
static void CanDisabledModeEnter (  
    void ) [static]
```

Places the CAN controller in disabled mode.

Returns

none.

Referenced by CanInit().

7.310.2.2 CanDisabledModeExit()

```
static void CanDisabledModeExit (  
    void ) [static]
```

Places the CAN controller in enabled mode.

Returns

none.

Referenced by CanInit().

7.310.2.3 CanFreezeModeEnter()

```
static void CanFreezeModeEnter (  
    void ) [static]
```

Places the CAN controller in freeze mode. Note that the CAN controller can only be placed in freeze mode, if it is actually enabled.

Returns

none.

Referenced by CanInit().

7.310.2.4 CanFreezeModeExit()

```
static void CanFreezeModeExit (
    void ) [static]
```

Leaves the CAN controller's freeze mode. Note that this operation can only be done, if it is actually enabled.

Returns

none.

Referenced by CanInit().

7.310.2.5 CanGetSpeedConfig()

```
static blt_bool CanGetSpeedConfig (
    blt_int16u baud,
    blt_int16u * prescaler,
    tCanBusTiming * busTimingCfg ) [static]
```

Search algorithm to match the desired baudrate to a possible bus timing configuration.

Parameters

<i>baud</i>	The desired baudrate in kbps. Valid values are 10..1000.
<i>prescaler</i>	Pointer to where the value for the prescaler will be stored.
<i>busTimingCfg</i>	Pointer to where the bus timing values will be stored.

Returns

BLT_TRUE if the CAN bustiming register values were found, BLT_FALSE otherwise.

Referenced by CanInit().

7.310.2.6 CanInit()

```
void CanInit (
    void )
```

Initializes the CAN controller and synchronizes it to the CAN bus.

Returns

none.

7.310.2.7 CanReceivePacket()

```
blt_bool CanReceivePacket (
    blt_int8u * data,
    blt_int8u * len )
```

Receives a communication interface packet if one is present.

Parameters

<i>data</i>	Pointer to byte array where the data is to be stored.
<i>len</i>	Pointer where the length of the packet is to be stored.

Returns

BLT_TRUE is a packet was received, BLT_FALSE otherwise.

7.310.2.8 CanTransmitPacket()

```
void CanTransmitPacket (
    blt_int8u * data,
    blt_int8u len )
```

Transmits a packet formatted for the communication interface.

Parameters

<i>data</i>	Pointer to byte array with data that it to be transmitted.
<i>len</i>	Number of bytes that are to be transmitted.

Returns

none.

7.310.3 Variable Documentation

7.310.3.1 canTiming

```
const tCanBusTiming canTiming[] [static]
```

Initial value:

```

=
{
    { 8U, 3U, 2U, 2U },
    { 9U, 3U, 3U, 2U },
    { 10U, 3U, 3U, 3U },
    { 11U, 4U, 3U, 3U },
    { 12U, 4U, 4U, 3U },
    { 13U, 5U, 4U, 3U },
    { 14U, 5U, 4U, 4U },
    { 15U, 6U, 4U, 4U },
    { 16U, 6U, 5U, 4U },
    { 17U, 7U, 5U, 4U },
    { 18U, 7U, 5U, 5U },
    { 19U, 8U, 5U, 5U },
    { 20U, 8U, 6U, 5U },
    { 21U, 8U, 7U, 5U },
    { 22U, 8U, 7U, 6U },
    { 23U, 8U, 8U, 6U },
    { 24U, 8U, 8U, 7U },
    { 25U, 8U, 8U, 8U }
}

```

CAN bit timing table for dynamically calculating the bittiming settings.

According to the CAN protocol 1 bit-time can be made up of between 8..25 time quanta (TQ). The total TQ in a bit is SYNC + TSEG1 + TSEG2 with SYNC always being 1. The sample point is (SYNC + TSEG1) / (SYNC + TSEG1 + TSEG2)

- 100%. This array contains possible and valid time quanta configurations with a sample point between 68..78%. A visual representation of the TQ in a bit is: | SYNCSEG | TIME1SEG | TIME2SEG | Or with an alternative representation: | SYNCSEG | PROPSEG | PHASE1SEG | PHASE2SEG | With the alternative representation TIME1SEG = PROPSEG + PHASE1SEG.

Referenced by CanGetSpeedConfig().

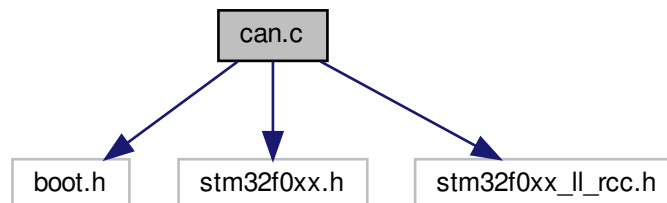
7.311 can.c File Reference

Bootloader CAN communication interface source file.

```

#include "boot.h"
#include "stm32f0xx.h"
#include "stm32f0xx_ll_rcc.h"
Include dependency graph for ARMCM0_STM32F0/can.c:

```



Data Structures

- struct [tCanBusTiming](#)
Structure type for grouping CAN bus timing related information.

Macros

- #define [CAN_MSG_TX_TIMEOUT_MS](#) (50u)
Timeout for transmitting a CAN message in milliseconds.
- #define [CAN_CHANNEL](#) CAN
Set CAN base address to CAN1.

Functions

- static [blt_bool](#) [CanGetSpeedConfig](#) ([blt_int16u](#) baud, [blt_int16u](#) *prescaler, [blt_int8u](#) *tseg1, [blt_int8u](#) *tseg2)
Search algorithm to match the desired baudrate to a possible bus timing configuration.
- void [CanInit](#) (void)
Initializes the CAN controller and synchronizes it to the CAN bus.
- void [CanTransmitPacket](#) ([blt_int8u](#) *data, [blt_int8u](#) len)
Transmits a packet formatted for the communication interface.
- [blt_bool](#) [CanReceivePacket](#) ([blt_int8u](#) *data, [blt_int8u](#) *len)
Receives a communication interface packet if one is present.

Variables

- static const [tCanBusTiming](#) [canTiming](#) []
CAN bittiming table for dynamically calculating the bittiming settings.
- static CAN_HandleTypeDef [canHandle](#)
CAN handle to be used in API calls.

7.311.1 Detailed Description

Bootloader CAN communication interface source file.

7.311.2 Function Documentation

7.311.2.1 CanGetSpeedConfig()

```
static blt\_bool CanGetSpeedConfig (
    blt\_int16u baud,
    blt\_int16u * prescaler,
    blt\_int8u * tseg1,
    blt\_int8u * tseg2 ) [static]
```

Search algorithm to match the desired baudrate to a possible bus timing configuration.

Parameters

<i>baud</i>	The desired baudrate in kbps. Valid values are 10..1000.
<i>prescaler</i>	Pointer to where the value for the prescaler will be stored.
<i>tseg1</i>	Pointer to where the value for TSEG2 will be stored.
<i>tseg2</i>	Pointer to where the value for TSEG2 will be stored.

Returns

BLT_TRUE if the CAN bustiming register values were found, BLT_FALSE otherwise.

Referenced by CanInit().

7.311.2.2 CanInit()

```
void CanInit (
    void )
```

Initializes the CAN controller and synchronizes it to the CAN bus.

Returns

none.

7.311.2.3 CanReceivePacket()

```
blt_bool CanReceivePacket (
    blt_int8u * data,
    blt_int8u * len )
```

Receives a communication interface packet if one is present.

Parameters

<i>data</i>	Pointer to byte array where the data is to be stored.
<i>len</i>	Pointer where the length of the packet is to be stored.

Returns

BLT_TRUE is a packet was received, BLT_FALSE otherwise.

7.311.2.4 CanTransmitPacket()

```
void CanTransmitPacket (
    blt_int8u * data,
    blt_int8u len )
```

Transmits a packet formatted for the communication interface.

Parameters

<i>data</i>	Pointer to byte array with data that it to be transmitted.
<i>len</i>	Number of bytes that are to be transmitted.

Returns

none.

7.311.3 Variable Documentation

7.311.3.1 canTiming

```
const tCanBusTiming canTiming[] [static]
```

Initial value:

```
=
{
    { 5, 2 },
    { 6, 2 },
    { 6, 3 },
    { 7, 3 },
    { 8, 3 },
    { 9, 3 },
    { 9, 4 },
    { 10, 4 },
    { 11, 4 },
    { 12, 4 },
    { 12, 5 },
    { 13, 5 },
    { 14, 5 },
    { 15, 5 },
    { 15, 6 },
    { 16, 6 },
    { 16, 7 },
    { 16, 8 }
}
```

CAN bittiming table for dynamically calculating the bittiming settings.

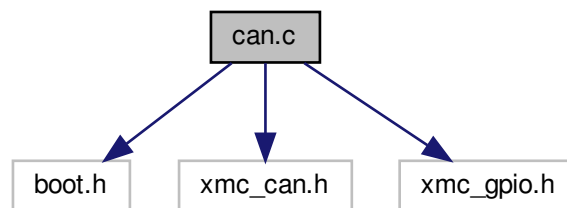
According to the CAN protocol 1 bit-time can be made up of between 8..25 time quanta (TQ). The total TQ in a bit is SYNC + TSEG1 + TSEG2 with SYNC always being 1. The sample point is (SYNC + TSEG1) / (SYNC + TSEG1 + SEG2) * 100%. This array contains possible and valid time quanta configurations with a sample point between 68..78%.

Referenced by CanGetSpeedConfig().

7.312 can.c File Reference

Bootloader CAN communication interface source file.

```
#include "boot.h"
#include "xmc_can.h"
#include "xmc_gpio.h"
Include dependency graph for ARMCM0_XMC1/can.c:
```



Macros

- `#define CAN_MSG_TX_TIMEOUT_MS (50u)`
Timeout for transmitting a CAN message in milliseconds.
- `#define CAN_CHANNEL ((CAN_NODE_TypeDef *) (canChannelMap[BOOT_COM_CAN_CHANNEL_INDEX]))`
Macro for accessing the CAN channel handle in the format that is expected by the XMCLib CAN driver.
- `#define CAN_TX_MSBOBJ (CAN_MO0)`
Message object dedicated to message transmission.
- `#define CAN_TX_MSBOBJ_IDX (0)`
Index of the message object dedicated to message transmission.
- `#define CAN_RX_MSBOBJ (CAN_MO1)`
Message object dedicated to message reception.
- `#define CAN_RX_MSBOBJ_IDX (1)`
Index of the message object dedicated to message reception.

Functions

- `void CanInit (void)`
Initializes the CAN controller and synchronizes it to the CAN bus.
- `void CanTransmitPacket (blt_int8u *data, blt_int8u len)`
Transmits a packet formatted for the communication interface.
- `blt_bool CanReceivePacket (blt_int8u *data, blt_int8u *len)`
Receives a communication interface packet if one is present.

Variables

- static const CAN_NODE_TypeDef * [canChannelMap](#) []
Helper array to quickly convert the channel index, as specific in the boot- loader's configuration header, to the associated channel handle that the XMCLib's CAN driver requires.
- static XMC_CAN_MO_t [transmitMsgObj](#)
Transmit message object data structure.
- static XMC_CAN_MO_t [receiveMsgObj](#)
Receive message object data structure.

7.312.1 Detailed Description

Bootloader CAN communication interface source file.

7.312.2 Function Documentation

7.312.2.1 CanInit()

```
void CanInit (
    void )
```

Initializes the CAN controller and synchronizes it to the CAN bus.

Returns

none.

7.312.2.2 CanReceivePacket()

```
blt_bool CanReceivePacket (
    blt_int8u * data,
    blt_int8u * len )
```

Receives a communication interface packet if one is present.

Parameters

<i>data</i>	Pointer to byte array where the data is to be stored.
<i>len</i>	Pointer where the length of the packet is to be stored.

Returns

BLT_TRUE is a packet was received, BLT_FALSE otherwise.

7.312.2.3 CanTransmitPacket()

```
void CanTransmitPacket (
    blt_int8u * data,
    blt_int8u len )
```

Transmits a packet formatted for the communication interface.

Parameters

<i>data</i>	Pointer to byte array with data that it to be transmitted.
<i>len</i>	Number of bytes that are to be transmitted.

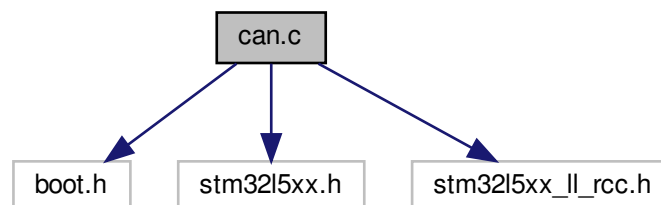
Returns

none.

7.313 can.c File Reference

Bootloader CAN communication interface source file.

```
#include "boot.h"
#include "stm32l5xx.h"
#include "stm32l5xx_ll_rcc.h"
Include dependency graph for ARMCM33_STM32L5/can.c:
```



Data Structures

- struct [tCanBusTiming](#)

Structure type for grouping CAN bus timing related information.

Macros

- `#define CAN_MSG_TX_TIMEOUT_MS` (50u)
Timeout for transmitting a CAN message in milliseconds.
- `#define CAN_CHANNEL` FDCAN1
Set CAN base address to CAN1.

Functions

- static `blt_bool CanGetSpeedConfig` (`blt_int16u` baud, `blt_int16u` *prescaler, `blt_int8u` *tseg1, `blt_int8u` *tseg2)
Search algorithm to match the desired baudrate to a possible bus timing configuration.
- void `CanInit` (void)
Initializes the CAN controller and synchronizes it to the CAN bus.
- void `CanTransmitPacket` (`blt_int8u` *data, `blt_int8u` len)
Transmits a packet formatted for the communication interface.
- `blt_bool CanReceivePacket` (`blt_int8u` *data, `blt_int8u` *len)
Receives a communication interface packet if one is present.

Variables

- static const `tCanBusTiming` canTiming []
CAN bittiming table for dynamically calculating the bittiming settings.
- static FDCAN_HandleTypeDef `canHandle`
CAN handle to be used in API calls.

7.313.1 Detailed Description

Bootloader CAN communication interface source file.

7.313.2 Function Documentation

7.313.2.1 CanGetSpeedConfig()

```
static blt_bool CanGetSpeedConfig (
    blt_int16u baud,
    blt_int16u * prescaler,
    blt_int8u * tseg1,
    blt_int8u * tseg2 ) [static]
```

Search algorithm to match the desired baudrate to a possible bus timing configuration.

Parameters

<i>baud</i>	The desired baudrate in kbps. Valid values are 10..1000.
<i>prescaler</i>	Pointer to where the value for the prescaler will be stored.
<i>tseg1</i>	Pointer to where the value for TSEG2 will be stored.
<i>tseg2</i>	Pointer to where the value for TSEG2 will be stored.

Returns

BLT_TRUE if the CAN bustiming register values were found, BLT_FALSE otherwise.

Referenced by CanInit().

7.313.2.2 CanInit()

```
void CanInit (
    void )
```

Initializes the CAN controller and synchronizes it to the CAN bus.

Returns

none.

7.313.2.3 CanReceivePacket()

```
blt_bool CanReceivePacket (
    blt_int8u * data,
    blt_int8u * len )
```

Receives a communication interface packet if one is present.

Parameters

<i>data</i>	Pointer to byte array where the data is to be stored.
<i>len</i>	Pointer where the length of the packet is to be stored.

Returns

BLT_TRUE if a packet was received, BLT_FALSE otherwise.

7.313.2.4 CanTransmitPacket()

```
void CanTransmitPacket (
    blt_int8u * data,
    blt_int8u len )
```

Transmits a packet formatted for the communication interface.

Parameters

<i>data</i>	Pointer to byte array with data that it to be transmitted.
<i>len</i>	Number of bytes that are to be transmitted.

Returns

none.

7.313.3 Variable Documentation**7.313.3.1 canTiming**

```
const tCanBusTiming canTiming[] [static]
```

Initial value:

```
=  
{  
  
    { 5, 2 },  
    { 6, 2 },  
    { 6, 3 },  
    { 7, 3 },  
    { 8, 3 },  
    { 9, 3 },  
    { 9, 4 },  
    { 10, 4 },  
    { 11, 4 },  
    { 12, 4 },  
    { 12, 5 },  
    { 13, 5 },  
    { 14, 5 },  
    { 15, 5 },  
    { 15, 6 },  
    { 16, 6 },  
    { 16, 7 },  
    { 16, 8 }  
}
```

CAN bittiming table for dynamically calculating the bittiming settings.

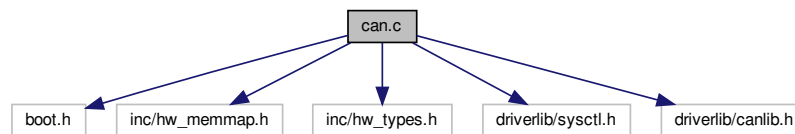
According to the CAN protocol 1 bit-time can be made up of between 8..25 time quanta (TQ). The total TQ in a bit is SYNC + TSEG1 + TSEG2 with SYNC always being 1. The sample point is (SYNC + TSEG1) / (SYNC + TSEG1 + SEG2) * 100%. This array contains possible and valid time quanta configurations with a sample point between 68..78%.

Referenced by CanGetSpeedConfig().

7.314 can.c File Reference

Bootloader CAN communication interface source file.

```
#include "boot.h"
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "driverlib/sysctl.h"
#include "driverlib/canlib.h"
Include dependency graph for ARMCM3_LM3S/can.c:
```



Macros

- `#define CAN_MSG_TX_TIMEOUT_MS (50u)`
Timeout for transmitting a CAN message in milliseconds.
- `#define CAN_RX_MSGOBJECT_IDX (0)`
Index of the used reception message object.
- `#define CAN_TX_MSGOBJECT_IDX (1)`
Index of the used transmission message object.

Functions

- static `blt_int8u CanSetBittiming (void)`
*Attempts to match the bittiming parameters to the requested baudrate for a sample point between 65 and 75%, through a linear search algorithm. It is based on the equation: $\text{baudrate} = \text{CAN Clock Freq} / ((1 + \text{PropSeg} + \text{Phase1} \leftrightarrow \text{Seg} + \text{Phase2Seg}) * \text{Prescaler})$*
- void `CanInit (void)`
Initializes the CAN controller and synchronizes it to the CAN bus.
- void `CanTransmitPacket (blt_int8u *data, blt_int8u len)`
Transmits a packet formatted for the communication interface.
- `blt_bool CanReceivePacket (blt_int8u *data, blt_int8u *len)`
Receives a communication interface packet if one is present.

Variables

- static const `blt_int16u canBitNum2Mask []`
Lookup table to quickly and efficiently convert a bit number to a bit mask.

7.314.1 Detailed Description

Bootloader CAN communication interface source file.

7.314.2 Function Documentation

7.314.2.1 CanInit()

```
void CanInit (
    void )
```

Initializes the CAN controller and synchronizes it to the CAN bus.

Returns

none.

7.314.2.2 CanReceivePacket()

```
blt_bool CanReceivePacket (
    blt_int8u * data,
    blt_int8u * len )
```

Receives a communication interface packet if one is present.

Parameters

<i>data</i>	Pointer to byte array where the data is to be stored.
<i>len</i>	Pointer where the length of the packet is to be stored.

Returns

BLT_TRUE is a packet was received, BLT_FALSE otherwise.

7.314.2.3 CanSetBittiming()

```
static blt_int8u CanSetBittiming (
    void ) [static]
```

Attempts to match the bittiming parameters to the requested baudrate for a sample point between 65 and 75%, through a linear search algorithm. It is based on the equation: baudrate = CAN Clock Freq/((1+PropSeg+Phase1↔Seg+Phase2Seg)*Prescaler)

Returns

BLT_TRUE if a valid bittiming configuration was found and set. BLT_FALSE otherwise.

Referenced by CanInit().

7.314.2.4 CanTransmitPacket()

```
void CanTransmitPacket (
    blt_int8u * data,
    blt_int8u len )
```

Transmits a packet formatted for the communication interface.

Parameters

<i>data</i>	Pointer to byte array with data that it to be transmitted.
<i>len</i>	Number of bytes that are to be transmitted.

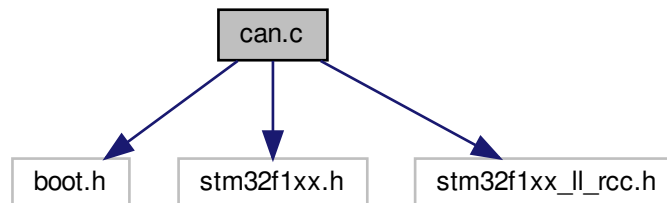
Returns

none.

7.315 can.c File Reference

Bootloader CAN communication interface source file.

```
#include "boot.h"
#include "stm32f1xx.h"
#include "stm32f1xx_ll_rcc.h"
Include dependency graph for ARMCM3_STM32F1/can.c:
```



Data Structures

- struct [tCanBusTiming](#)
Structure type for grouping CAN bus timing related information.

Macros

- #define [CAN_MSG_TX_TIMEOUT_MS](#) (50u)
Timeout for transmitting a CAN message in milliseconds.
- #define [CAN_CHANNEL](#) CAN1
Set CAN base address to CAN1.

Functions

- static `blt_bool CanGetSpeedConfig (blt_int16u baud, blt_int16u *prescaler, blt_int8u *tseg1, blt_int8u *tseg2)`
Search algorithm to match the desired baudrate to a possible bus timing configuration.
- void `CanInit (void)`
Initializes the CAN controller and synchronizes it to the CAN bus.
- void `CanTransmitPacket (blt_int8u *data, blt_int8u len)`
Transmits a packet formatted for the communication interface.
- `blt_bool CanReceivePacket (blt_int8u *data, blt_int8u *len)`
Receives a communication interface packet if one is present.

Variables

- static const `tCanBusTiming canTiming []`
CAN bittiming table for dynamically calculating the bittiming settings.
- static `CAN_HandleTypeDef canHandle`
CAN handle to be used in API calls.

7.315.1 Detailed Description

Bootloader CAN communication interface source file.

7.315.2 Function Documentation

7.315.2.1 CanGetSpeedConfig()

```
static blt_bool CanGetSpeedConfig (
    blt_int16u baud,
    blt_int16u * prescaler,
    blt_int8u * tseg1,
    blt_int8u * tseg2 ) [static]
```

Search algorithm to match the desired baudrate to a possible bus timing configuration.

Parameters

<i>baud</i>	The desired baudrate in kbps. Valid values are 10..1000.
<i>prescaler</i>	Pointer to where the value for the prescaler will be stored.
<i>tseg1</i>	Pointer to where the value for TSEG2 will be stored.
<i>tseg2</i>	Pointer to where the value for TSEG2 will be stored.

Returns

BLT_TRUE if the CAN bustiming register values were found, BLT_FALSE otherwise.

Referenced by CanInit().

7.315.2.2 CanInit()

```
void CanInit (
    void )
```

Initializes the CAN controller and synchronizes it to the CAN bus.

Returns

none.

7.315.2.3 CanReceivePacket()

```
blt_bool CanReceivePacket (
    blt_int8u * data,
    blt_int8u * len )
```

Receives a communication interface packet if one is present.

Parameters

<i>data</i>	Pointer to byte array where the data is to be stored.
<i>len</i>	Pointer where the length of the packet is to be stored.

Returns

BLT_TRUE is a packet was received, BLT_FALSE otherwise.

7.315.2.4 CanTransmitPacket()

```
void CanTransmitPacket (
    blt_int8u * data,
    blt_int8u len )
```

Transmits a packet formatted for the communication interface.

Parameters

<i>data</i>	Pointer to byte array with data that it to be transmitted.
<i>len</i>	Number of bytes that are to be transmitted.

Returns

none.

7.315.3 Variable Documentation**7.315.3.1 canTiming**

```
const tCanBusTiming canTiming[] [static]
```

Initial value:

```
=
{
    { 5, 2 },
    { 6, 2 },
    { 6, 3 },
    { 7, 3 },
    { 8, 3 },
    { 9, 3 },
    { 9, 4 },
    { 10, 4 },
    { 11, 4 },
    { 12, 4 },
    { 12, 5 },
    { 13, 5 },
    { 14, 5 },
    { 15, 5 },
    { 15, 6 },
    { 16, 6 },
    { 16, 7 },
    { 16, 8 }
}
```

CAN bittiming table for dynamically calculating the bittiming settings.

According to the CAN protocol 1 bit-time can be made up of between 8..25 time quanta (TQ). The total TQ in a bit is SYNC + TSEG1 + TSEG2 with SYNC always being 1. The sample point is (SYNC + TSEG1) / (SYNC + TSEG1 + SEG2) * 100%. This array contains possible and valid time quanta configurations with a sample point between 68..78%.

Referenced by CanGetSpeedConfig().

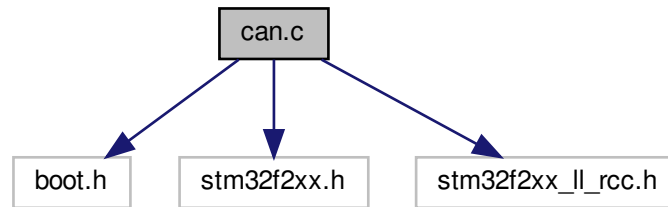
7.316 can.c File Reference

Bootloader CAN communication interface source file.

```
#include "boot.h"
#include "stm32f2xx.h"
```

```
#include "stm32f2xx_ll_rcc.h"
```

Include dependency graph for ARMCM3_STM32F2/can.c:



Data Structures

- struct [tCanBusTiming](#)
Structure type for grouping CAN bus timing related information.

Macros

- #define [CAN_MSG_TX_TIMEOUT_MS](#) (50u)
Timeout for transmitting a CAN message in milliseconds.
- #define [CAN_CHANNEL](#) CAN1
Set CAN base address to CAN1.

Functions

- static [blt_bool CanGetSpeedConfig](#) ([blt_int16u](#) baud, [blt_int16u](#) *prescaler, [blt_int8u](#) *tseg1, [blt_int8u](#) *tseg2)
Search algorithm to match the desired baudrate to a possible bus timing configuration.
- void [CanInit](#) (void)
Initializes the CAN controller and synchronizes it to the CAN bus.
- void [CanTransmitPacket](#) ([blt_int8u](#) *data, [blt_int8u](#) len)
Transmits a packet formatted for the communication interface.
- [blt_bool CanReceivePacket](#) ([blt_int8u](#) *data, [blt_int8u](#) *len)
Receives a communication interface packet if one is present.

Variables

- static const [tCanBusTiming](#) [canTiming](#) []
CAN bittiming table for dynamically calculating the bittiming settings.
- static CAN_HandleTypeDef [canHandle](#)
CAN handle to be used in API calls.

7.316.1 Detailed Description

Bootloader CAN communication interface source file.

7.316.2 Function Documentation

7.316.2.1 CanGetSpeedConfig()

```
static blt_bool CanGetSpeedConfig (
    blt_int16u baud,
    blt_int16u * prescaler,
    blt_int8u * tseg1,
    blt_int8u * tseg2 ) [static]
```

Search algorithm to match the desired baudrate to a possible bus timing configuration.

Parameters

<i>baud</i>	The desired baudrate in kbps. Valid values are 10..1000.
<i>prescaler</i>	Pointer to where the value for the prescaler will be stored.
<i>tseg1</i>	Pointer to where the value for TSEG2 will be stored.
<i>tseg2</i>	Pointer to where the value for TSEG2 will be stored.

Returns

BLT_TRUE if the CAN bustiming register values were found, BLT_FALSE otherwise.

Referenced by CanInit().

7.316.2.2 CanInit()

```
void CanInit (
    void )
```

Initializes the CAN controller and synchronizes it to the CAN bus.

Returns

none.

7.316.2.3 CanReceivePacket()

```
blt_bool CanReceivePacket (
    blt_int8u * data,
    blt_int8u * len )
```

Receives a communication interface packet if one is present.

Parameters

<i>data</i>	Pointer to byte array where the data is to be stored.
<i>len</i>	Pointer where the length of the packet is to be stored.

Returns

BLT_TRUE is a packet was received, BLT_FALSE otherwise.

7.316.2.4 CanTransmitPacket()

```
void CanTransmitPacket (
    blt_int8u * data,
    blt_int8u len )
```

Transmits a packet formatted for the communication interface.

Parameters

<i>data</i>	Pointer to byte array with data that it to be transmitted.
<i>len</i>	Number of bytes that are to be transmitted.

Returns

none.

7.316.3 Variable Documentation**7.316.3.1 canTiming**

```
const tCanBusTiming canTiming[] [static]
```

Initial value:

```
=
{
    { 5, 2 },
    { 6, 2 },
    { 6, 3 },
    { 7, 3 },
    { 8, 3 },
    { 9, 3 },
    { 9, 4 },
    { 10, 4 },
    { 11, 4 },
    { 12, 4 },
    { 12, 5 },
```

```

{ 13, 5 },
{ 14, 5 },
{ 15, 5 },
{ 15, 6 },
{ 16, 6 },
{ 16, 7 },
{ 16, 8 }
}

```

CAN bittiming table for dynamically calculating the bittiming settings.

According to the CAN protocol 1 bit-time can be made up of between 8..25 time quanta (TQ). The total TQ in a bit is SYNC + TSEG1 + TSEG2 with SYNC always being 1. The sample point is (SYNC + TSEG1) / (SYNC + TSEG1 + SEG2) * 100%. This array contains possible and valid time quanta configurations with a sample point between 68..78%.

Referenced by CanGetSpeedConfig().

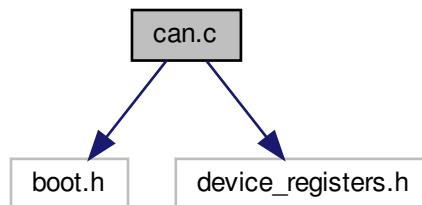
7.317 can.c File Reference

Bootloader CAN communication interface source file.

```

#include "boot.h"
#include "device_registers.h"
Include dependency graph for ARMCM4_S32K14/can.c:

```



Data Structures

- struct [tCanBusTiming](#)
Structure type for grouping CAN bus timing related information.

Macros

- #define [CAN_INIT_TIMEOUT_MS](#) (250U)
Timeout for entering/leaving CAN initialization mode in milliseconds.
- #define [CAN_MSG_TX_TIMEOUT_MS](#) (50U)
Timeout for transmitting a CAN message in milliseconds.
- #define [CANx](#) (CAN0)
Set the peripheral CAN0 base pointer.

- `#define PCC_FlexCANx_INDEX` (`PCC_FlexCAN0_INDEX`)
Set the PCC index offset for CAN0.
- `#define CANx_MAX_MB_NUM` (`FEATURE_CAN0_MAX_MB_NUM`)
Set the number of message boxes supported by CAN0.
- `#define CAN_TX_MSGBOX_NUM` (8U)
The mailbox used for transmitting the XCP respond message.
- `#define CAN_RX_MSGBOX_NUM` (9U)
The mailbox used for receiving the XCP command message.

Functions

- static `blt_bool CanGetSpeedConfig` (`blt_int16u` baud, `blt_int16u` *prescaler, `tCanBusTiming` *busTimingCfg)
Search algorithm to match the desired baudrate to a possible bus timing configuration.
- static void `CanFreezeModeEnter` (void)
Places the CAN controller in freeze mode. Note that the CAN controller can only be placed in freeze mode, if it is actually enabled.
- static void `CanFreezeModeExit` (void)
Leaves the CAN controller's freeze mode. Note that this operation can only be done, if it is actually enabled.
- static void `CanDisabledModeEnter` (void)
Places the CAN controller in disabled mode.
- static void `CanDisabledModeExit` (void)
Places the CAN controller in enabled mode.
- void `CanInit` (void)
Initializes the CAN controller and synchronizes it to the CAN bus.
- void `CanTransmitPacket` (`blt_int8u` *data, `blt_int8u` len)
Transmits a packet formatted for the communication interface.
- `blt_bool CanReceivePacket` (`blt_int8u` *data, `blt_int8u` *len)
Receives a communication interface packet if one is present.

Variables

- static const `tCanBusTiming` canTiming []
CAN bit timing table for dynamically calculating the bittiming settings.
- static volatile `blt_int32u` dummyTimerVal
Dummy variable to store the CAN controller's free running timer value in. This is needed at the end of a CAN message reception to unlock the mailbox again. If this variable is declared locally within the function, it generates an unwanted compiler warning about assigning a value and not using it. For this reason this dummy variable is declare here as a module global.

7.317.1 Detailed Description

Bootloader CAN communication interface source file.

7.317.2 Function Documentation

7.317.2.1 CanDisabledModeEnter()

```
static void CanDisabledModeEnter (  
    void ) [static]
```

Places the CAN controller in disabled mode.

Returns

none.

Referenced by CanInit().

7.317.2.2 CanDisabledModeExit()

```
static void CanDisabledModeExit (  
    void ) [static]
```

Places the CAN controller in enabled mode.

Returns

none.

Referenced by CanInit().

7.317.2.3 CanFreezeModeEnter()

```
static void CanFreezeModeEnter (  
    void ) [static]
```

Places the CAN controller in freeze mode. Note that the CAN controller can only be placed in freeze mode, if it is actually enabled.

Returns

none.

Referenced by CanInit().

7.317.2.4 CanFreezeModeExit()

```
static void CanFreezeModeExit (  
    void ) [static]
```

Leaves the CAN controller's freeze mode. Note that this operation can only be done, if it is actually enabled.

Returns

none.

Referenced by CanInit().

7.317.2.5 CanGetSpeedConfig()

```
static blt_bool CanGetSpeedConfig (  
    blt_int16u baud,  
    blt_int16u * prescaler,  
    tCanBusTiming * busTimingCfg ) [static]
```

Search algorithm to match the desired baudrate to a possible bus timing configuration.

Parameters

<i>baud</i>	The desired baudrate in kbps. Valid values are 10..1000.
<i>prescaler</i>	Pointer to where the value for the prescaler will be stored.
<i>busTimingCfg</i>	Pointer to where the bus timing values will be stored.

Returns

BLT_TRUE if the CAN bustiming register values were found, BLT_FALSE otherwise.

Referenced by CanInit().

7.317.2.6 CanInit()

```
void CanInit (  
    void )
```

Initializes the CAN controller and synchronizes it to the CAN bus.

Returns

none.

7.317.2.7 CanReceivePacket()

```
blt_bool CanReceivePacket (
    blt_int8u * data,
    blt_int8u * len )
```

Receives a communication interface packet if one is present.

Parameters

<i>data</i>	Pointer to byte array where the data is to be stored.
<i>len</i>	Pointer where the length of the packet is to be stored.

Returns

BLT_TRUE is a packet was received, BLT_FALSE otherwise.

7.317.2.8 CanTransmitPacket()

```
void CanTransmitPacket (
    blt_int8u * data,
    blt_int8u len )
```

Transmits a packet formatted for the communication interface.

Parameters

<i>data</i>	Pointer to byte array with data that it to be transmitted.
<i>len</i>	Number of bytes that are to be transmitted.

Returns

none.

7.317.3 Variable Documentation

7.317.3.1 canTiming

```
const tCanBusTiming canTiming[] [static]
```

Initial value:

```

=
{

    { 8U, 3U, 2U, 2U },
    { 9U, 3U, 3U, 2U },
    { 10U, 3U, 3U, 3U },
    { 11U, 4U, 3U, 3U },
    { 12U, 4U, 4U, 3U },
    { 13U, 5U, 4U, 3U },
    { 14U, 5U, 4U, 4U },
    { 15U, 6U, 4U, 4U },
    { 16U, 6U, 5U, 4U },
    { 17U, 7U, 5U, 4U },
    { 18U, 7U, 5U, 5U },
    { 19U, 8U, 5U, 5U },
    { 20U, 8U, 6U, 5U },
    { 21U, 8U, 7U, 5U },
    { 22U, 8U, 7U, 6U },
    { 23U, 8U, 8U, 6U },
    { 24U, 8U, 8U, 7U },
    { 25U, 8U, 8U, 8U }
}

```

CAN bit timing table for dynamically calculating the bittiming settings.

According to the CAN protocol 1 bit-time can be made up of between 8..25 time quanta (TQ). The total TQ in a bit is SYNC + TSEG1 + TSEG2 with SYNC always being 1. The sample point is (SYNC + TSEG1) / (SYNC + TSEG1 + TSEG2)

- 100%. This array contains possible and valid time quanta configurations with a sample point between 68..78%. A visual representation of the TQ in a bit is: | SYNCSEG | TIME1SEG | TIME2SEG | Or with an alternative representation: | SYNCSEG | PROPSEG | PHASE1SEG | PHASE2SEG | With the alternative representation TIME1SEG = PROPSEG + PHASE1SEG.

Referenced by CanGetSpeedConfig().

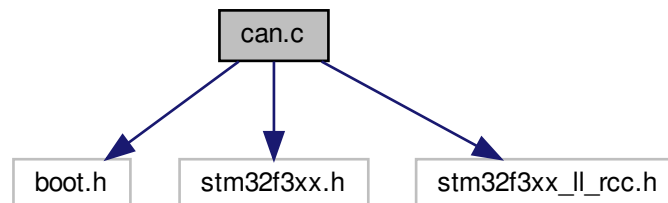
7.318 can.c File Reference

Bootloader CAN communication interface source file.

```

#include "boot.h"
#include "stm32f3xx.h"
#include "stm32f3xx_ll_rcc.h"
Include dependency graph for ARMCM4_STM32F3/can.c:

```



Data Structures

- struct [tCanBusTiming](#)
Structure type for grouping CAN bus timing related information.

Macros

- #define [CAN_MSG_TX_TIMEOUT_MS](#) (50u)
Timeout for transmitting a CAN message in milliseconds.
- #define [CAN_CHANNEL](#) CAN
Set CAN base address to CAN1.

Functions

- static [blt_bool](#) [CanGetSpeedConfig](#) ([blt_int16u](#) baud, [blt_int16u](#) *prescaler, [blt_int8u](#) *tseg1, [blt_int8u](#) *tseg2)
Search algorithm to match the desired baudrate to a possible bus timing configuration.
- void [CanInit](#) (void)
Initializes the CAN controller and synchronizes it to the CAN bus.
- void [CanTransmitPacket](#) ([blt_int8u](#) *data, [blt_int8u](#) len)
Transmits a packet formatted for the communication interface.
- [blt_bool](#) [CanReceivePacket](#) ([blt_int8u](#) *data, [blt_int8u](#) *len)
Receives a communication interface packet if one is present.

Variables

- static const [tCanBusTiming](#) [canTiming](#) []
CAN bittiming table for dynamically calculating the bittiming settings.
- static CAN_HandleTypeDef [canHandle](#)
CAN handle to be used in API calls.

7.318.1 Detailed Description

Bootloader CAN communication interface source file.

7.318.2 Function Documentation

7.318.2.1 CanGetSpeedConfig()

```
static blt\_bool CanGetSpeedConfig (
    blt\_int16u baud,
    blt\_int16u * prescaler,
    blt\_int8u * tseg1,
    blt\_int8u * tseg2 ) [static]
```

Search algorithm to match the desired baudrate to a possible bus timing configuration.

Parameters

<i>baud</i>	The desired baudrate in kbps. Valid values are 10..1000.
<i>prescaler</i>	Pointer to where the value for the prescaler will be stored.
<i>tseg1</i>	Pointer to where the value for TSEG2 will be stored.
<i>tseg2</i>	Pointer to where the value for TSEG2 will be stored.

Returns

BLT_TRUE if the CAN bustiming register values were found, BLT_FALSE otherwise.

Referenced by CanInit().

7.318.2.2 CanInit()

```
void CanInit (
    void )
```

Initializes the CAN controller and synchronizes it to the CAN bus.

Returns

none.

7.318.2.3 CanReceivePacket()

```
blt_bool CanReceivePacket (
    blt_int8u * data,
    blt_int8u * len )
```

Receives a communication interface packet if one is present.

Parameters

<i>data</i>	Pointer to byte array where the data is to be stored.
<i>len</i>	Pointer where the length of the packet is to be stored.

Returns

BLT_TRUE is a packet was received, BLT_FALSE otherwise.

7.318.2.4 CanTransmitPacket()

```
void CanTransmitPacket (
    blt_int8u * data,
    blt_int8u len )
```

Transmits a packet formatted for the communication interface.

Parameters

<i>data</i>	Pointer to byte array with data that it to be transmitted.
<i>len</i>	Number of bytes that are to be transmitted.

Returns

none.

7.318.3 Variable Documentation

7.318.3.1 canTiming

```
const tCanBusTiming canTiming[] [static]
```

Initial value:

```
=
{
    { 5, 2 },
    { 6, 2 },
    { 6, 3 },
    { 7, 3 },
    { 8, 3 },
    { 9, 3 },
    { 9, 4 },
    { 10, 4 },
    { 11, 4 },
    { 12, 4 },
    { 12, 5 },
    { 13, 5 },
    { 14, 5 },
    { 15, 5 },
    { 15, 6 },
    { 16, 6 },
    { 16, 7 },
    { 16, 8 }
}
```

CAN bittiming table for dynamically calculating the bittiming settings.

According to the CAN protocol 1 bit-time can be made up of between 8..25 time quanta (TQ). The total TQ in a bit is SYNC + TSEG1 + TSEG2 with SYNC always being 1. The sample point is (SYNC + TSEG1) / (SYNC + TSEG1 + SEG2) * 100%. This array contains possible and valid time quanta configurations with a sample point between 68..78%.

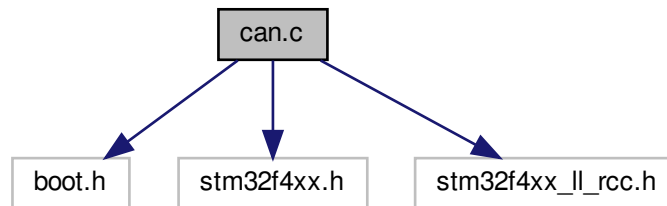
Referenced by CanGetSpeedConfig().

7.319 can.c File Reference

Bootloader CAN communication interface source file.

```
#include "boot.h"
#include "stm32f4xx.h"
#include "stm32f4xx_ll_rcc.h"
```

Include dependency graph for ARMCM4_STM32F4/can.c:



Data Structures

- struct [tCanBusTiming](#)
Structure type for grouping CAN bus timing related information.

Macros

- `#define CAN_MSG_TX_TIMEOUT_MS (50u)`
Timeout for transmitting a CAN message in milliseconds.
- `#define CAN_CHANNEL CAN1`
Set CAN base address to CAN1.

Functions

- static [blt_bool](#) [CanGetSpeedConfig](#) ([blt_int16u](#) baud, [blt_int16u](#) *prescaler, [blt_int8u](#) *tseg1, [blt_int8u](#) *tseg2)
Search algorithm to match the desired baudrate to a possible bus timing configuration.
- void [CanInit](#) (void)
Initializes the CAN controller and synchronizes it to the CAN bus.
- void [CanTransmitPacket](#) ([blt_int8u](#) *data, [blt_int8u](#) len)
Transmits a packet formatted for the communication interface.
- [blt_bool](#) [CanReceivePacket](#) ([blt_int8u](#) *data, [blt_int8u](#) *len)
Receives a communication interface packet if one is present.

Variables

- static const [tCanBusTiming](#) [canTiming](#) []
CAN bittiming table for dynamically calculating the bittiming settings.
- static CAN_HandleTypeDef [canHandle](#)
CAN handle to be used in API calls.

7.319.1 Detailed Description

Bootloader CAN communication interface source file.

7.319.2 Function Documentation

7.319.2.1 CanGetSpeedConfig()

```
static blt_bool CanGetSpeedConfig (  
    blt_int16u baud,  
    blt_int16u * prescaler,  
    blt_int8u * tseg1,  
    blt_int8u * tseg2 ) [static]
```

Search algorithm to match the desired baudrate to a possible bus timing configuration.

Parameters

<i>baud</i>	The desired baudrate in kbps. Valid values are 10..1000.
<i>prescaler</i>	Pointer to where the value for the prescaler will be stored.
<i>tseg1</i>	Pointer to where the value for TSEG2 will be stored.
<i>tseg2</i>	Pointer to where the value for TSEG2 will be stored.

Returns

BLT_TRUE if the CAN bustiming register values were found, BLT_FALSE otherwise.

Referenced by CanInit().

7.319.2.2 CanInit()

```
void CanInit (  
    void )
```

Initializes the CAN controller and synchronizes it to the CAN bus.

Returns

none.

7.319.2.3 CanReceivePacket()

```
blt_bool CanReceivePacket (  
    blt_int8u * data,  
    blt_int8u * len )
```

Receives a communication interface packet if one is present.

Parameters

<i>data</i>	Pointer to byte array where the data is to be stored.
<i>len</i>	Pointer where the length of the packet is to be stored.

Returns

BLT_TRUE is a packet was received, BLT_FALSE otherwise.

7.319.2.4 CanTransmitPacket()

```
void CanTransmitPacket (
    blt_int8u * data,
    blt_int8u len )
```

Transmits a packet formatted for the communication interface.

Parameters

<i>data</i>	Pointer to byte array with data that it to be transmitted.
<i>len</i>	Number of bytes that are to be transmitted.

Returns

none.

7.319.3 Variable Documentation**7.319.3.1 canTiming**

```
const tCanBusTiming canTiming[] [static]
```

Initial value:

```
=
{
    { 5, 2 },
    { 6, 2 },
    { 6, 3 },
    { 7, 3 },
    { 8, 3 },
    { 9, 3 },
    { 9, 4 },
    { 10, 4 },
    { 11, 4 },
    { 12, 4 },
    { 12, 5 },
```

```

{ 13, 5 },
{ 14, 5 },
{ 15, 5 },
{ 15, 6 },
{ 16, 6 },
{ 16, 7 },
{ 16, 8 }
}

```

CAN bittiming table for dynamically calculating the bittiming settings.

According to the CAN protocol 1 bit-time can be made up of between 8..25 time quanta (TQ). The total TQ in a bit is SYNC + TSEG1 + TSEG2 with SYNC always being 1. The sample point is (SYNC + TSEG1) / (SYNC + TSEG1 + SEG2) * 100%. This array contains possible and valid time quanta configurations with a sample point between 68..78%.

Referenced by CanGetSpeedConfig().

7.320 can.c File Reference

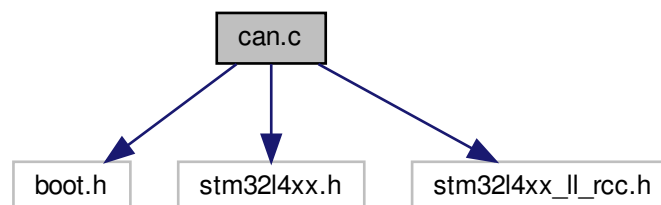
Bootloader CAN communication interface source file.

```

#include "boot.h"
#include "stm32l4xx.h"
#include "stm32l4xx_ll_rcc.h"

```

Include dependency graph for ARMCM4_STM32L4/can.c:



Data Structures

- struct [tCanBusTiming](#)
Structure type for grouping CAN bus timing related information.

Macros

- #define [CAN_MSG_TX_TIMEOUT_MS](#) (50u)
Timeout for transmitting a CAN message in milliseconds.

Functions

- static `blt_bool CanGetSpeedConfig (blt_int16u baud, blt_int16u *prescaler, blt_int8u *tseg1, blt_int8u *tseg2)`
Search algorithm to match the desired baudrate to a possible bus timing configuration.
- void `CanInit (void)`
Initializes the CAN controller and synchronizes it to the CAN bus.
- void `CanTransmitPacket (blt_int8u *data, blt_int8u len)`
Transmits a packet formatted for the communication interface.
- `blt_bool CanReceivePacket (blt_int8u *data, blt_int8u *len)`
Receives a communication interface packet if one is present.

Variables

- static const `tCanBusTiming canTiming []`
CAN bittiming table for dynamically calculating the bittiming settings.
- static `CAN_HandleTypeDef canHandle`
CAN handle to be used in API calls.

7.320.1 Detailed Description

Bootloader CAN communication interface source file.

7.320.2 Function Documentation

7.320.2.1 CanGetSpeedConfig()

```
static blt_bool CanGetSpeedConfig (
    blt_int16u baud,
    blt_int16u * prescaler,
    blt_int8u * tseg1,
    blt_int8u * tseg2 ) [static]
```

Search algorithm to match the desired baudrate to a possible bus timing configuration.

Parameters

<i>baud</i>	The desired baudrate in kbps. Valid values are 10..1000.
<i>prescaler</i>	Pointer to where the value for the prescaler will be stored.
<i>tseg1</i>	Pointer to where the value for TSEG2 will be stored.
<i>tseg2</i>	Pointer to where the value for TSEG2 will be stored.

Returns

BLT_TRUE if the CAN bustiming register values were found, BLT_FALSE otherwise.

Referenced by CanInit().

7.320.2.2 CanInit()

```
void CanInit (
    void )
```

Initializes the CAN controller and synchronizes it to the CAN bus.

Returns

none.

7.320.2.3 CanReceivePacket()

```
blt_bool CanReceivePacket (
    blt_int8u * data,
    blt_int8u * len )
```

Receives a communication interface packet if one is present.

Parameters

<i>data</i>	Pointer to byte array where the data is to be stored.
<i>len</i>	Pointer where the length of the packet is to be stored.

Returns

BLT_TRUE is a packet was received, BLT_FALSE otherwise.

7.320.2.4 CanTransmitPacket()

```
void CanTransmitPacket (
    blt_int8u * data,
    blt_int8u len )
```

Transmits a packet formatted for the communication interface.

Parameters

<i>data</i>	Pointer to byte array with data that it to be transmitted.
<i>len</i>	Number of bytes that are to be transmitted.

Returns

none.

7.320.3 Variable Documentation**7.320.3.1 canTiming**

```
const tCanBusTiming canTiming[] [static]
```

Initial value:

```
=
{
    { 5, 2 },
    { 6, 2 },
    { 6, 3 },
    { 7, 3 },
    { 8, 3 },
    { 9, 3 },
    { 9, 4 },
    { 10, 4 },
    { 11, 4 },
    { 12, 4 },
    { 12, 5 },
    { 13, 5 },
    { 14, 5 },
    { 15, 5 },
    { 15, 6 },
    { 16, 6 },
    { 16, 7 },
    { 16, 8 }
}
```

CAN bittiming table for dynamically calculating the bittiming settings.

According to the CAN protocol 1 bit-time can be made up of between 8..25 time quanta (TQ). The total TQ in a bit is SYNC + TSEG1 + TSEG2 with SYNC always being 1. The sample point is (SYNC + TSEG1) / (SYNC + TSEG1 + SEG2) * 100%. This array contains possible and valid time quanta configurations with a sample point between 68..78%.

Referenced by CanGetSpeedConfig().

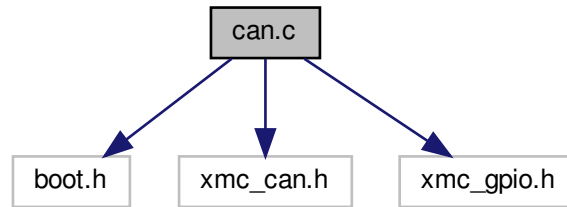
7.321 can.c File Reference

Bootloader CAN communication interface source file.

```
#include "boot.h"
#include "xmc_can.h"
```

```
#include "xmc_gpio.h"
```

Include dependency graph for ARMCM4_XMC4/can.c:



Macros

- `#define CAN_MSG_TX_TIMEOUT_MS (50u)`
Timeout for transmitting a CAN message in milliseconds.
- `#define CAN_CHANNEL ((CAN_NODE_TypeDef *)(canChannelMap[BOOT_COM_CAN_CHANNEL_INDEX]))`
Macro for accessing the CAN channel handle in the format that is expected by the XMCLib CAN driver.
- `#define CAN_TX_MSBOBJ (CAN_MO0)`
Message object dedicated to message transmission.
- `#define CAN_TX_MSBOBJ_IDX (0)`
Index of the message object dedicated to message transmission.
- `#define CAN_RX_MSBOBJ (CAN_MO1)`
Message object dedicated to message reception.
- `#define CAN_RX_MSBOBJ_IDX (1)`
Index of the message object dedicated to message reception.

Functions

- `void CanInit (void)`
Initializes the CAN controller and synchronizes it to the CAN bus.
- `void CanTransmitPacket (blt_int8u *data, blt_int8u len)`
Transmits a packet formatted for the communication interface.
- `blt_bool CanReceivePacket (blt_int8u *data, blt_int8u *len)`
Receives a communication interface packet if one is present.

Variables

- `static const CAN_NODE_TypeDef * canChannelMap []`
Helper array to quickly convert the channel index, as specific in the boot- loader's configuration header, to the associated channel handle that the XMCLib's CAN driver requires.
- `static XMC_CAN_MO_t transmitMsgObj`
Transmit message object data structure.
- `static XMC_CAN_MO_t receiveMsgObj`
Receive message object data structure.

7.321.1 Detailed Description

Bootloader CAN communication interface source file.

7.321.2 Function Documentation

7.321.2.1 CanInit()

```
void CanInit (
    void )
```

Initializes the CAN controller and synchronizes it to the CAN bus.

Returns

none.

7.321.2.2 CanReceivePacket()

```
blt_bool CanReceivePacket (
    blt_int8u * data,
    blt_int8u * len )
```

Receives a communication interface packet if one is present.

Parameters

<i>data</i>	Pointer to byte array where the data is to be stored.
<i>len</i>	Pointer where the length of the packet is to be stored.

Returns

BLT_TRUE is a packet was received, BLT_FALSE otherwise.

7.321.2.3 CanTransmitPacket()

```
void CanTransmitPacket (
    blt_int8u * data,
    blt_int8u len )
```

Transmits a packet formatted for the communication interface.

Parameters

<i>data</i>	Pointer to byte array with data that it to be transmitted.
<i>len</i>	Number of bytes that are to be transmitted.

Returns

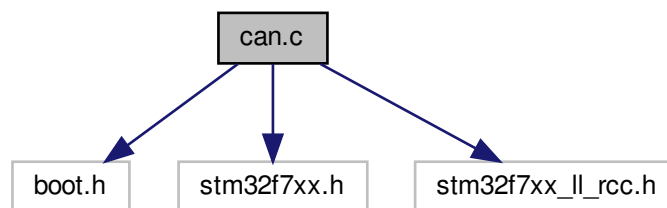
none.

7.322 can.c File Reference

Bootloader CAN communication interface source file.

```
#include "boot.h"
#include "stm32f7xx.h"
#include "stm32f7xx_ll_rcc.h"
```

Include dependency graph for ARMCM7_STM32F7/can.c:

**Data Structures**

- struct [tCanBusTiming](#)
Structure type for grouping CAN bus timing related information.

Macros

- #define [CAN_MSG_TX_TIMEOUT_MS](#) (50u)
Timeout for transmitting a CAN message in milliseconds.
- #define [CAN_CHANNEL](#) CAN1
Set CAN base address to CAN1.

Functions

- static `blt_bool CanGetSpeedConfig (blt_int16u baud, blt_int16u *prescaler, blt_int8u *tseg1, blt_int8u *tseg2)`
Search algorithm to match the desired baudrate to a possible bus timing configuration.
- void `CanInit (void)`
Initializes the CAN controller and synchronizes it to the CAN bus.
- void `CanTransmitPacket (blt_int8u *data, blt_int8u len)`
Transmits a packet formatted for the communication interface.
- `blt_bool CanReceivePacket (blt_int8u *data, blt_int8u *len)`
Receives a communication interface packet if one is present.

Variables

- static const `tCanBusTiming canTiming []`
CAN bittiming table for dynamically calculating the bittiming settings.
- static `CAN_HandleTypeDef canHandle`
CAN handle to be used in API calls.

7.322.1 Detailed Description

Bootloader CAN communication interface source file.

7.322.2 Function Documentation

7.322.2.1 CanGetSpeedConfig()

```
static blt_bool CanGetSpeedConfig (
    blt_int16u baud,
    blt_int16u * prescaler,
    blt_int8u * tseg1,
    blt_int8u * tseg2 ) [static]
```

Search algorithm to match the desired baudrate to a possible bus timing configuration.

Parameters

<i>baud</i>	The desired baudrate in kbps. Valid values are 10..1000.
<i>prescaler</i>	Pointer to where the value for the prescaler will be stored.
<i>tseg1</i>	Pointer to where the value for TSEG2 will be stored.
<i>tseg2</i>	Pointer to where the value for TSEG2 will be stored.

Returns

BLT_TRUE if the CAN bustiming register values were found, BLT_FALSE otherwise.

Referenced by CanInit().

7.322.2.2 CanInit()

```
void CanInit (
    void )
```

Initializes the CAN controller and synchronizes it to the CAN bus.

Returns

none.

7.322.2.3 CanReceivePacket()

```
blt_bool CanReceivePacket (
    blt_int8u * data,
    blt_int8u * len )
```

Receives a communication interface packet if one is present.

Parameters

<i>data</i>	Pointer to byte array where the data is to be stored.
<i>len</i>	Pointer where the length of the packet is to be stored.

Returns

BLT_TRUE is a packet was received, BLT_FALSE otherwise.

7.322.2.4 CanTransmitPacket()

```
void CanTransmitPacket (
    blt_int8u * data,
    blt_int8u len )
```

Transmits a packet formatted for the communication interface.

Parameters

<i>data</i>	Pointer to byte array with data that it to be transmitted.
<i>len</i>	Number of bytes that are to be transmitted.

Returns

none.

7.322.3 Variable Documentation**7.322.3.1 canTiming**

```
const tCanBusTiming canTiming[] [static]
```

Initial value:

```
=
{
    { 5, 2 },
    { 6, 2 },
    { 6, 3 },
    { 7, 3 },
    { 8, 3 },
    { 9, 3 },
    { 9, 4 },
    { 10, 4 },
    { 11, 4 },
    { 12, 4 },
    { 12, 5 },
    { 13, 5 },
    { 14, 5 },
    { 15, 5 },
    { 15, 6 },
    { 16, 6 },
    { 16, 7 },
    { 16, 8 }
}
```

CAN bittiming table for dynamically calculating the bittiming settings.

According to the CAN protocol 1 bit-time can be made up of between 8..25 time quanta (TQ). The total TQ in a bit is SYNC + TSEG1 + TSEG2 with SYNC always being 1. The sample point is (SYNC + TSEG1) / (SYNC + TSEG1 + SEG2) * 100%. This array contains possible and valid time quanta configurations with a sample point between 68..78%.

Referenced by CanGetSpeedConfig().

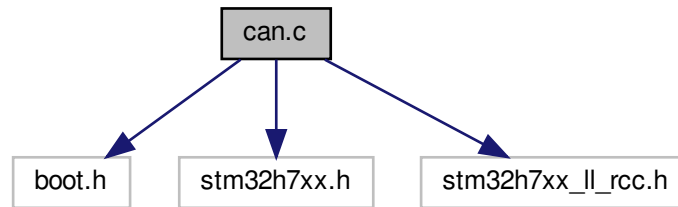
7.323 can.c File Reference

Bootloader CAN communication interface source file.

```
#include "boot.h"
#include "stm32h7xx.h"
```

```
#include "stm32h7xx_ll_rcc.h"
```

Include dependency graph for ARMCM7_STM32H7/can.c:



Data Structures

- struct [tCanBusTiming](#)
Structure type for grouping CAN bus timing related information.

Macros

- #define [CAN_MSG_TX_TIMEOUT_MS](#) (50u)
Timeout for transmitting a CAN message in milliseconds.
- #define [CAN_CHANNEL](#) FDCAN1
Set CAN base address to CAN1.

Functions

- static [blt_bool CanGetSpeedConfig](#) ([blt_int16u](#) baud, [blt_int16u](#) *prescaler, [blt_int8u](#) *tseg1, [blt_int8u](#) *tseg2)
Search algorithm to match the desired baudrate to a possible bus timing configuration.
- void [CanInit](#) (void)
Initializes the CAN controller and synchronizes it to the CAN bus.
- void [CanTransmitPacket](#) ([blt_int8u](#) *data, [blt_int8u](#) len)
Transmits a packet formatted for the communication interface.
- [blt_bool CanReceivePacket](#) ([blt_int8u](#) *data, [blt_int8u](#) *len)
Receives a communication interface packet if one is present.

Variables

- static const [tCanBusTiming](#) canTiming []
CAN bittiming table for dynamically calculating the bittiming settings.
- static FDCAN_HandleTypeDef [canHandle](#)
CAN handle to be used in API calls.

7.323.1 Detailed Description

Bootloader CAN communication interface source file.

7.323.2 Function Documentation

7.323.2.1 CanGetSpeedConfig()

```
static blt_bool CanGetSpeedConfig (  
    blt_int16u baud,  
    blt_int16u * prescaler,  
    blt_int8u * tseg1,  
    blt_int8u * tseg2 ) [static]
```

Search algorithm to match the desired baudrate to a possible bus timing configuration.

Parameters

<i>baud</i>	The desired baudrate in kbps. Valid values are 10..1000.
<i>prescaler</i>	Pointer to where the value for the prescaler will be stored.
<i>tseg1</i>	Pointer to where the value for TSEG2 will be stored.
<i>tseg2</i>	Pointer to where the value for TSEG2 will be stored.

Returns

BLT_TRUE if the CAN bustiming register values were found, BLT_FALSE otherwise.

Referenced by CanInit().

7.323.2.2 CanInit()

```
void CanInit (  
    void )
```

Initializes the CAN controller and synchronizes it to the CAN bus.

Returns

none.

7.323.2.3 CanReceivePacket()

```
blt_bool CanReceivePacket (  
    blt_int8u * data,  
    blt_int8u * len )
```

Receives a communication interface packet if one is present.

Parameters

<i>data</i>	Pointer to byte array where the data is to be stored.
<i>len</i>	Pointer where the length of the packet is to be stored.

Returns

BLT_TRUE is a packet was received, BLT_FALSE otherwise.

7.323.2.4 CanTransmitPacket()

```
void CanTransmitPacket (
    blt_int8u * data,
    blt_int8u len )
```

Transmits a packet formatted for the communication interface.

Parameters

<i>data</i>	Pointer to byte array with data that it to be transmitted.
<i>len</i>	Number of bytes that are to be transmitted.

Returns

none.

7.323.3 Variable Documentation**7.323.3.1 canTiming**

```
const tCanBusTiming canTiming[] [static]
```

Initial value:

```
=
{
    { 5, 2 },
    { 6, 2 },
    { 6, 3 },
    { 7, 3 },
    { 8, 3 },
    { 9, 3 },
    { 9, 4 },
    { 10, 4 },
    { 11, 4 },
    { 12, 4 },
    { 12, 5 },
```

```

{ 13, 5 },
{ 14, 5 },
{ 15, 5 },
{ 15, 6 },
{ 16, 6 },
{ 16, 7 },
{ 16, 8 }
}

```

CAN bittiming table for dynamically calculating the bittiming settings.

According to the CAN protocol 1 bit-time can be made up of between 8..25 time quanta (TQ). The total TQ in a bit is SYNC + TSEG1 + TSEG2 with SYNC always being 1. The sample point is (SYNC + TSEG1) / (SYNC + TSEG1 + SEG2) * 100%. This array contains possible and valid time quanta configurations with a sample point between 68..78%.

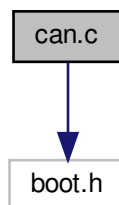
Referenced by CanGetSpeedConfig().

7.324 can.c File Reference

Bootloader CAN communication interface source file.

```
#include "boot.h"
```

Include dependency graph for HCS12/can.c:



Data Structures

- struct [tCanRxMsgSlot](#)
Structure type with the layout of a CAN reception message slot.
- struct [tCanTxMsgSlot](#)
Structure type with the layout of a CAN transmit message slot.
- struct [tCanRegs](#)
Structure type with the layout of the CAN related control registers.
- struct [tCanBusTiming](#)
Structure type for grouping CAN bus timing related information.

Macros

- `#define CAN_INIT_TIMEOUT_MS (250u)`
Timeout for entering/leaving CAN initialization mode in milliseconds.
- `#define CAN_MSG_TX_TIMEOUT_MS (50u)`
Timeout for transmitting a CAN message in milliseconds.
- `#define CAN_REGS_BASE_ADDRESS (0x0140)`
Set CAN base address to CAN0.
- `#define CAN ((volatile tCanRegs *)CAN_REGS_BASE_ADDRESS)`
Macro for accessing the CAN related control registers.
- `#define EXTIDMASK_BIT (0x80000000)`
Configures a CAN message id for 29-bit (extended).
- `#define INITRQ_BIT (0x01)`
Initialization mode request bit.
- `#define INITAK_BIT (0x01)`
Initialization mode handshake bit.
- `#define CANE_BIT (0x80)`
CAN controller enable bit.
- `#define IDAM0_BIT (0x10)`
Filter mode bit 0.
- `#define IDAM1_BIT (0x20)`
Filter mode bit 1.
- `#define TX0_BIT (0x01)`
Transmit buffer 0 select bit.
- `#define TXE0_BIT (0x01)`
Transmit buffer 0 empty bit.
- `#define IDE_BIT (0x08)`
29-bit extended id bit.
- `#define RXF_BIT (0x01)`
Receive buffer full flag bit.

Functions

- `static blt_bool CanGetSpeedConfig (blt_int16u baud, blt_int8u *btr0, blt_int8u *btr1)`
Search algorithm to match the desired baudrate to a possible bus timing configuration.
- `void CanInit (void)`
Initializes the CAN controller and synchronizes it to the CAN bus.
- `void CanTransmitPacket (blt_int8u *data, blt_int8u len)`
Transmits a packet formatted for the communication interface.
- `blt_bool CanReceivePacket (blt_int8u *data, blt_int8u *len)`
Receives a communication interface packet if one is present.

Variables

- `static const tCanBusTiming canTiming []`
Array with possible time quanta configurations.

7.324.1 Detailed Description

Bootloader CAN communication interface source file.

7.324.2 Function Documentation

7.324.2.1 CanGetSpeedConfig()

```
static blt_bool CanGetSpeedConfig (
    blt_int16u baud,
    blt_int8u * btr0,
    blt_int8u * btr1 ) [static]
```

Search algorithm to match the desired baudrate to a possible bus timing configuration.

Parameters

<i>baud</i>	The desired baudrate in kbps. Valid values are 10..1000.
<i>btr0</i>	Pointer to where the value for register CANxBTR0 will be stored.
<i>btr1</i>	Pointer to where the value for register CANxBTR1 will be stored.

Returns

BLT_TRUE if the CAN bustiming register values were found, BLT_FALSE otherwise.

Referenced by CanInit().

7.324.2.2 CanInit()

```
void CanInit (
    void )
```

Initializes the CAN controller and synchronizes it to the CAN bus.

Returns

none.

7.324.2.3 CanReceivePacket()

```
blt_bool CanReceivePacket (
    blt_int8u * data,
    blt_int8u * len )
```

Receives a communication interface packet if one is present.

Parameters

<i>data</i>	Pointer to byte array where the data is to be stored.
<i>len</i>	Pointer where the length of the packet is to be stored.

Returns

BLT_TRUE is a packet was received, BLT_FALSE otherwise.

7.324.2.4 CanTransmitPacket()

```
void CanTransmitPacket (
    blt_int8u * data,
    blt_int8u len )
```

Transmits a packet formatted for the communication interface.

Parameters

<i>data</i>	Pointer to byte array with data that it to be transmitted.
<i>len</i>	Number of bytes that are to be transmitted.

Returns

none.

7.324.3 Variable Documentation**7.324.3.1 canTiming**

```
const tCanBusTiming canTiming[] [static]
```

Initial value:

```
=
{
    { 5, 2 },
    { 6, 2 },
    { 6, 3 },
    { 7, 3 },
    { 8, 3 },
    { 9, 3 },
    { 9, 4 },
    { 10, 4 },
    { 11, 4 },
    { 12, 4 },
    { 12, 5 },
```

```

{ 13, 5 },
{ 14, 5 },
{ 15, 5 },
{ 15, 6 },
{ 16, 6 },
{ 16, 7 },
{ 16, 8 }
}

```

Array with possible time quanta configurations.

According to the CAN protocol 1 bit-time can be made up of between 8..25 time quanta (TQ). The total TQ in a bit is SYNC + TSEG1 + TSEG2 with SYNC always being 1. The sample point is (SYNC + TSEG1) / (SYNC + TSEG1 + SEG2)

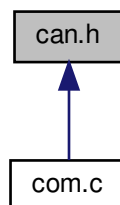
- 100%. This array contains possible and valid time quanta configurations with a sample point between 68..78%.

Referenced by CanGetSpeedConfig().

7.325 can.h File Reference

Bootloader CAN communication interface header file.

This graph shows which files directly or indirectly include this file:



Functions

- void [CanInit](#) (void)
Initializes the CAN controller and synchronizes it to the CAN bus.
- void [CanTransmitPacket](#) (blt_int8u *data, blt_int8u len)
Transmits a packet formatted for the communication interface.
- blt_bool [CanReceivePacket](#) (blt_int8u *data, blt_int8u *len)
Receives a communication interface packet if one is present.

7.325.1 Detailed Description

Bootloader CAN communication interface header file.

7.325.2 Function Documentation

7.325.2.1 CanInit()

```
void CanInit (
    void )
```

Initializes the CAN controller and synchronizes it to the CAN bus.

Returns

none.

Referenced by ComInit().

7.325.2.2 CanReceivePacket()

```
blt_bool CanReceivePacket (
    blt_int8u * data,
    blt_int8u * len )
```

Receives a communication interface packet if one is present.

Parameters

<i>data</i>	Pointer to byte array where the data is to be stored.
<i>len</i>	Pointer where the length of the packet is to be stored.

Returns

BLT_TRUE is a packet was received, BLT_FALSE otherwise.

Referenced by ComTask().

7.325.2.3 CanTransmitPacket()

```
void CanTransmitPacket (
    blt_int8u * data,
    blt_int8u len )
```

Transmits a packet formatted for the communication interface.

Parameters

<i>data</i>	Pointer to byte array with data that it to be transmitted.
<i>len</i>	Number of bytes that are to be transmitted.

Returns

none.

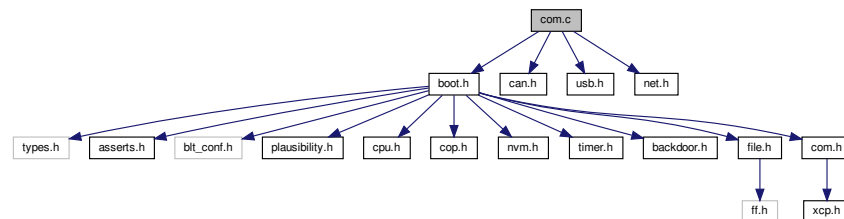
Referenced by ComTransmitPacket().

7.326 com.c File Reference

Bootloader communication interface source file.

```
#include "boot.h"
#include "can.h"
#include "usb.h"
#include "net.h"
```

Include dependency graph for com.c:



Functions

- void [ComInit](#) (void)
Initializes the communication module including the hardware needed for the communication.
- void [ComTask](#) (void)
Updates the communication module by checking if new data was received and submitting the request to process newly received data.
- void [ComFree](#) (void)
Releases the communication module.
- void [ComTransmitPacket](#) (blt_int8u *data, blt_int16u len)
Transmits the packet using the xcp transport layer.
- blt_int16u [ComGetActiveInterfaceMaxRxLen](#) (void)
Obtains the maximum number of bytes that can be received on the specified communication interface.
- blt_int16u [ComGetActiveInterfaceMaxTxLen](#) (void)
Obtains the maximum number of bytes that can be transmitted on the specified communication interface.
- blt_bool [ComIsConnected](#) (void)
This function obtains the XCP connection state.

Variables

- static `tComInterfaceId comActiveInterface = COM_IF_OTHER`
Holds the communication interface of the currently active interface.

7.326.1 Detailed Description

Bootloader communication interface source file.

7.326.2 Function Documentation

7.326.2.1 ComFree()

```
void ComFree (
    void )
```

Releases the communication module.

Returns

none

Referenced by `CpuStartUserProgram()`.

7.326.2.2 ComGetActiveInterfaceMaxRxLen()

```
blt_int16u ComGetActiveInterfaceMaxRxLen (
    void )
```

Obtains the maximum number of bytes that can be received on the specified communication interface.

Returns

Maximum number of bytes that can be received.

7.326.2.3 ComGetActiveInterfaceMaxTxLen()

```
blt_int16u ComGetActiveInterfaceMaxTxLen (
    void )
```

Obtains the maximum number of bytes that can be transmitted on the specified communication interface.

Returns

Maximum number of bytes that can be received.

7.326.2.4 ComInit()

```
void ComInit (
    void )
```

Initializes the communication module including the hardware needed for the communication.

Returns

none

Referenced by BootInit().

7.326.2.5 ComIsConnected()

```
blt_bool ComIsConnected (
    void )
```

This function obtains the XCP connection state.

Returns

BLT_TRUE when an XCP connection is established, BLT_FALSE otherwise.

Referenced by BackDoorCheck(), and FileHandleFirmwareUpdateRequest().

7.326.2.6 ComTask()

```
void ComTask (
    void )
```

Updates the communication module by checking if new data was received and submitting the request to process newly received data.

Returns

none

Referenced by BootTask().

7.326.2.7 ComTransmitPacket()

```
void ComTransmitPacket (
    blt_int8u * data,
    blt_int16u len )
```

Transmits the packet using the xcp transport layer.

Parameters

<i>data</i>	Pointer to the byte buffer with packet data.
<i>len</i>	Number of data bytes that need to be transmitted.

Returns

none

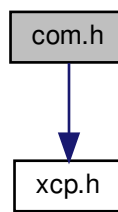
Referenced by XcpTransmitPacket().

7.327 com.h File Reference

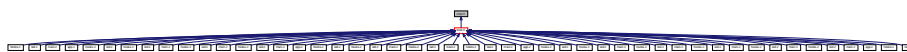
Bootloader communication interface header file.

```
#include "xcp.h"
```

Include dependency graph for com.h:



This graph shows which files directly or indirectly include this file:



Macros

- `#define BOOT_COM_RX_MAX_DATA (1)`
Defines the maximum number of bytes for transport layer reception depending on the activates interface(s).
- `#define BOOT_COM_RX_MAX_DATA (BOOT_COM_CAN_RX_MAX_DATA)`
Defines the maximum number of bytes for transport layer reception depending on the activates interface(s).
- `#define BOOT_COM_RX_MAX_DATA (BOOT_COM_NET_RX_MAX_DATA)`
Defines the maximum number of bytes for transport layer reception depending on the activates interface(s).
- `#define BOOT_COM_TX_MAX_DATA (1)`
Defines the maximum number of bytes for transport layer transmission depending on the activates interface(s).
- `#define BOOT_COM_TX_MAX_DATA (BOOT_COM_CAN_TX_MAX_DATA)`
Defines the maximum number of bytes for transport layer transmission depending on the activates interface(s).
- `#define BOOT_COM_TX_MAX_DATA (BOOT_COM_USB_TX_MAX_DATA)`
Defines the maximum number of bytes for transport layer transmission depending on the activates interface(s).
- `#define BOOT_COM_TX_MAX_DATA (BOOT_COM_NET_TX_MAX_DATA)`
Defines the maximum number of bytes for transport layer transmission depending on the activates interface(s).

Enumerations

- enum `tComInterfaceId` {
`COM_IF_RS232`, `COM_IF_CAN`, `COM_IF_USB`, `COM_IF_NET`,
`COM_IF_OTHER` }

Enumeration for the different communication interfaces.

Functions

- void `ComInit` (void)
Initializes the communication module including the hardware needed for the communication.
- void `ComTask` (void)
Updates the communication module by checking if new data was received and submitting the request to process newly received data.
- void `ComFree` (void)
Releases the communication module.
- `blt_int16u ComGetActiveInterfaceMaxRxLen` (void)
Obtains the maximum number of bytes that can be received on the specified communication interface.
- `blt_int16u ComGetActiveInterfaceMaxTxLen` (void)
Obtains the maximum number of bytes that can be transmitted on the specified communication interface.
- void `ComTransmitPacket` (`blt_int8u *data`, `blt_int16u len`)
Transmits the packet using the xcp transport layer.
- `blt_bool ComIsConnected` (void)
This function obtains the XCP connection state.

7.327.1 Detailed Description

Bootloader communication interface header file.

7.327.2 Enumeration Type Documentation

7.327.2.1 tComInterfaceId

enum `tComInterfaceId`

Enumeration for the different communication interfaces.

Enumerator

<code>COM_IF_RS232</code>	RS232 interface
<code>COM_IF_CAN</code>	CAN interface
<code>COM_IF_USB</code>	USB interface
<code>COM_IF_NET</code>	NET interface
<code>COM_IF_OTHER</code>	Other interface

7.327.3 Function Documentation

7.327.3.1 ComFree()

```
void ComFree (
    void )
```

Releases the communication module.

Returns

none

Referenced by CpuStartUserProgram().

7.327.3.2 ComGetActiveInterfaceMaxRxLen()

```
blt_int16u ComGetActiveInterfaceMaxRxLen (
    void )
```

Obtains the maximum number of bytes that can be received on the specified communication interface.

Returns

Maximum number of bytes that can be received.

7.327.3.3 ComGetActiveInterfaceMaxTxLen()

```
blt_int16u ComGetActiveInterfaceMaxTxLen (
    void )
```

Obtains the maximum number of bytes that can be transmitted on the specified communication interface.

Returns

Maximum number of bytes that can be received.

7.327.3.4 ComInit()

```
void ComInit (
    void )
```

Initializes the communication module including the hardware needed for the communication.

Returns

none

Referenced by BootInit().

7.327.3.5 ComIsConnected()

```
blt_bool ComIsConnected (
    void )
```

This function obtains the XCP connection state.

Returns

BLT_TRUE when an XCP connection is established, BLT_FALSE otherwise.

Referenced by BackDoorCheck(), and FileHandleFirmwareUpdateRequest().

7.327.3.6 ComTask()

```
void ComTask (
    void )
```

Updates the communication module by checking if new data was received and submitting the request to process newly received data.

Returns

none

Referenced by BootTask().

7.327.3.7 ComTransmitPacket()

```
void ComTransmitPacket (
    blt_int8u * data,
    blt_int16u len )
```

Transmits the packet using the xcp transport layer.

Parameters

<i>data</i>	Pointer to the byte buffer with packet data.
<i>len</i>	Number of data bytes that need to be transmitted.

Returns

none

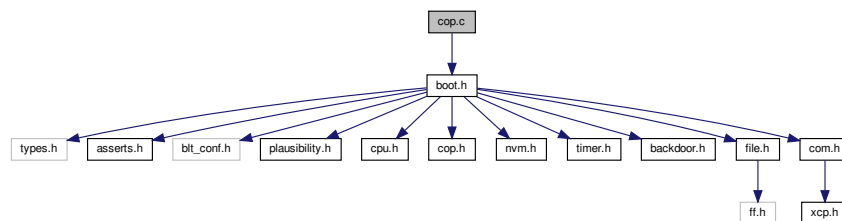
Referenced by XcpTransmitPacket().

7.328 cop.c File Reference

Bootloader watchdog module source file.

```
#include "boot.h"
```

Include dependency graph for cop.c:



Functions

- void [CopInitHook](#) (void)
Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.
- void [CopServiceHook](#) (void)
Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.
- void [CopInit](#) (void)
Watchdog initialization function.
- void [CopService](#) (void)
Watchdog service function to prevent the watchdog from timing out.

7.328.1 Detailed Description

Bootloader watchdog module source file.

7.328.2 Function Documentation

7.328.2.1 CopInit()

```
void CopInit (
    void )
```

Watchdog initialization function.

Returns

none

Referenced by BootInit().

7.328.2.2 CopInitHook()

```
void CopInitHook (
    void )
```

Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.

Returns

none.

Referenced by CopInit().

7.328.2.3 CopService()

```
void CopService (
    void )
```

Watchdog service function to prevent the watchdog from timing out.

Returns

none

Referenced by AssertFailure(), BootTask(), CanDisabledModeEnter(), CanDisabledModeExit(), CanFreeze↔ModeEnter(), CanFreezeModeExit(), CanGetSpeedConfig(), CanInit(), CanTransmitPacket(), CpuMemCopy(), CpuMemSet(), FileFirmwareUpdateLogHook(), FlashAddToBlock(), FlashErase(), FlashEraseSectors(), Flash↔GetSector(), FlashGetSectorBaseAddr(), FlashGetSectorIdx(), FlashGetSectorSize(), FlashWriteBlock(), Usb↔TransmitPacket(), and XcpComputeChecksum().

7.328.2.4 CopServiceHook()

```
void CopServiceHook (
    void )
```

Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.

Returns

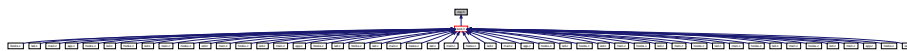
none.

Referenced by CopService().

7.329 cop.h File Reference

Bootloader watchdog module header file.

This graph shows which files directly or indirectly include this file:



Functions

- void [CopInit](#) (void)
Watchdog initialization function.
- void [CopService](#) (void)
Watchdog service function to prevent the watchdog from timing out.

7.329.1 Detailed Description

Bootloader watchdog module header file.

7.329.2 Function Documentation

7.329.2.1 CopInit()

```
void CopInit (
    void )
```

Watchdog initialization function.

Returns

none

Referenced by BootInit().

7.329.2.2 CopService()

```
void CopService (
    void )
```

Watchdog service function to prevent the watchdog from timing out.

Returns

none

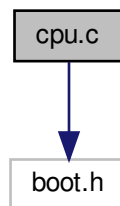
Referenced by AssertFailure(), BootTask(), CanDisabledModeEnter(), CanDisabledModeExit(), CanFreezeModeEnter(), CanFreezeModeExit(), CanGetSpeedConfig(), CanInit(), CanTransmitPacket(), CpuMemCopy(), CpuMemSet(), FileFirmwareUpdateLogHook(), FlashAddToBlock(), FlashErase(), FlashEraseSectors(), FlashGetSector(), FlashGetSectorBaseAddr(), FlashGetSectorIdx(), FlashGetSectorSize(), FlashWriteBlock(), UsbTransmitPacket(), and XcpComputeChecksum().

7.330 cpu.c File Reference

Bootloader cpu module source file.

```
#include "boot.h"
```

Include dependency graph for _template/cpu.c:



Macros

- `#define CPU_USER_PROGRAM_VECTABLE_OFFSET ((blt_addr)NvmGetUserProgBaseAddress())`
Pointer to the user program's vector table.

Functions

- `blt_bool CpuUserProgramStartHook (void)`
Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.
- `void CpuInit (void)`
Initializes the CPU module.
- `void CpuStartUserProgram (void)`
Starts the user program, if one is present. In this case this function does not return.
- `void CpuMemCopy (blt_addr dest, blt_addr src, blt_int16u len)`
Copies data from the source to the destination address.
- `void CpuMemSet (blt_addr dest, blt_int8u value, blt_int16u len)`
Sets the bytes at the destination address to the specified value.

7.330.1 Detailed Description

Bootloader cpu module source file.

7.330.2 Function Documentation

7.330.2.1 CpuInit()

```
void CpuInit (
    void )
```

Initializes the CPU module.

Returns

none.

Referenced by BootInit().

7.330.2.2 CpuMemCopy()

```
void CpuMemCopy (
    blt_addr dest,
    blt_addr src,
    blt_int16u len )
```

Copies data from the source to the destination address.

Parameters

<i>dest</i>	Destination address for the data.
<i>src</i>	Source address of the data.
<i>len</i>	length of the data in bytes.

Returns

none.

Referenced by FlashInitBlock(), UsbReceivePacket(), XcpCmdGetSeed(), XcpCmdShortUpload(), XcpCmdUnlock(), and XcpCmdUpload().

7.330.2.3 CpuMemSet()

```
void CpuMemSet (
    blt_addr dest,
    blt_int8u value,
    blt_int16u len )
```

Sets the bytes at the destination address to the specified value.

Parameters

<i>dest</i>	Destination address for the data.
<i>value</i>	Value to write.
<i>len</i>	Number of bytes to write.

Returns

none.

Referenced by XcpCmdShortUpload(), and XcpCmdUpload().

7.330.2.4 CpuStartUserProgram()

```
void CpuStartUserProgram (
    void )
```

Starts the user program, if one is present. In this case this function does not return.

Returns

none.

Referenced by BackDoorCheck(), BackDoorInit(), FileTask(), and XcpCmdProgramReset().

7.330.2.5 CpuUserProgramStartHook()

```
blt_bool CpuUserProgramStartHook (
    void )
```

Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.

Returns

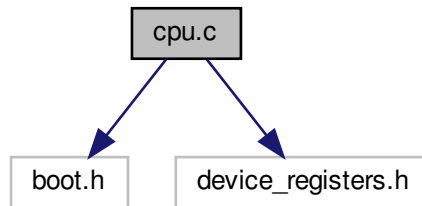
BLT_TRUE if it is okay to start the user program, BLT_FALSE to keep keep the bootloader active.

Referenced by CpuStartUserProgram().

7.331 cpu.c File Reference

Bootloader cpu module source file.

```
#include "boot.h"
#include "device_registers.h"
Include dependency graph for ARMCM0_S32K11/cpu.c:
```



Macros

- `#define CPU_USER_PROGRAM_STARTADDR_PTR ((blt_addr)(NvmGetUserProgBaseAddress() + 0x00000004))`
Pointer to the user program's reset vector.
- `#define CPU_USER_PROGRAM_VECTABLE_OFFSET ((blt_addr)NvmGetUserProgBaseAddress())`
Pointer to the user program's vector table.

Functions

- `blt_bool CpuUserProgramStartHook (void)`
Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.
- `void CpuInit (void)`
Initializes the CPU module.
- `void CpuStartUserProgram (void)`
Starts the user program, if one is present. In this case this function does not return.
- `void CpuMemCopy (blt_addr dest, blt_addr src, blt_int16u len)`
Copies data from the source to the destination address.
- `void CpuMemSet (blt_addr dest, blt_int8u value, blt_int16u len)`
Sets the bytes at the destination address to the specified value.

7.331.1 Detailed Description

Bootloader cpu module source file.

7.331.2 Function Documentation

7.331.2.1 CpuInit()

```
void CpuInit (
    void )
```

Initializes the CPU module.

Returns

none.

7.331.2.2 CpuMemCopy()

```
void CpuMemCopy (
    blt_addr dest,
    blt_addr src,
    blt_int16u len )
```

Copies data from the source to the destination address.

Parameters

<i>dest</i>	Destination address for the data.
<i>src</i>	Source address of the data.
<i>len</i>	length of the data in bytes.

Returns

none.

7.331.2.3 CpuMemSet()

```
void CpuMemSet (
    blt_addr dest,
    blt_int8u value,
    blt_int16u len )
```

Sets the bytes at the destination address to the specified value.

Parameters

<i>dest</i>	Destination address for the data.
<i>value</i>	Value to write.
<i>len</i>	Number of bytes to write.

Returns

none.

7.331.2.4 CpuStartUserProgram()

```
void CpuStartUserProgram (
    void )
```

Starts the user program, if one is present. In this case this function does not return.

Returns

none.

7.331.2.5 CpuUserProgramStartHook()

```
blt_bool CpuUserProgramStartHook (
    void )
```

Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.

Returns

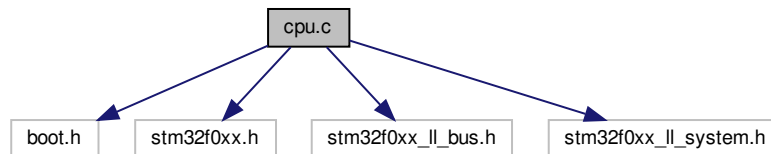
BLT_TRUE if it is okay to start the user program, BLT_FALSE to keep keep the bootloader active.

Referenced by CpuStartUserProgram().

7.332 cpu.c File Reference

Bootloader cpu module source file.

```
#include "boot.h"
#include "stm32f0xx.h"
#include "stm32f0xx_ll_bus.h"
#include "stm32f0xx_ll_system.h"
Include dependency graph for ARMCM0_STM32F0/cpu.c:
```



Macros

- `#define CPU_USER_PROGRAM_STARTADDR_PTR ((blt_addr)(NvmGetUserProgBaseAddress() + 0x00000004))`
Pointer to the user program's reset vector.
- `#define CPU_USER_PROGRAM_VECTABLE_OFFSET ((blt_int32u)NvmGetUserProgBaseAddress())`
Pointer to the user program's vector table.
- `#define CPU_USER_PROGRAM_VECTABLE_SIZE (0xC0u)`
Size in bytes of the user program's vector table.
- `#define CPU_USER_PROGRAM_RAM_BASEADDR ((blt_addr)(0x20000000))`
Start address of the user program's RAM where its vector table will be copied to.

Functions

- `blt_bool CpuUserProgramStartHook (void)`
Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.
- `void CpuInit (void)`
Initializes the CPU module.
- `void CpuStartUserProgram (void)`
Starts the user program, if one is present. In this case this function does not return.
- `void CpuMemCopy (blt_addr dest, blt_addr src, blt_int16u len)`
Copies data from the source to the destination address.
- `void CpuMemSet (blt_addr dest, blt_int8u value, blt_int16u len)`
Sets the bytes at the destination address to the specified value.

7.332.1 Detailed Description

Bootloader cpu module source file.

7.332.2 Function Documentation

7.332.2.1 CpuInit()

```
void CpuInit (
    void )
```

Initializes the CPU module.

Returns

none.

7.332.2.2 CpuMemCopy()

```
void CpuMemCopy (
    blt_addr dest,
    blt_addr src,
    blt_int16u len )
```

Copies data from the source to the destination address.

Parameters

<i>dest</i>	Destination address for the data.
<i>src</i>	Source address of the data.
<i>len</i>	length of the data in bytes.

Returns

none.

Referenced by CpuStartUserProgram().

7.332.2.3 CpuMemSet()

```
void CpuMemSet (
    blt_addr dest,
    blt_int8u value,
    blt_int16u len )
```

Sets the bytes at the destination address to the specified value.

Parameters

<i>dest</i>	Destination address for the data.
<i>value</i>	Value to write.
<i>len</i>	Number of bytes to write.

Returns

none.

7.332.2.4 CpuStartUserProgram()

```
void CpuStartUserProgram (  
    void )
```

Starts the user program, if one is present. In this case this function does not return.

Returns

none.

7.332.2.5 CpuUserProgramStartHook()

```
blt_bool CpuUserProgramStartHook (  
    void )
```

Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.

Returns

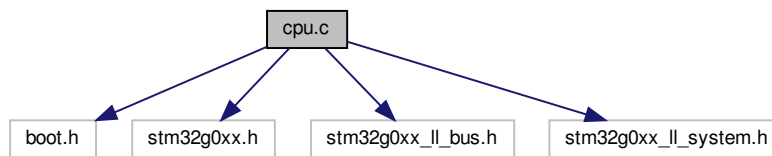
BLT_TRUE if it is okay to start the user program, BLT_FALSE to keep keep the bootloader active.

Referenced by CpuStartUserProgram().

7.333 cpu.c File Reference

Bootloader cpu module source file.

```
#include "boot.h"
#include "stm32g0xx.h"
#include "stm32g0xx_ll_bus.h"
#include "stm32g0xx_ll_system.h"
Include dependency graph for ARMCM0_STM32G0/cpu.c:
```



Macros

- `#define CPU_USER_PROGRAM_STARTADDR_PTR ((blt_addr)(NvmGetUserProgBaseAddress() + 0x00000004))`
Pointer to the user program's reset vector.
- `#define CPU_USER_PROGRAM_VECTABLE_OFFSET ((blt_int32u)NvmGetUserProgBaseAddress())`
Pointer to the user program's vector table.

Functions

- `blt_bool CpuUserProgramStartHook (void)`
Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.
- `void CpuInit (void)`
Initializes the CPU module.
- `void CpuStartUserProgram (void)`
Starts the user program, if one is present. In this case this function does not return.
- `void CpuMemCopy (blt_addr dest, blt_addr src, blt_int16u len)`
Copies data from the source to the destination address.
- `void CpuMemSet (blt_addr dest, blt_int8u value, blt_int16u len)`
Sets the bytes at the destination address to the specified value.

7.333.1 Detailed Description

Bootloader cpu module source file.

7.333.2 Function Documentation

7.333.2.1 CpuInit()

```
void CpuInit (
    void )
```

Initializes the CPU module.

Returns

none.

7.333.2.2 CpuMemCopy()

```
void CpuMemCopy (
    blt_addr dest,
    blt_addr src,
    blt_int16u len )
```

Copies data from the source to the destination address.

Parameters

<i>dest</i>	Destination address for the data.
<i>src</i>	Source address of the data.
<i>len</i>	length of the data in bytes.

Returns

none.

7.333.2.3 CpuMemSet()

```
void CpuMemSet (
    blt_addr dest,
    blt_int8u value,
    blt_int16u len )
```

Sets the bytes at the destination address to the specified value.

Parameters

<i>dest</i>	Destination address for the data.
<i>value</i>	Value to write.
<i>len</i>	Number of bytes to write.

Returns

none.

7.333.2.4 CpuStartUserProgram()

```
void CpuStartUserProgram (
    void )
```

Starts the user program, if one is present. In this case this function does not return.

Returns

none.

7.333.2.5 CpuUserProgramStartHook()

```
blt_bool CpuUserProgramStartHook (
    void )
```

Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.

Returns

BLT_TRUE if it is okay to start the user program, BLT_FALSE to keep keep the bootloader active.

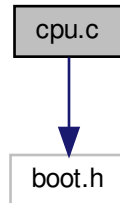
Referenced by CpuStartUserProgram().

7.334 cpu.c File Reference

Bootloader cpu module source file.

```
#include "boot.h"
```

Include dependency graph for ARMCM0_XMC1/cpu.c:



Macros

- `#define CPU_USER_PROGRAM_STARTADDR_PTR ((blt_addr)(NvmGetUserProgBaseAddress() + 0x00000004))`
Pointer to the user program's reset vector.
- `#define CPU_USER_PROGRAM_VECTABLE_OFFSET ((blt_int32u)NvmGetUserProgBaseAddress())`
Pointer to the user program's vector table.

Functions

- `blt_bool CpuUserProgramStartHook (void)`
Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.
- `void CpuInit (void)`
Initializes the CPU module.
- `void CpuStartUserProgram (void)`
Starts the user program, if one is present. In this case this function does not return.
- `void CpuMemCopy (blt_addr dest, blt_addr src, blt_int16u len)`
Copies data from the source to the destination address.
- `void CpuMemSet (blt_addr dest, blt_int8u value, blt_int16u len)`
Sets the bytes at the destination address to the specified value.

7.334.1 Detailed Description

Bootloader cpu module source file.

7.334.2 Function Documentation

7.334.2.1 CpuInit()

```
void CpuInit (
    void )
```

Initializes the CPU module.

Returns

none.

7.334.2.2 CpuMemCopy()

```
void CpuMemCopy (
    blt_addr dest,
    blt_addr src,
    blt_int16u len )
```

Copies data from the source to the destination address.

Parameters

<i>dest</i>	Destination address for the data.
<i>src</i>	Source address of the data.
<i>len</i>	length of the data in bytes.

Returns

none.

7.334.2.3 CpuMemSet()

```
void CpuMemSet (
    blt_addr dest,
    blt_int8u value,
    blt_int16u len )
```

Sets the bytes at the destination address to the specified value.

Parameters

<i>dest</i>	Destination address for the data.
<i>value</i>	Value to write.
<i>len</i>	Number of bytes to write.

Returns

none.

7.334.2.4 CpuStartUserProgram()

```
void CpuStartUserProgram (  
    void )
```

Starts the user program, if one is present. In this case this function does not return.

Returns

none.

7.334.2.5 CpuUserProgramStartHook()

```
blt_bool CpuUserProgramStartHook (  
    void )
```

Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.

Returns

BLT_TRUE if it is okay to start the user program, BLT_FALSE to keep keep the bootloader active.

Referenced by CpuStartUserProgram().

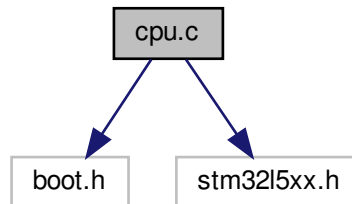
7.335 cpu.c File Reference

Bootloader cpu module source file.

```
#include "boot.h"
```

```
#include "stm32l5xx.h"
```

Include dependency graph for ARMCM33_STM32L5/cpu.c:



Macros

- `#define CPU_USER_PROGRAM_STARTADDR_PTR ((blt_addr)(NvmGetUserProgBaseAddress() + 0x00000004))`
Pointer to the user program's reset vector.
- `#define CPU_USER_PROGRAM_VECTABLE_OFFSET ((blt_int32u)NvmGetUserProgBaseAddress())`
Pointer to the user program's vector table.

Functions

- `blt_bool CpuUserProgramStartHook (void)`
Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.
- `void CpuInit (void)`
Initializes the CPU module.
- `void CpuStartUserProgram (void)`
Starts the user program, if one is present. In this case this function does not return.
- `void CpuMemCopy (blt_addr dest, blt_addr src, blt_int16u len)`
Copies data from the source to the destination address.
- `void CpuMemSet (blt_addr dest, blt_int8u value, blt_int16u len)`
Sets the bytes at the destination address to the specified value.

7.335.1 Detailed Description

Bootloader cpu module source file.

7.335.2 Function Documentation

7.335.2.1 CpuInit()

```
void CpuInit (
    void )
```

Initializes the CPU module.

Returns

none.

7.335.2.2 CpuMemCopy()

```
void CpuMemCopy (
    blt_addr dest,
    blt_addr src,
    blt_int16u len )
```

Copies data from the source to the destination address.

Parameters

<i>dest</i>	Destination address for the data.
<i>src</i>	Source address of the data.
<i>len</i>	length of the data in bytes.

Returns

none.

7.335.2.3 CpuMemSet()

```
void CpuMemSet (
    blt_addr dest,
    blt_int8u value,
    blt_int16u len )
```

Sets the bytes at the destination address to the specified value.

Parameters

<i>dest</i>	Destination address for the data.
<i>value</i>	Value to write.
<i>len</i>	Number of bytes to write.

Returns

none.

7.335.2.4 CpuStartUserProgram()

```
void CpuStartUserProgram (  
    void )
```

Starts the user program, if one is present. In this case this function does not return.

Returns

none.

7.335.2.5 CpuUserProgramStartHook()

```
blt_bool CpuUserProgramStartHook (  
    void )
```

Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.

Returns

BLT_TRUE if it is okay to start the user program, BLT_FALSE to keep keep the bootloader active.

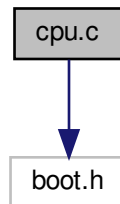
Referenced by CpuStartUserProgram().

7.336 cpu.c File Reference

Bootloader cpu module source file.

```
#include "boot.h"
```

Include dependency graph for ARMCM3_EFM32/cpu.c:



Macros

- `#define CPU_USER_PROGRAM_STARTADDR_PTR ((blt_addr)(NvmGetUserProgBaseAddress() + 0x00000004))`
Pointer to the user program's reset vector.
- `#define CPU_USER_PROGRAM_VECTABLE_OFFSET ((blt_int32u)NvmGetUserProgBaseAddress())`
Pointer to the user program's vector table.
- `#define SCB_VTOR (*(volatile blt_int32u *) 0xE00ED08)`
Vector table offset register.

Functions

- `blt_bool CpuUserProgramStartHook (void)`
Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.
- `void CpuInit (void)`
Initializes the CPU module.
- `void CpuStartUserProgram (void)`
Starts the user program, if one is present. In this case this function does not return.
- `void CpuMemCopy (blt_addr dest, blt_addr src, blt_int16u len)`
Copies data from the source to the destination address.
- `void CpuMemSet (blt_addr dest, blt_int8u value, blt_int16u len)`
Sets the bytes at the destination address to the specified value.

7.336.1 Detailed Description

Bootloader cpu module source file.

7.336.2 Function Documentation

7.336.2.1 CpuInit()

```
void CpuInit (
    void )
```

Initializes the CPU module.

Returns

none.

7.336.2.2 CpuMemCopy()

```
void CpuMemCopy (
    blt_addr dest,
    blt_addr src,
    blt_int16u len )
```

Copies data from the source to the destination address.

Parameters

<i>dest</i>	Destination address for the data.
<i>src</i>	Source address of the data.
<i>len</i>	length of the data in bytes.

Returns

none.

7.336.2.3 CpuMemSet()

```
void CpuMemSet (
    blt_addr dest,
    blt_int8u value,
    blt_int16u len )
```

Sets the bytes at the destination address to the specified value.

Parameters

<i>dest</i>	Destination address for the data.
<i>value</i>	Value to write.
<i>len</i>	Number of bytes to write.

Returns

none.

7.336.2.4 CpuStartUserProgram()

```
void CpuStartUserProgram (  
    void )
```

Starts the user program, if one is present. In this case this function does not return.

Returns

none.

7.336.2.5 CpuUserProgramStartHook()

```
blt_bool CpuUserProgramStartHook (  
    void )
```

Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.

Returns

BLT_TRUE if it is okay to start the user program, BLT_FALSE to keep keep the bootloader active.

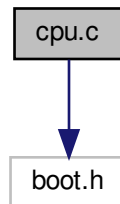
Referenced by CpuStartUserProgram().

7.337 cpu.c File Reference

Bootloader cpu module source file.

```
#include "boot.h"
```

Include dependency graph for ARMCM3_LM3S/cpu.c:



Macros

- `#define CPU_USER_PROGRAM_STARTADDR_PTR ((blt_addr)(NvmGetUserProgBaseAddress() + 0x00000004))`
Pointer to the user program's reset vector.
- `#define CPU_USER_PROGRAM_VECTABLE_OFFSET ((blt_int32u)NvmGetUserProgBaseAddress())`
Pointer to the user program's vector table.
- `#define SCB_VTOR (*(volatile blt_int32u *) 0xE00ED08)`
Vector table offset register.

Functions

- `blt_bool CpuUserProgramStartHook (void)`
Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.
- `void CpuInit (void)`
Initializes the CPU module.
- `void CpuStartUserProgram (void)`
Starts the user program, if one is present. In this case this function does not return.
- `void CpuMemCopy (blt_addr dest, blt_addr src, blt_int16u len)`
Copies data from the source to the destination address.
- `void CpuMemSet (blt_addr dest, blt_int8u value, blt_int16u len)`
Sets the bytes at the destination address to the specified value.

7.337.1 Detailed Description

Bootloader cpu module source file.

7.337.2 Function Documentation

7.337.2.1 CpuInit()

```
void CpuInit (
    void )
```

Initializes the CPU module.

Returns

none.

7.337.2.2 CpuMemCopy()

```
void CpuMemCopy (
    blt_addr dest,
    blt_addr src,
    blt_int16u len )
```

Copies data from the source to the destination address.

Parameters

<i>dest</i>	Destination address for the data.
<i>src</i>	Source address of the data.
<i>len</i>	length of the data in bytes.

Returns

none.

7.337.2.3 CpuMemSet()

```
void CpuMemSet (
    blt_addr dest,
    blt_int8u value,
    blt_int16u len )
```

Sets the bytes at the destination address to the specified value.

Parameters

<i>dest</i>	Destination address for the data.
<i>value</i>	Value to write.
<i>len</i>	Number of bytes to write.

Returns

none.

7.337.2.4 CpuStartUserProgram()

```
void CpuStartUserProgram (
    void )
```

Starts the user program, if one is present. In this case this function does not return.

Returns

none.

7.337.2.5 CpuUserProgramStartHook()

```
blt_bool CpuUserProgramStartHook (
    void )
```

Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.

Returns

BLT_TRUE if it is okay to start the user program, BLT_FALSE to keep keep the bootloader active.

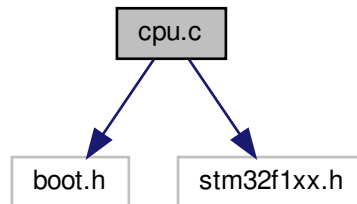
Referenced by CpuStartUserProgram().

7.338 cpu.c File Reference

Bootloader cpu module source file.

```
#include "boot.h"
#include "stm32f1xx.h"
```

Include dependency graph for ARMCM3_STM32F1/cpu.c:



Macros

- `#define CPU_USER_PROGRAM_STARTADDR_PTR ((blt_addr)(NvmGetUserProgBaseAddress() + 0x00000004))`
Pointer to the user program's reset vector.
- `#define CPU_USER_PROGRAM_VECTABLE_OFFSET ((blt_int32u)NvmGetUserProgBaseAddress())`
Pointer to the user program's vector table.

Functions

- `blt_bool CpuUserProgramStartHook (void)`
Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.
- `void CpuInit (void)`
Initializes the CPU module.
- `void CpuStartUserProgram (void)`
Starts the user program, if one is present. In this case this function does not return.
- `void CpuMemCopy (blt_addr dest, blt_addr src, blt_int16u len)`
Copies data from the source to the destination address.
- `void CpuMemSet (blt_addr dest, blt_int8u value, blt_int16u len)`
Sets the bytes at the destination address to the specified value.

7.338.1 Detailed Description

Bootloader cpu module source file.

7.338.2 Function Documentation

7.338.2.1 CpuInit()

```
void CpuInit (
    void )
```

Initializes the CPU module.

Returns

none.

7.338.2.2 CpuMemCopy()

```
void CpuMemCopy (
    blt_addr dest,
    blt_addr src,
    blt_int16u len )
```

Copies data from the source to the destination address.

Parameters

<i>dest</i>	Destination address for the data.
<i>src</i>	Source address of the data.
<i>len</i>	length of the data in bytes.

Returns

none.

7.338.2.3 CpuMemSet()

```
void CpuMemSet (
    blt_addr dest,
    blt_int8u value,
    blt_int16u len )
```

Sets the bytes at the destination address to the specified value.

Parameters

<i>dest</i>	Destination address for the data.
<i>value</i>	Value to write.
<i>len</i>	Number of bytes to write.

Returns

none.

7.338.2.4 CpuStartUserProgram()

```
void CpuStartUserProgram (  
    void )
```

Starts the user program, if one is present. In this case this function does not return.

Returns

none.

7.338.2.5 CpuUserProgramStartHook()

```
blt_bool CpuUserProgramStartHook (  
    void )
```

Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.

Returns

BLT_TRUE if it is okay to start the user program, BLT_FALSE to keep keep the bootloader active.

Referenced by CpuStartUserProgram().

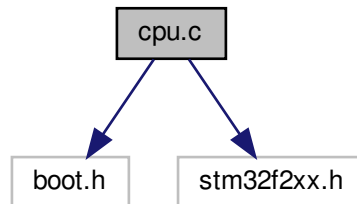
7.339 cpu.c File Reference

Bootloader cpu module source file.

```
#include "boot.h"
```

```
#include "stm32f2xx.h"
```

Include dependency graph for ARMCM3_STM32F2/cpu.c:



Macros

- `#define CPU_USER_PROGRAM_STARTADDR_PTR ((blt_addr)(NvmGetUserProgBaseAddress() + 0x00000004))`
Pointer to the user program's reset vector.
- `#define CPU_USER_PROGRAM_VECTABLE_OFFSET ((blt_int32u)NvmGetUserProgBaseAddress())`
Pointer to the user program's vector table.

Functions

- `blt_bool CpuUserProgramStartHook (void)`
Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.
- `void CpuInit (void)`
Initializes the CPU module.
- `void CpuStartUserProgram (void)`
Starts the user program, if one is present. In this case this function does not return.
- `void CpuMemCopy (blt_addr dest, blt_addr src, blt_int16u len)`
Copies data from the source to the destination address.
- `void CpuMemSet (blt_addr dest, blt_int8u value, blt_int16u len)`
Sets the bytes at the destination address to the specified value.

7.339.1 Detailed Description

Bootloader cpu module source file.

7.339.2 Function Documentation

7.339.2.1 CpuInit()

```
void CpuInit (
    void )
```

Initializes the CPU module.

Returns

none.

7.339.2.2 CpuMemCopy()

```
void CpuMemCopy (
    blt_addr dest,
    blt_addr src,
    blt_int16u len )
```

Copies data from the source to the destination address.

Parameters

<i>dest</i>	Destination address for the data.
<i>src</i>	Source address of the data.
<i>len</i>	length of the data in bytes.

Returns

none.

7.339.2.3 CpuMemSet()

```
void CpuMemSet (
    blt_addr dest,
    blt_int8u value,
    blt_int16u len )
```

Sets the bytes at the destination address to the specified value.

Parameters

<i>dest</i>	Destination address for the data.
<i>value</i>	Value to write.
<i>len</i>	Number of bytes to write.

Returns

none.

7.339.2.4 CpuStartUserProgram()

```
void CpuStartUserProgram (
    void )
```

Starts the user program, if one is present. In this case this function does not return.

Returns

none.

7.339.2.5 CpuUserProgramStartHook()

```
blt_bool CpuUserProgramStartHook (
    void )
```

Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.

Returns

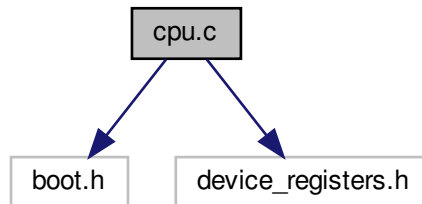
BLT_TRUE if it is okay to start the user program, BLT_FALSE to keep keep the bootloader active.

Referenced by CpuStartUserProgram().

7.340 cpu.c File Reference

Bootloader cpu module source file.

```
#include "boot.h"
#include "device_registers.h"
Include dependency graph for ARMCM4_S32K14/cpu.c:
```



Macros

- `#define CPU_USER_PROGRAM_STARTADDR_PTR ((blt_addr)(NvmGetUserProgBaseAddress() + 0x00000004))`
Pointer to the user program's reset vector.
- `#define CPU_USER_PROGRAM_VECTABLE_OFFSET ((blt_addr)NvmGetUserProgBaseAddress())`
Pointer to the user program's vector table.

Functions

- `blt_bool CpuUserProgramStartHook (void)`
Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.
- `void CpuInit (void)`
Initializes the CPU module.
- `void CpuStartUserProgram (void)`
Starts the user program, if one is present. In this case this function does not return.
- `void CpuMemCopy (blt_addr dest, blt_addr src, blt_int16u len)`
Copies data from the source to the destination address.
- `void CpuMemSet (blt_addr dest, blt_int8u value, blt_int16u len)`
Sets the bytes at the destination address to the specified value.

7.340.1 Detailed Description

Bootloader cpu module source file.

7.340.2 Function Documentation

7.340.2.1 CpuInit()

```
void CpuInit (
    void )
```

Initializes the CPU module.

Returns

none.

7.340.2.2 CpuMemCopy()

```
void CpuMemCopy (
    blt_addr dest,
    blt_addr src,
    blt_int16u len )
```

Copies data from the source to the destination address.

Parameters

<i>dest</i>	Destination address for the data.
<i>src</i>	Source address of the data.
<i>len</i>	length of the data in bytes.

Returns

none.

7.340.2.3 CpuMemSet()

```
void CpuMemSet (
    blt_addr dest,
    blt_int8u value,
    blt_int16u len )
```

Sets the bytes at the destination address to the specified value.

Parameters

<i>dest</i>	Destination address for the data.
<i>value</i>	Value to write.
<i>len</i>	Number of bytes to write.

Returns

none.

7.340.2.4 CpuStartUserProgram()

```
void CpuStartUserProgram (  
    void )
```

Starts the user program, if one is present. In this case this function does not return.

Returns

none.

7.340.2.5 CpuUserProgramStartHook()

```
blt_bool CpuUserProgramStartHook (  
    void )
```

Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.

Returns

BLT_TRUE if it is okay to start the user program, BLT_FALSE to keep keep the bootloader active.

Referenced by CpuStartUserProgram().

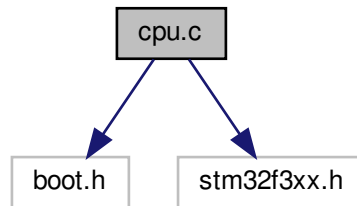
7.341 cpu.c File Reference

Bootloader cpu module source file.

```
#include "boot.h"
```

```
#include "stm32f3xx.h"
```

Include dependency graph for ARMCM4_STM32F3/cpu.c:



Macros

- `#define CPU_USER_PROGRAM_STARTADDR_PTR ((blt_addr)(NvmGetUserProgBaseAddress() + 0x00000004))`
Pointer to the user program's reset vector.
- `#define CPU_USER_PROGRAM_VECTABLE_OFFSET ((blt_int32u)NvmGetUserProgBaseAddress())`
Pointer to the user program's vector table.

Functions

- `blt_bool CpuUserProgramStartHook (void)`
Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.
- `void CpuInit (void)`
Initializes the CPU module.
- `void CpuStartUserProgram (void)`
Starts the user program, if one is present. In this case this function does not return.
- `void CpuMemCopy (blt_addr dest, blt_addr src, blt_int16u len)`
Copies data from the source to the destination address.
- `void CpuMemSet (blt_addr dest, blt_int8u value, blt_int16u len)`
Sets the bytes at the destination address to the specified value.

7.341.1 Detailed Description

Bootloader cpu module source file.

7.341.2 Function Documentation

7.341.2.1 CpuInit()

```
void CpuInit (
    void )
```

Initializes the CPU module.

Returns

none.

7.341.2.2 CpuMemCopy()

```
void CpuMemCopy (
    blt_addr dest,
    blt_addr src,
    blt_int16u len )
```

Copies data from the source to the destination address.

Parameters

<i>dest</i>	Destination address for the data.
<i>src</i>	Source address of the data.
<i>len</i>	length of the data in bytes.

Returns

none.

7.341.2.3 CpuMemSet()

```
void CpuMemSet (
    blt_addr dest,
    blt_int8u value,
    blt_int16u len )
```

Sets the bytes at the destination address to the specified value.

Parameters

<i>dest</i>	Destination address for the data.
<i>value</i>	Value to write.
<i>len</i>	Number of bytes to write.

Returns

none.

7.341.2.4 CpuStartUserProgram()

```
void CpuStartUserProgram (
    void )
```

Starts the user program, if one is present. In this case this function does not return.

Returns

none.

7.341.2.5 CpuUserProgramStartHook()

```
blt_bool CpuUserProgramStartHook (
    void )
```

Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.

Returns

BLT_TRUE if it is okay to start the user program, BLT_FALSE to keep keep the bootloader active.

Referenced by CpuStartUserProgram().

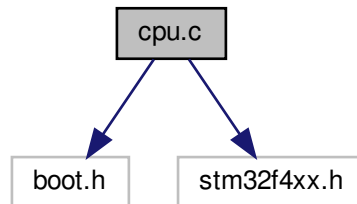
7.342 cpu.c File Reference

Bootloader cpu module source file.

```
#include "boot.h"
```

```
#include "stm32f4xx.h"
```

Include dependency graph for ARMCM4_STM32F4/cpu.c:



Macros

- `#define CPU_USER_PROGRAM_STARTADDR_PTR ((blt_addr)(NvmGetUserProgBaseAddress() + 0x00000004))`
Pointer to the user program's reset vector.
- `#define CPU_USER_PROGRAM_VECTABLE_OFFSET ((blt_int32u)NvmGetUserProgBaseAddress())`
Pointer to the user program's vector table.

Functions

- `blt_bool CpuUserProgramStartHook (void)`
Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.
- `void CpuInit (void)`
Initializes the CPU module.
- `void CpuStartUserProgram (void)`
Starts the user program, if one is present. In this case this function does not return.
- `void CpuMemCopy (blt_addr dest, blt_addr src, blt_int16u len)`
Copies data from the source to the destination address.
- `void CpuMemSet (blt_addr dest, blt_int8u value, blt_int16u len)`
Sets the bytes at the destination address to the specified value.

7.342.1 Detailed Description

Bootloader cpu module source file.

7.342.2 Function Documentation

7.342.2.1 CpuInit()

```
void CpuInit (
    void )
```

Initializes the CPU module.

Returns

none.

7.342.2.2 CpuMemCopy()

```
void CpuMemCopy (
    blt_addr dest,
    blt_addr src,
    blt_int16u len )
```

Copies data from the source to the destination address.

Parameters

<i>dest</i>	Destination address for the data.
<i>src</i>	Source address of the data.
<i>len</i>	length of the data in bytes.

Returns

none.

7.342.2.3 CpuMemSet()

```
void CpuMemSet (
    blt_addr dest,
    blt_int8u value,
    blt_int16u len )
```

Sets the bytes at the destination address to the specified value.

Parameters

<i>dest</i>	Destination address for the data.
<i>value</i>	Value to write.
<i>len</i>	Number of bytes to write.

Returns

none.

7.342.2.4 CpuStartUserProgram()

```
void CpuStartUserProgram (  
    void )
```

Starts the user program, if one is present. In this case this function does not return.

Returns

none.

7.342.2.5 CpuUserProgramStartHook()

```
blt_bool CpuUserProgramStartHook (  
    void )
```

Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.

Returns

BLT_TRUE if it is okay to start the user program, BLT_FALSE to keep keep the bootloader active.

Referenced by CpuStartUserProgram().

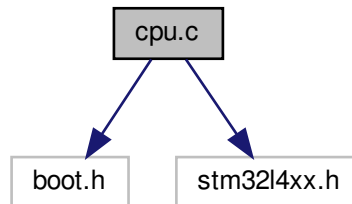
7.343 cpu.c File Reference

Bootloader cpu module source file.

```
#include "boot.h"
```

```
#include "stm32l4xx.h"
```

Include dependency graph for ARMCM4_STM32L4/cpu.c:



Macros

- `#define CPU_USER_PROGRAM_STARTADDR_PTR ((blt_addr)(NvmGetUserProgBaseAddress() + 0x00000004))`
Pointer to the user program's reset vector.
- `#define CPU_USER_PROGRAM_VECTABLE_OFFSET ((blt_int32u)NvmGetUserProgBaseAddress())`
Pointer to the user program's vector table.

Functions

- `blt_bool CpuUserProgramStartHook (void)`
Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.
- `void CpuInit (void)`
Initializes the CPU module.
- `void CpuStartUserProgram (void)`
Starts the user program, if one is present. In this case this function does not return.
- `void CpuMemCopy (blt_addr dest, blt_addr src, blt_int16u len)`
Copies data from the source to the destination address.
- `void CpuMemSet (blt_addr dest, blt_int8u value, blt_int16u len)`
Sets the bytes at the destination address to the specified value.

7.343.1 Detailed Description

Bootloader cpu module source file.

7.343.2 Function Documentation

7.343.2.1 CpuInit()

```
void CpuInit (
    void )
```

Initializes the CPU module.

Returns

none.

7.343.2.2 CpuMemCopy()

```
void CpuMemCopy (
    blt_addr dest,
    blt_addr src,
    blt_int16u len )
```

Copies data from the source to the destination address.

Parameters

<i>dest</i>	Destination address for the data.
<i>src</i>	Source address of the data.
<i>len</i>	length of the data in bytes.

Returns

none.

7.343.2.3 CpuMemSet()

```
void CpuMemSet (
    blt_addr dest,
    blt_int8u value,
    blt_int16u len )
```

Sets the bytes at the destination address to the specified value.

Parameters

<i>dest</i>	Destination address for the data.
<i>value</i>	Value to write.
<i>len</i>	Number of bytes to write.

Returns

none.

7.343.2.4 CpuStartUserProgram()

```
void CpuStartUserProgram (
    void )
```

Starts the user program, if one is present. In this case this function does not return.

Returns

none.

7.343.2.5 CpuUserProgramStartHook()

```
blt_bool CpuUserProgramStartHook (
    void )
```

Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.

Returns

BLT_TRUE if it is okay to start the user program, BLT_FALSE to keep keep the bootloader active.

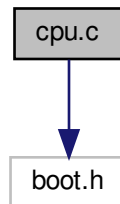
Referenced by CpuStartUserProgram().

7.344 cpu.c File Reference

Bootloader cpu module source file.

```
#include "boot.h"
```

Include dependency graph for ARMCM4_TM4C/cpu.c:



Macros

- `#define CPU_USER_PROGRAM_STARTADDR_PTR ((blt_addr)(NvmGetUserProgBaseAddress() + 0x00000004))`
Pointer to the user program's reset vector.
- `#define CPU_USER_PROGRAM_VECTABLE_OFFSET ((blt_int32u)NvmGetUserProgBaseAddress())`
Pointer to the user program's vector table.
- `#define SCB_VTOR (*(volatile blt_int32u *) 0xE00ED08)`
Vector table offset register.

Functions

- `blt_bool CpuUserProgramStartHook (void)`
Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.
- `void CpuInit (void)`
Initializes the CPU module.
- `void CpuStartUserProgram (void)`
Starts the user program, if one is present. In this case this function does not return.
- `void CpuMemCopy (blt_addr dest, blt_addr src, blt_int16u len)`
Copies data from the source to the destination address.
- `void CpuMemSet (blt_addr dest, blt_int8u value, blt_int16u len)`
Sets the bytes at the destination address to the specified value.

7.344.1 Detailed Description

Bootloader cpu module source file.

7.344.2 Function Documentation

7.344.2.1 CpuInit()

```
void CpuInit (
    void )
```

Initializes the CPU module.

Returns

none.

7.344.2.2 CpuMemCopy()

```
void CpuMemCopy (
    blt_addr dest,
    blt_addr src,
    blt_int16u len )
```

Copies data from the source to the destination address.

Parameters

<i>dest</i>	Destination address for the data.
<i>src</i>	Source address of the data.
<i>len</i>	length of the data in bytes.

Returns

none.

7.344.2.3 CpuMemSet()

```
void CpuMemSet (
    blt_addr dest,
    blt_int8u value,
    blt_int16u len )
```

Sets the bytes at the destination address to the specified value.

Parameters

<i>dest</i>	Destination address for the data.
<i>value</i>	Value to write.
<i>len</i>	Number of bytes to write.

Returns

none.

7.344.2.4 CpuStartUserProgram()

```
void CpuStartUserProgram (  
    void )
```

Starts the user program, if one is present. In this case this function does not return.

Returns

none.

7.344.2.5 CpuUserProgramStartHook()

```
blt_bool CpuUserProgramStartHook (  
    void )
```

Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.

Returns

BLT_TRUE if it is okay to start the user program, BLT_FALSE to keep keep the bootloader active.

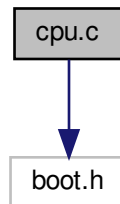
Referenced by CpuStartUserProgram().

7.345 cpu.c File Reference

Bootloader cpu module source file.

```
#include "boot.h"
```

Include dependency graph for ARMCM4_XMC4/cpu.c:



Macros

- `#define CPU_USER_PROGRAM_STARTADDR_PTR ((blt_addr)(NvmGetUserProgBaseAddress() + 0x00000004))`
Pointer to the user program's reset vector.
- `#define CPU_USER_PROGRAM_VECTABLE_OFFSET ((blt_int32u)NvmGetUserProgBaseAddress())`
Pointer to the user program's vector table.
- `#define SCB_VTOR (*(volatile blt_int32u *) 0xE00ED08)`
Vector table offset register.

Functions

- `blt_bool CpuUserProgramStartHook (void)`
Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.
- `void CpuInit (void)`
Initializes the CPU module.
- `void CpuStartUserProgram (void)`
Starts the user program, if one is present. In this case this function does not return.
- `void CpuMemCopy (blt_addr dest, blt_addr src, blt_int16u len)`
Copies data from the source to the destination address.
- `void CpuMemSet (blt_addr dest, blt_int8u value, blt_int16u len)`
Sets the bytes at the destination address to the specified value.

7.345.1 Detailed Description

Bootloader cpu module source file.

7.345.2 Function Documentation

7.345.2.1 CpuInit()

```
void CpuInit (
    void )
```

Initializes the CPU module.

Returns

none.

7.345.2.2 CpuMemCopy()

```
void CpuMemCopy (
    blt_addr dest,
    blt_addr src,
    blt_int16u len )
```

Copies data from the source to the destination address.

Parameters

<i>dest</i>	Destination address for the data.
<i>src</i>	Source address of the data.
<i>len</i>	length of the data in bytes.

Returns

none.

7.345.2.3 CpuMemSet()

```
void CpuMemSet (
    blt_addr dest,
    blt_int8u value,
    blt_int16u len )
```

Sets the bytes at the destination address to the specified value.

Parameters

<i>dest</i>	Destination address for the data.
<i>value</i>	Value to write.
<i>len</i>	Number of bytes to write.

Returns

none.

7.345.2.4 CpuStartUserProgram()

```
void CpuStartUserProgram (
    void )
```

Starts the user program, if one is present. In this case this function does not return.

Returns

none.

7.345.2.5 CpuUserProgramStartHook()

```
blt_bool CpuUserProgramStartHook (
    void )
```

Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.

Returns

BLT_TRUE if it is okay to start the user program, BLT_FALSE to keep keep the bootloader active.

Referenced by CpuStartUserProgram().

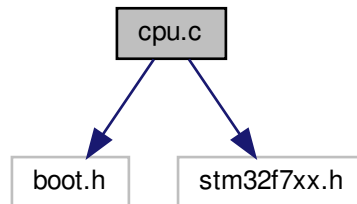
7.346 cpu.c File Reference

Bootloader cpu module source file.

```
#include "boot.h"
```

```
#include "stm32f7xx.h"
```

Include dependency graph for ARMCM7_STM32F7/cpu.c:



Macros

- `#define CPU_USER_PROGRAM_STARTADDR_PTR ((blt_addr)(NvmGetUserProgBaseAddress() + 0x00000004))`
Pointer to the user program's reset vector.
- `#define CPU_USER_PROGRAM_VECTABLE_OFFSET ((blt_int32u)NvmGetUserProgBaseAddress())`
Pointer to the user program's vector table.

Functions

- `blt_bool CpuUserProgramStartHook (void)`
Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.
- `void CpuInit (void)`
Initializes the CPU module.
- `void CpuStartUserProgram (void)`
Starts the user program, if one is present. In this case this function does not return.
- `void CpuMemCopy (blt_addr dest, blt_addr src, blt_int16u len)`
Copies data from the source to the destination address.
- `void CpuMemSet (blt_addr dest, blt_int8u value, blt_int16u len)`
Sets the bytes at the destination address to the specified value.

7.346.1 Detailed Description

Bootloader cpu module source file.

7.346.2 Function Documentation

7.346.2.1 CpuInit()

```
void CpuInit (
    void )
```

Initializes the CPU module.

Returns

none.

7.346.2.2 CpuMemCopy()

```
void CpuMemCopy (
    blt_addr dest,
    blt_addr src,
    blt_int16u len )
```

Copies data from the source to the destination address.

Parameters

<i>dest</i>	Destination address for the data.
<i>src</i>	Source address of the data.
<i>len</i>	length of the data in bytes.

Returns

none.

7.346.2.3 CpuMemSet()

```
void CpuMemSet (
    blt_addr dest,
    blt_int8u value,
    blt_int16u len )
```

Sets the bytes at the destination address to the specified value.

Parameters

<i>dest</i>	Destination address for the data.
<i>value</i>	Value to write.
<i>len</i>	Number of bytes to write.

Returns

none.

7.346.2.4 CpuStartUserProgram()

```
void CpuStartUserProgram (  
    void )
```

Starts the user program, if one is present. In this case this function does not return.

Returns

none.

7.346.2.5 CpuUserProgramStartHook()

```
blt_bool CpuUserProgramStartHook (  
    void )
```

Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.

Returns

BLT_TRUE if it is okay to start the user program, BLT_FALSE to keep keep the bootloader active.

Referenced by CpuStartUserProgram().

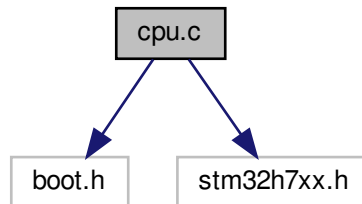
7.347 cpu.c File Reference

Bootloader cpu module source file.

```
#include "boot.h"
```

```
#include "stm32h7xx.h"
```

Include dependency graph for ARMCM7_STM32H7/cpu.c:



Macros

- `#define CPU_USER_PROGRAM_STARTADDR_PTR ((blt_addr)(NvmGetUserProgBaseAddress() + 0x00000004))`
Pointer to the user program's reset vector.
- `#define CPU_USER_PROGRAM_VECTABLE_OFFSET ((blt_addr)NvmGetUserProgBaseAddress())`
Pointer to the user program's vector table.

Functions

- `blt_bool CpuUserProgramStartHook (void)`
Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.
- `void CpuInit (void)`
Initializes the CPU module.
- `void CpuStartUserProgram (void)`
Starts the user program, if one is present. In this case this function does not return.
- `void CpuMemCopy (blt_addr dest, blt_addr src, blt_int16u len)`
Copies data from the source to the destination address.
- `void CpuMemSet (blt_addr dest, blt_int8u value, blt_int16u len)`
Sets the bytes at the destination address to the specified value.

7.347.1 Detailed Description

Bootloader cpu module source file.

7.347.2 Function Documentation

7.347.2.1 CpuInit()

```
void CpuInit (
    void )
```

Initializes the CPU module.

Returns

none.

7.347.2.2 CpuMemCopy()

```
void CpuMemCopy (
    blt_addr dest,
    blt_addr src,
    blt_int16u len )
```

Copies data from the source to the destination address.

Parameters

<i>dest</i>	Destination address for the data.
<i>src</i>	Source address of the data.
<i>len</i>	length of the data in bytes.

Returns

none.

7.347.2.3 CpuMemSet()

```
void CpuMemSet (
    blt_addr dest,
    blt_int8u value,
    blt_int16u len )
```

Sets the bytes at the destination address to the specified value.

Parameters

<i>dest</i>	Destination address for the data.
<i>value</i>	Value to write.
<i>len</i>	Number of bytes to write.

Returns

none.

7.347.2.4 CpuStartUserProgram()

```
void CpuStartUserProgram (
    void )
```

Starts the user program, if one is present. In this case this function does not return.

Returns

none.

7.347.2.5 CpuUserProgramStartHook()

```
blt_bool CpuUserProgramStartHook (
    void )
```

Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.

Returns

BLT_TRUE if it is okay to start the user program, BLT_FALSE to keep keep the bootloader active.

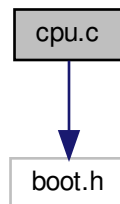
Referenced by CpuStartUserProgram().

7.348 cpu.c File Reference

Bootloader cpu module source file.

```
#include "boot.h"
```

Include dependency graph for HCS12/cpu.c:



Macros

- `#define CPU_USER_PROGRAM_STARTADDR_PTR (NvmGetUserProgBaseAddress() - 2)`
Start address of the user program. This is the address of the reset vector in the user program's vector table.

Functions

- `blt_bool CpuUserProgramStartHook (void)`
Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.
- `void CpuInit (void)`
Initializes the CPU module.
- `void CpuStartUserProgram (void)`
Starts the user program, if one is present. In this case this function does not return.
- `void CpuMemCopy (blt_addr dest, blt_addr src, blt_int16u len)`
Copies data from the source to the destination address.
- `void CpuMemSet (blt_addr dest, blt_int8u value, blt_int16u len)`
Sets the bytes at the destination address to the specified value.

7.348.1 Detailed Description

Bootloader cpu module source file.

7.348.2 Macro Definition Documentation

7.348.2.1 CPU_USER_PROGRAM_STARTADDR_PTR

```
#define CPU_USER_PROGRAM_STARTADDR_PTR (NvmGetUserProgBaseAddress() - 2)
```

Start address of the user program. This is the address of the reset vector in the user program's vector table.

Attention

This value must be updated if the memory reserved for the bootloader changes.

Referenced by CpuStartUserProgram().

7.348.3 Function Documentation

7.348.3.1 CpuInit()

```
void CpuInit (  
    void )
```

Initializes the CPU module.

Returns

none.

7.348.3.2 CpuMemCopy()

```
void CpuMemCopy (  
    blt_addr dest,  
    blt_addr src,  
    blt_int16u len )
```

Copies data from the source to the destination address.

Parameters

<i>dest</i>	Destination address for the data.
<i>src</i>	Source address of the data.
<i>len</i>	length of the data in bytes.

Returns

none.

7.348.3.3 CpuMemSet()

```
void CpuMemSet (
    blt_addr dest,
    blt_int8u value,
    blt_int16u len )
```

Sets the bytes at the destination address to the specified value.

Parameters

<i>dest</i>	Destination address for the data.
<i>value</i>	Value to write.
<i>len</i>	Number of bytes to write.

Returns

none.

7.348.3.4 CpuStartUserProgram()

```
void CpuStartUserProgram (
    void )
```

Starts the user program, if one is present. In this case this function does not return.

Returns

none.

7.348.3.5 CpuUserProgramStartHook()

```
blt_bool CpuUserProgramStartHook (
    void )
```

Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.

Returns

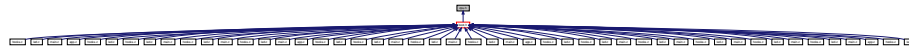
BLT_TRUE if it is okay to start the user program, BLT_FALSE to keep keep the bootloader active.

Referenced by CpuStartUserProgram().

7.349 cpu.h File Reference

Bootloader cpu module header file.

This graph shows which files directly or indirectly include this file:



Functions

- void [CpuInit](#) (void)
Initializes the CPU module.
- void [CpuStartUserProgram](#) (void)
Starts the user program, if one is present. In this case this function does not return.
- void [CpuMemCopy](#) (blt_addr dest, blt_addr src, blt_int16u len)
Copies data from the source to the destination address.
- void [CpuMemSet](#) (blt_addr dest, blt_int8u value, blt_int16u len)
Sets the bytes at the destination address to the specified value.
- void [CpuIrqDisable](#) (void)
Disable global interrupts.
- void [CpuIrqEnable](#) (void)
Enable global interrupts.

7.349.1 Detailed Description

Bootloader cpu module header file.

7.349.2 Function Documentation

7.349.2.1 CpuInit()

```
void CpuInit (
    void )
```

Initializes the CPU module.

Returns

none.

Referenced by [BootInit\(\)](#).

7.349.2.2 CpuIrqDisable()

```
void CpuIrqDisable (
    void )
```

Disable global interrupts.

Returns

none.

Referenced by CpuInit().

7.349.2.3 CpuIrqEnable()

```
void CpuIrqEnable (
    void )
```

Enable global interrupts.

Returns

none.

Referenced by CpuStartUserProgram().

7.349.2.4 CpuMemCopy()

```
void CpuMemCopy (
    blt_addr dest,
    blt_addr src,
    blt_int16u len )
```

Copies data from the source to the destination address.

Parameters

<i>dest</i>	Destination address for the data.
<i>src</i>	Source address of the data.
<i>len</i>	length of the data in bytes.

Returns

none.

Referenced by CpuStartUserProgram(), FlashInitBlock(), UsbReceivePacket(), XcpCmdGetSeed(), XcpCmdShortUpload(), XcpCmdUnlock(), and XcpCmdUpload().

7.349.2.5 CpuMemSet()

```
void CpuMemSet (
    blt_addr dest,
    blt_int8u value,
    blt_int16u len )
```

Sets the bytes at the destination address to the specified value.

Parameters

<i>dest</i>	Destination address for the data.
<i>value</i>	Value to write.
<i>len</i>	Number of bytes to write.

Returns

none.

Referenced by XcpCmdShortUpload(), and XcpCmdUpload().

7.349.2.6 CpuStartUserProgram()

```
void CpuStartUserProgram (
    void )
```

Starts the user program, if one is present. In this case this function does not return.

Returns

none.

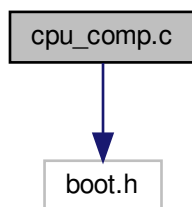
Referenced by BackDoorCheck(), BackDoorInit(), FileTask(), and XcpCmdProgramReset().

7.350 cpu_comp.c File Reference

Bootloader cpu module source file.

```
#include "boot.h"
```

Include dependency graph for _template/GCC/cpu_comp.c:



Functions

- void [CpuIrqDisable](#) (void)
Disable global interrupts.
- void [CpuIrqEnable](#) (void)
Enable global interrupts.

7.350.1 Detailed Description

Bootloader cpu module source file.

7.350.2 Function Documentation

7.350.2.1 CpuIrqDisable()

```
void CpuIrqDisable (  
    void )
```

Disable global interrupts.

Returns

none.

Referenced by CpuInit().

7.350.2.2 CpuIrqEnable()

```
void CpuIrqEnable (
    void )
```

Enable global interrupts.

Returns

none.

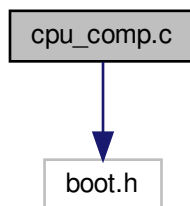
Referenced by CpuStartUserProgram().

7.351 cpu_comp.c File Reference

Bootloader cpu module source file.

```
#include "boot.h"
```

Include dependency graph for ARMCM0_S32K11/GCC/cpu_comp.c:



Functions

- void [CpuIrqDisable](#) (void)
Disable global interrupts.
- void [CpuIrqEnable](#) (void)
Enable global interrupts.

7.351.1 Detailed Description

Bootloader cpu module source file.

7.351.2 Function Documentation

7.351.2.1 CpuIrqDisable()

```
void CpuIrqDisable (
    void )
```

Disable global interrupts.

Returns

none.

7.351.2.2 CpuIrqEnable()

```
void CpuIrqEnable (
    void )
```

Enable global interrupts.

Returns

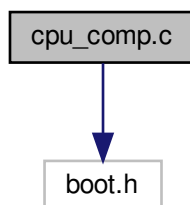
none.

7.352 cpu_comp.c File Reference

Bootloader cpu module source file.

```
#include "boot.h"
```

Include dependency graph for ARMCM0_S32K11/IAR/cpu_comp.c:



Functions

- void [CpuIrqDisable](#) (void)
Disable global interrupts.
- void [CpuIrqEnable](#) (void)
Enable global interrupts.

7.352.1 Detailed Description

Bootloader cpu module source file.

7.352.2 Function Documentation

7.352.2.1 CpuIrqDisable()

```
void CpuIrqDisable (
    void )
```

Disable global interrupts.

Returns

none.

7.352.2.2 CpuIrqEnable()

```
void CpuIrqEnable (
    void )
```

Enable global interrupts.

Returns

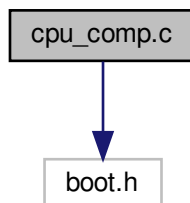
none.

7.353 cpu_comp.c File Reference

Bootloader cpu module source file.

```
#include "boot.h"
```

Include dependency graph for ARMCM0_STM32F0/GCC/cpu_comp.c:



Functions

- void [CpulrqDisable](#) (void)
Disable global interrupts.
- void [CpulrqEnable](#) (void)
Enable global interrupts.

7.353.1 Detailed Description

Bootloader cpu module source file.

7.353.2 Function Documentation

7.353.2.1 CpulrqDisable()

```
void CpuIrqDisable (  
    void )
```

Disable global interrupts.

Returns

none.

7.353.2.2 CpulrqEnable()

```
void CpuIrqEnable (  
    void )
```

Enable global interrupts.

Returns

none.

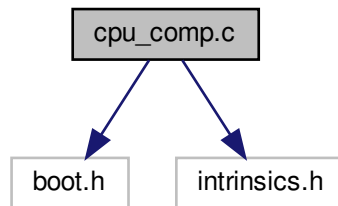
7.354 cpu_comp.c File Reference

Bootloader cpu module source file.

```
#include "boot.h"
```

```
#include <intrinsics.h>
```

Include dependency graph for ARMCM0_STM32F0/IAR/cpu_comp.c:



Functions

- void [CpulrqDisable](#) (void)
Disable global interrupts.
- void [CpulrqEnable](#) (void)
Enable global interrupts.

7.354.1 Detailed Description

Bootloader cpu module source file.

7.354.2 Function Documentation

7.354.2.1 CpulrqDisable()

```
void CpuIrqDisable (  
    void )
```

Disable global interrupts.

Returns

none.

7.354.2.2 CpuIrqEnable()

```
void CpuIrqEnable (
    void )
```

Enable global interrupts.

Returns

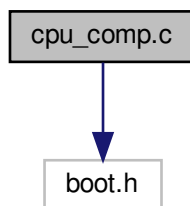
none.

7.355 cpu_comp.c File Reference

Bootloader cpu module source file.

```
#include "boot.h"
```

Include dependency graph for ARMCM0_STM32F0/Keil/cpu_comp.c:



Functions

- void [CpuIrqDisable](#) (void)
Disable global interrupts.
- void [CpuIrqEnable](#) (void)
Enable global interrupts.

7.355.1 Detailed Description

Bootloader cpu module source file.

7.355.2 Function Documentation

7.355.2.1 CpuIrqDisable()

```
void CpuIrqDisable (
    void )
```

Disable global interrupts.

Returns

none.

7.355.2.2 CpuIrqEnable()

```
void CpuIrqEnable (
    void )
```

Enable global interrupts.

Returns

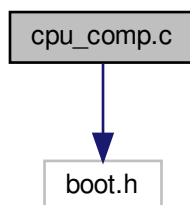
none.

7.356 cpu_comp.c File Reference

Bootloader cpu module source file.

```
#include "boot.h"
```

Include dependency graph for ARMCM0_STM32G0/GCC/cpu_comp.c:



Functions

- void [CpuIrqDisable](#) (void)
Disable global interrupts.
- void [CpuIrqEnable](#) (void)
Enable global interrupts.

7.356.1 Detailed Description

Bootloader cpu module source file.

7.356.2 Function Documentation

7.356.2.1 CpuIrqDisable()

```
void CpuIrqDisable (  
    void )
```

Disable global interrupts.

Returns

none.

7.356.2.2 CpuIrqEnable()

```
void CpuIrqEnable (  
    void )
```

Enable global interrupts.

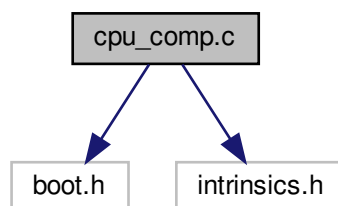
Returns

none.

7.357 cpu_comp.c File Reference

Bootloader cpu module source file.

```
#include "boot.h"  
#include <intrinsics.h>  
Include dependency graph for ARMCM0_STM32G0/IAR/cpu_comp.c:
```



Functions

- void [CpulrQDisable](#) (void)
Disable global interrupts.
- void [CpulrQEnable](#) (void)
Enable global interrupts.

7.357.1 Detailed Description

Bootloader cpu module source file.

7.357.2 Function Documentation

7.357.2.1 CpulrQDisable()

```
void CpuIrqDisable (  
    void )
```

Disable global interrupts.

Returns

none.

7.357.2.2 CpulrQEnable()

```
void CpuIrqEnable (  
    void )
```

Enable global interrupts.

Returns

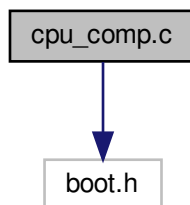
none.

7.358 cpu_comp.c File Reference

Bootloader cpu module source file.

```
#include "boot.h"
```

Include dependency graph for ARMC00_STM32G0/Keil/cpu_comp.c:



Functions

- void [CpulrQDisable](#) (void)
Disable global interrupts.
- void [CpulrQEnable](#) (void)
Enable global interrupts.

7.358.1 Detailed Description

Bootloader cpu module source file.

7.358.2 Function Documentation

7.358.2.1 CpulrQDisable()

```
void CpuIrQDisable (  
    void )
```

Disable global interrupts.

Returns

none.

7.358.2.2 CpuIrqEnable()

```
void CpuIrqEnable (  
    void )
```

Enable global interrupts.

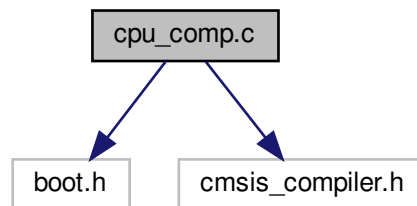
Returns

none.

7.359 cpu_comp.c File Reference

Bootloader cpu module source file.

```
#include "boot.h"  
#include "cmsis_compiler.h"  
Include dependency graph for ARMCM0_XMC1/GCC/cpu_comp.c:
```



Functions

- void [CpuIrqDisable](#) (void)
Disable global interrupts.
- void [CpuIrqEnable](#) (void)
Enable global interrupts.

7.359.1 Detailed Description

Bootloader cpu module source file.

7.359.2 Function Documentation

7.359.2.1 CpuIrqDisable()

```
void CpuIrqDisable (  
    void )
```

Disable global interrupts.

Returns

none.

7.359.2.2 CpuIrqEnable()

```
void CpuIrqEnable (  
    void )
```

Enable global interrupts.

Returns

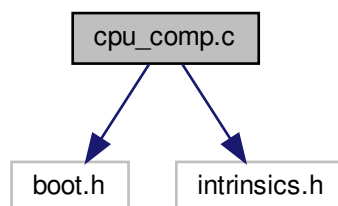
none.

7.360 cpu_comp.c File Reference

Bootloader cpu module source file.

```
#include "boot.h"  
#include <intrinsics.h>
```

Include dependency graph for ARMCM0_XMC1/IAR/cpu_comp.c:



Functions

- void [CpuIrqDisable](#) (void)
Disable global interrupts.
- void [CpuIrqEnable](#) (void)
Enable global interrupts.

7.360.1 Detailed Description

Bootloader cpu module source file.

7.360.2 Function Documentation

7.360.2.1 CpuIrqDisable()

```
void CpuIrqDisable (
    void )
```

Disable global interrupts.

Returns

none.

7.360.2.2 CpuIrqEnable()

```
void CpuIrqEnable (
    void )
```

Enable global interrupts.

Returns

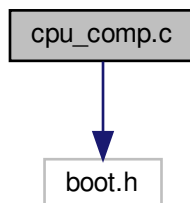
none.

7.361 cpu_comp.c File Reference

Bootloader cpu module source file.

```
#include "boot.h"
```

Include dependency graph for ARMCM33_STM32L5/GCC/cpu_comp.c:



Functions

- void [CpulrqDisable](#) (void)
Disable global interrupts.
- void [CpulrqEnable](#) (void)
Enable global interrupts.

7.361.1 Detailed Description

Bootloader cpu module source file.

7.361.2 Function Documentation

7.361.2.1 CpulrqDisable()

```
void CpuIrqDisable (  
    void )
```

Disable global interrupts.

Returns

none.

7.361.2.2 CpulrqEnable()

```
void CpuIrqEnable (  
    void )
```

Enable global interrupts.

Returns

none.

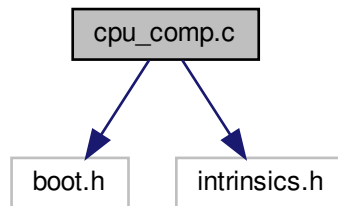
7.362 cpu_comp.c File Reference

Bootloader cpu module source file.

```
#include "boot.h"
```

```
#include <intrinsics.h>
```

Include dependency graph for ARMCM33_STM32L5/IAR/cpu_comp.c:



Functions

- void [CpulrqDisable](#) (void)
Disable global interrupts.
- void [CpulrqEnable](#) (void)
Enable global interrupts.

7.362.1 Detailed Description

Bootloader cpu module source file.

7.362.2 Function Documentation

7.362.2.1 CpulrqDisable()

```
void CpuIrqDisable (  
    void )
```

Disable global interrupts.

Returns

none.

7.362.2.2 CpuIrqEnable()

```
void CpuIrqEnable (
    void )
```

Enable global interrupts.

Returns

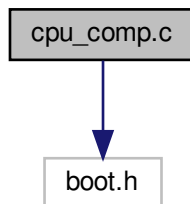
none.

7.363 cpu_comp.c File Reference

Bootloader cpu module source file.

```
#include "boot.h"
```

Include dependency graph for ARMCM33_STM32L5/Keil/cpu_comp.c:



Functions

- void [CpuIrqDisable](#) (void)
Disable global interrupts.
- void [CpuIrqEnable](#) (void)
Enable global interrupts.

7.363.1 Detailed Description

Bootloader cpu module source file.

7.363.2 Function Documentation

7.363.2.1 CpuIrqDisable()

```
void CpuIrqDisable (
    void )
```

Disable global interrupts.

Returns

none.

7.363.2.2 CpuIrqEnable()

```
void CpuIrqEnable (
    void )
```

Enable global interrupts.

Returns

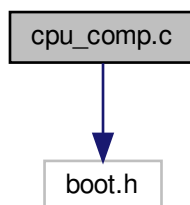
none.

7.364 cpu_comp.c File Reference

Bootloader cpu module source file.

```
#include "boot.h"
```

Include dependency graph for ARMC32_EFM32/GCC/cpu_comp.c:



Functions

- void [CpuIrqDisable](#) (void)
Disable global interrupts.
- void [CpuIrqEnable](#) (void)
Enable global interrupts.

7.364.1 Detailed Description

Bootloader cpu module source file.

7.364.2 Function Documentation

7.364.2.1 CpuIrqDisable()

```
void CpuIrqDisable (  
    void )
```

Disable global interrupts.

Returns

none.

7.364.2.2 CpuIrqEnable()

```
void CpuIrqEnable (  
    void )
```

Enable global interrupts.

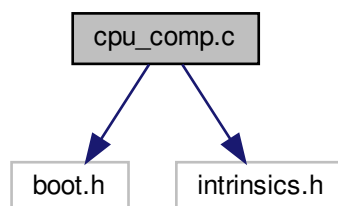
Returns

none.

7.365 cpu_comp.c File Reference

Bootloader cpu module source file.

```
#include "boot.h"  
#include <intrinsics.h>  
Include dependency graph for ARMCM3_EFM32/IAR/cpu_comp.c:
```



Functions

- void [CpulrQDisable](#) (void)
Disable global interrupts.
- void [CpulrQEnable](#) (void)
Enable global interrupts.

7.365.1 Detailed Description

Bootloader cpu module source file.

7.365.2 Function Documentation

7.365.2.1 CpulrQDisable()

```
void CpuIrqDisable (  
    void )
```

Disable global interrupts.

Returns

none.

7.365.2.2 CpulrQEnable()

```
void CpuIrqEnable (  
    void )
```

Enable global interrupts.

Returns

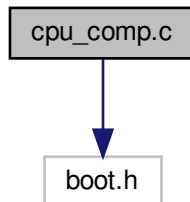
none.

7.366 cpu_comp.c File Reference

Bootloader cpu module source file.

```
#include "boot.h"
```

Include dependency graph for ARMC3_LM3S/GCC/cpu_comp.c:



Functions

- void [CpulrQDisable](#) (void)
Disable global interrupts.
- void [CpulrQEnable](#) (void)
Enable global interrupts.

7.366.1 Detailed Description

Bootloader cpu module source file.

7.366.2 Function Documentation

7.366.2.1 CpulrQDisable()

```
void CpuIrQDisable (  
    void )
```

Disable global interrupts.

Returns

none.

7.366.2.2 CpuIrqEnable()

```
void CpuIrqEnable (  
    void )
```

Enable global interrupts.

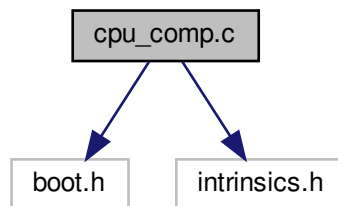
Returns

none.

7.367 cpu_comp.c File Reference

Bootloader cpu module source file.

```
#include "boot.h"  
#include <intrinsics.h>  
Include dependency graph for ARMCM3_LM3S/IAR/cpu_comp.c:
```



Functions

- void [CpuIrqDisable](#) (void)
Disable global interrupts.
- void [CpuIrqEnable](#) (void)
Enable global interrupts.

7.367.1 Detailed Description

Bootloader cpu module source file.

7.367.2 Function Documentation

7.367.2.1 CpuIrqDisable()

```
void CpuIrqDisable (
    void )
```

Disable global interrupts.

Returns

none.

7.367.2.2 CpuIrqEnable()

```
void CpuIrqEnable (
    void )
```

Enable global interrupts.

Returns

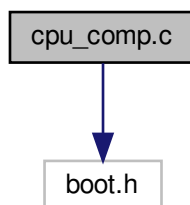
none.

7.368 cpu_comp.c File Reference

Bootloader cpu module source file.

```
#include "boot.h"
```

Include dependency graph for ARMC3_STM32F1/GCC/cpu_comp.c:



Functions

- void [CpuIrqDisable](#) (void)
Disable global interrupts.
- void [CpuIrqEnable](#) (void)
Enable global interrupts.

7.368.1 Detailed Description

Bootloader cpu module source file.

7.368.2 Function Documentation

7.368.2.1 CpuIrqDisable()

```
void CpuIrqDisable (  
    void )
```

Disable global interrupts.

Returns

none.

7.368.2.2 CpuIrqEnable()

```
void CpuIrqEnable (  
    void )
```

Enable global interrupts.

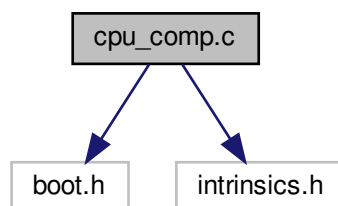
Returns

none.

7.369 cpu_comp.c File Reference

Bootloader cpu module source file.

```
#include "boot.h"  
#include <intrinsics.h>  
Include dependency graph for ARMCM3_STM32F1/IAR/cpu_comp.c:
```



Functions

- void [CpulrqDisable](#) (void)
Disable global interrupts.
- void [CpulrqEnable](#) (void)
Enable global interrupts.

7.369.1 Detailed Description

Bootloader cpu module source file.

7.369.2 Function Documentation

7.369.2.1 CpulrqDisable()

```
void CpuIrqDisable (  
    void )
```

Disable global interrupts.

Returns

none.

7.369.2.2 CpulrqEnable()

```
void CpuIrqEnable (  
    void )
```

Enable global interrupts.

Returns

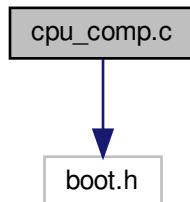
none.

7.370 cpu_comp.c File Reference

Bootloader cpu module source file.

```
#include "boot.h"
```

Include dependency graph for ARMC3_STM32F1/Keil/cpu_comp.c:



Functions

- void [CpulrQDisable](#) (void)
Disable global interrupts.
- void [CpulrQEnable](#) (void)
Enable global interrupts.

7.370.1 Detailed Description

Bootloader cpu module source file.

7.370.2 Function Documentation

7.370.2.1 CpulrQDisable()

```
void CpuIrqDisable (  
    void )
```

Disable global interrupts.

Returns

none.

7.370.2.2 CpuIrqEnable()

```
void CpuIrqEnable (
    void )
```

Enable global interrupts.

Returns

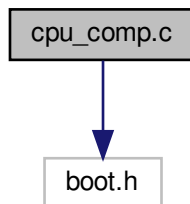
none.

7.371 cpu_comp.c File Reference

Bootloader cpu module source file.

```
#include "boot.h"
```

Include dependency graph for ARMCM3_STM32F2/GCC/cpu_comp.c:



Functions

- void [CpuIrqDisable](#) (void)
Disable global interrupts.
- void [CpuIrqEnable](#) (void)
Enable global interrupts.

7.371.1 Detailed Description

Bootloader cpu module source file.

7.371.2 Function Documentation

7.371.2.1 CpuIrqDisable()

```
void CpuIrqDisable (
    void )
```

Disable global interrupts.

Returns

none.

7.371.2.2 CpuIrqEnable()

```
void CpuIrqEnable (
    void )
```

Enable global interrupts.

Returns

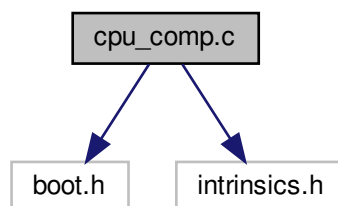
none.

7.372 cpu_comp.c File Reference

Bootloader cpu module source file.

```
#include "boot.h"
#include <intrinsics.h>
```

Include dependency graph for ARMC32F2/IAR/cpu_comp.c:



Functions

- void [CpuIrqDisable](#) (void)
Disable global interrupts.
- void [CpuIrqEnable](#) (void)
Enable global interrupts.

7.372.1 Detailed Description

Bootloader cpu module source file.

7.372.2 Function Documentation

7.372.2.1 CpuIrqDisable()

```
void CpuIrqDisable (  
    void )
```

Disable global interrupts.

Returns

none.

7.372.2.2 CpuIrqEnable()

```
void CpuIrqEnable (  
    void )
```

Enable global interrupts.

Returns

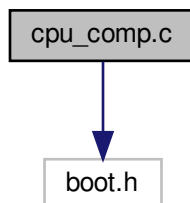
none.

7.373 cpu_comp.c File Reference

Bootloader cpu module source file.

```
#include "boot.h"
```

Include dependency graph for ARMCM3_STM32F2/Keil/cpu_comp.c:



Functions

- void [CpulrQDisable](#) (void)
Disable global interrupts.
- void [CpulrQEnable](#) (void)
Enable global interrupts.

7.373.1 Detailed Description

Bootloader cpu module source file.

7.373.2 Function Documentation

7.373.2.1 CpulrQDisable()

```
void CpuIrqDisable (  
    void )
```

Disable global interrupts.

Returns

none.

7.373.2.2 CpulrQEnable()

```
void CpuIrqEnable (  
    void )
```

Enable global interrupts.

Returns

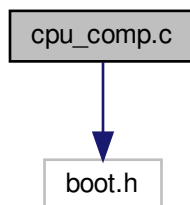
none.

7.374 cpu_comp.c File Reference

Bootloader cpu module source file.

```
#include "boot.h"
```

Include dependency graph for ARMC4_S32K14/GCC/cpu_comp.c:



Functions

- void [CpulrQDisable](#) (void)
Disable global interrupts.
- void [CpulrQEnable](#) (void)
Enable global interrupts.

7.374.1 Detailed Description

Bootloader cpu module source file.

7.374.2 Function Documentation

7.374.2.1 CpulrQDisable()

```
void CpuIrqDisable (  
    void )
```

Disable global interrupts.

Returns

none.

7.374.2.2 CpuIrqEnable()

```
void CpuIrqEnable (
    void )
```

Enable global interrupts.

Returns

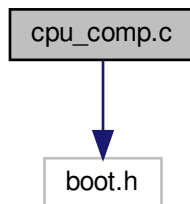
none.

7.375 cpu_comp.c File Reference

Bootloader cpu module source file.

```
#include "boot.h"
```

Include dependency graph for ARMCM4_S32K14/IAR/cpu_comp.c:



Functions

- void [CpuIrqDisable](#) (void)
Disable global interrupts.
- void [CpuIrqEnable](#) (void)
Enable global interrupts.

7.375.1 Detailed Description

Bootloader cpu module source file.

7.375.2 Function Documentation

7.375.2.1 CpuIrqDisable()

```
void CpuIrqDisable (
    void )
```

Disable global interrupts.

Returns

none.

7.375.2.2 CpuIrqEnable()

```
void CpuIrqEnable (
    void )
```

Enable global interrupts.

Returns

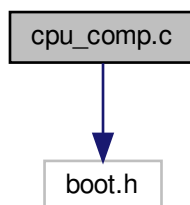
none.

7.376 cpu_comp.c File Reference

Bootloader cpu module source file.

```
#include "boot.h"
```

Include dependency graph for ARMCM4_STM32F3/GCC/cpu_comp.c:



Functions

- void [CpuIrqDisable](#) (void)
Disable global interrupts.
- void [CpuIrqEnable](#) (void)
Enable global interrupts.

7.376.1 Detailed Description

Bootloader cpu module source file.

7.376.2 Function Documentation

7.376.2.1 CpuIrqDisable()

```
void CpuIrqDisable (  
    void )
```

Disable global interrupts.

Returns

none.

7.376.2.2 CpuIrqEnable()

```
void CpuIrqEnable (  
    void )
```

Enable global interrupts.

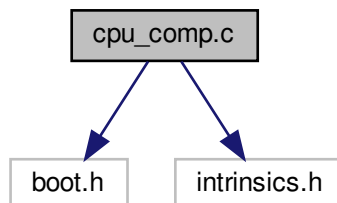
Returns

none.

7.377 cpu_comp.c File Reference

Bootloader cpu module source file.

```
#include "boot.h"  
#include <intrinsics.h>  
Include dependency graph for ARMCM4_STM32F3/IAR/cpu_comp.c:
```



Functions

- void [CpulrqDisable](#) (void)
Disable global interrupts.
- void [CpulrqEnable](#) (void)
Enable global interrupts.

7.377.1 Detailed Description

Bootloader cpu module source file.

7.377.2 Function Documentation

7.377.2.1 CpulrqDisable()

```
void CpuIrqDisable (  
    void )
```

Disable global interrupts.

Returns

none.

7.377.2.2 CpulrqEnable()

```
void CpuIrqEnable (  
    void )
```

Enable global interrupts.

Returns

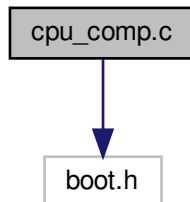
none.

7.378 cpu_comp.c File Reference

Bootloader cpu module source file.

```
#include "boot.h"
```

Include dependency graph for ARMC4_STM32F3/Keil/cpu_comp.c:



Functions

- void [CpulrQDisable](#) (void)
Disable global interrupts.
- void [CpulrQEnable](#) (void)
Enable global interrupts.

7.378.1 Detailed Description

Bootloader cpu module source file.

7.378.2 Function Documentation

7.378.2.1 CpulrQDisable()

```
void CpuIrqDisable (  
    void )
```

Disable global interrupts.

Returns

none.

7.378.2.2 CpuIrqEnable()

```
void CpuIrqEnable (
    void )
```

Enable global interrupts.

Returns

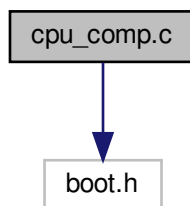
none.

7.379 cpu_comp.c File Reference

Bootloader cpu module source file.

```
#include "boot.h"
```

Include dependency graph for ARMCM4_STM32F4/GCC/cpu_comp.c:



Functions

- void [CpuIrqDisable](#) (void)
Disable global interrupts.
- void [CpuIrqEnable](#) (void)
Enable global interrupts.

7.379.1 Detailed Description

Bootloader cpu module source file.

7.379.2 Function Documentation

7.379.2.1 CpuIrqDisable()

```
void CpuIrqDisable (  
    void )
```

Disable global interrupts.

Returns

none.

7.379.2.2 CpuIrqEnable()

```
void CpuIrqEnable (  
    void )
```

Enable global interrupts.

Returns

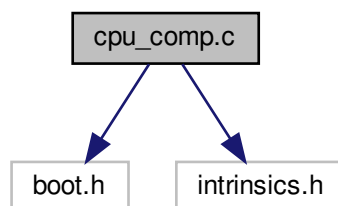
none.

7.380 cpu_comp.c File Reference

Bootloader cpu module source file.

```
#include "boot.h"  
#include <intrinsics.h>
```

Include dependency graph for ARMCM4_STM32F4/IAR/cpu_comp.c:



Functions

- void [CpuIrqDisable](#) (void)
Disable global interrupts.
- void [CpuIrqEnable](#) (void)
Enable global interrupts.

7.380.1 Detailed Description

Bootloader cpu module source file.

7.380.2 Function Documentation

7.380.2.1 CpuIrqDisable()

```
void CpuIrqDisable (  
    void )
```

Disable global interrupts.

Returns

none.

7.380.2.2 CpuIrqEnable()

```
void CpuIrqEnable (  
    void )
```

Enable global interrupts.

Returns

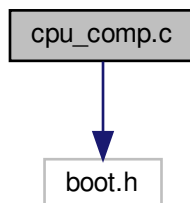
none.

7.381 cpu_comp.c File Reference

Bootloader cpu module source file.

```
#include "boot.h"
```

Include dependency graph for ARMCM4_STM32F4/Keil/cpu_comp.c:



Functions

- void [CpulrQDisable](#) (void)
Disable global interrupts.
- void [CpulrQEnable](#) (void)
Enable global interrupts.

7.381.1 Detailed Description

Bootloader cpu module source file.

7.381.2 Function Documentation

7.381.2.1 CpulrQDisable()

```
void CpuIrqDisable (  
    void )
```

Disable global interrupts.

Returns

none.

7.381.2.2 CpulrQEnable()

```
void CpuIrqEnable (  
    void )
```

Enable global interrupts.

Returns

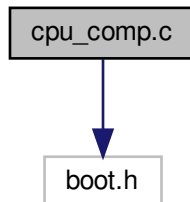
none.

7.382 cpu_comp.c File Reference

Bootloader cpu module source file.

```
#include "boot.h"
```

Include dependency graph for ARMC4_STM32L4/GCC/cpu_comp.c:



Functions

- void [CpulrQDisable](#) (void)
Disable global interrupts.
- void [CpulrQEnable](#) (void)
Enable global interrupts.

7.382.1 Detailed Description

Bootloader cpu module source file.

7.382.2 Function Documentation

7.382.2.1 CpulrQDisable()

```
void CpuIrqDisable (  
    void )
```

Disable global interrupts.

Returns

none.

7.382.2.2 CpuIrqEnable()

```
void CpuIrqEnable (  
    void )
```

Enable global interrupts.

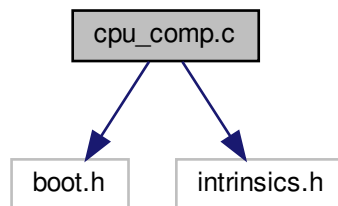
Returns

none.

7.383 cpu_comp.c File Reference

Bootloader cpu module source file.

```
#include "boot.h"  
#include <intrinsics.h>  
Include dependency graph for ARMCM4_STM32L4/IAR/cpu_comp.c:
```



Functions

- void [CpuIrqDisable](#) (void)
Disable global interrupts.
- void [CpuIrqEnable](#) (void)
Enable global interrupts.

7.383.1 Detailed Description

Bootloader cpu module source file.

7.383.2 Function Documentation

7.383.2.1 CpuIrqDisable()

```
void CpuIrqDisable (
    void )
```

Disable global interrupts.

Returns

none.

7.383.2.2 CpuIrqEnable()

```
void CpuIrqEnable (
    void )
```

Enable global interrupts.

Returns

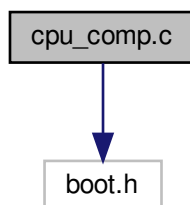
none.

7.384 cpu_comp.c File Reference

Bootloader cpu module source file.

```
#include "boot.h"
```

Include dependency graph for ARMCM4_STM32L4/Keil/cpu_comp.c:



Functions

- void [CpuIrqDisable](#) (void)
Disable global interrupts.
- void [CpuIrqEnable](#) (void)
Enable global interrupts.

7.384.1 Detailed Description

Bootloader cpu module source file.

7.384.2 Function Documentation

7.384.2.1 CpuIrqDisable()

```
void CpuIrqDisable (  
    void )
```

Disable global interrupts.

Returns

none.

7.384.2.2 CpuIrqEnable()

```
void CpuIrqEnable (  
    void )
```

Enable global interrupts.

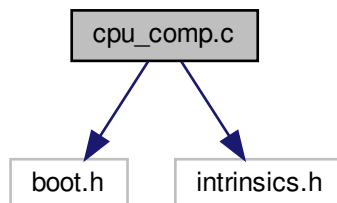
Returns

none.

7.385 cpu_comp.c File Reference

Bootloader cpu module source file.

```
#include "boot.h"  
#include <intrinsics.h>  
Include dependency graph for ARMCM4_TM4C/IAR/cpu_comp.c:
```



Functions

- void [CpulrqDisable](#) (void)
Disable global interrupts.
- void [CpulrqEnable](#) (void)
Enable global interrupts.

7.385.1 Detailed Description

Bootloader cpu module source file.

7.385.2 Function Documentation

7.385.2.1 CpulrqDisable()

```
void CpuIrqDisable (  
    void )
```

Disable global interrupts.

Returns

none.

7.385.2.2 CpulrqEnable()

```
void CpuIrqEnable (  
    void )
```

Enable global interrupts.

Returns

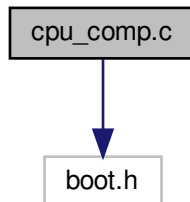
none.

7.386 cpu_comp.c File Reference

Bootloader cpu module source file.

```
#include "boot.h"
```

Include dependency graph for ARMCM4_XMC4/GCC/cpu_comp.c:



Functions

- void [CpulrQDisable](#) (void)
Disable global interrupts.
- void [CpulrQEnable](#) (void)
Enable global interrupts.

7.386.1 Detailed Description

Bootloader cpu module source file.

7.386.2 Function Documentation

7.386.2.1 CpulrQDisable()

```
void CpuIrqDisable (  
    void )
```

Disable global interrupts.

Returns

none.

7.386.2.2 CpuIrqEnable()

```
void CpuIrqEnable (
    void )
```

Enable global interrupts.

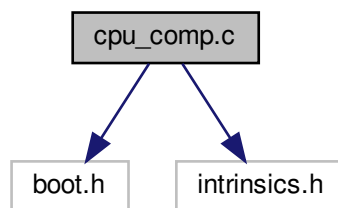
Returns

none.

7.387 cpu_comp.c File Reference

Bootloader cpu module source file.

```
#include "boot.h"
#include <intrinsics.h>
Include dependency graph for ARMCM4_XMC4/IAR/cpu_comp.c:
```



Functions

- void [CpuIrqDisable](#) (void)
Disable global interrupts.
- void [CpuIrqEnable](#) (void)
Enable global interrupts.

7.387.1 Detailed Description

Bootloader cpu module source file.

7.387.2 Function Documentation

7.387.2.1 CpuIrqDisable()

```
void CpuIrqDisable (
    void )
```

Disable global interrupts.

Returns

none.

7.387.2.2 CpuIrqEnable()

```
void CpuIrqEnable (
    void )
```

Enable global interrupts.

Returns

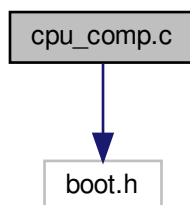
none.

7.388 cpu_comp.c File Reference

Bootloader cpu module source file.

```
#include "boot.h"
```

Include dependency graph for ARMC7_STM32F7/GCC/cpu_comp.c:



Functions

- void [CpuIrqDisable](#) (void)
Disable global interrupts.
- void [CpuIrqEnable](#) (void)
Enable global interrupts.

7.388.1 Detailed Description

Bootloader cpu module source file.

7.388.2 Function Documentation

7.388.2.1 CpuIrqDisable()

```
void CpuIrqDisable (  
    void )
```

Disable global interrupts.

Returns

none.

7.388.2.2 CpuIrqEnable()

```
void CpuIrqEnable (  
    void )
```

Enable global interrupts.

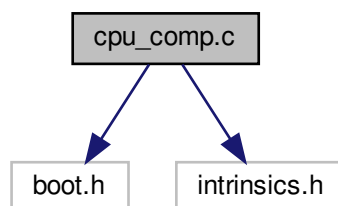
Returns

none.

7.389 cpu_comp.c File Reference

Bootloader cpu module source file.

```
#include "boot.h"  
#include <intrinsics.h>  
Include dependency graph for ARMCM7_STM32F7/IAR/cpu_comp.c:
```



Functions

- void [CpulrQDisable](#) (void)
Disable global interrupts.
- void [CpulrQEnable](#) (void)
Enable global interrupts.

7.389.1 Detailed Description

Bootloader cpu module source file.

7.389.2 Function Documentation

7.389.2.1 CpulrQDisable()

```
void CpuIrqDisable (  
    void )
```

Disable global interrupts.

Returns

none.

7.389.2.2 CpulrQEnable()

```
void CpuIrqEnable (  
    void )
```

Enable global interrupts.

Returns

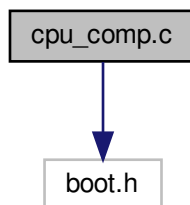
none.

7.390 cpu_comp.c File Reference

Bootloader cpu module source file.

```
#include "boot.h"
```

Include dependency graph for ARMC7_STM32F7/Keil/cpu_comp.c:



Functions

- void [CpulrQDisable](#) (void)
Disable global interrupts.
- void [CpulrQEnable](#) (void)
Enable global interrupts.

7.390.1 Detailed Description

Bootloader cpu module source file.

7.390.2 Function Documentation

7.390.2.1 CpulrQDisable()

```
void CpuIrqDisable (  
    void )
```

Disable global interrupts.

Returns

none.

7.390.2.2 CpuIrqEnable()

```
void CpuIrqEnable (
    void )
```

Enable global interrupts.

Returns

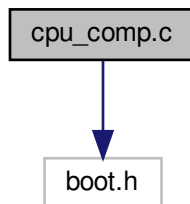
none.

7.391 cpu_comp.c File Reference

Bootloader cpu module source file.

```
#include "boot.h"
```

Include dependency graph for ARMCM7_STM32H7/GCC/cpu_comp.c:



Functions

- void [CpuIrqDisable](#) (void)
Disable global interrupts.
- void [CpuIrqEnable](#) (void)
Enable global interrupts.

7.391.1 Detailed Description

Bootloader cpu module source file.

7.391.2 Function Documentation

7.391.2.1 CpuIrqDisable()

```
void CpuIrqDisable (  
    void )
```

Disable global interrupts.

Returns

none.

7.391.2.2 CpuIrqEnable()

```
void CpuIrqEnable (  
    void )
```

Enable global interrupts.

Returns

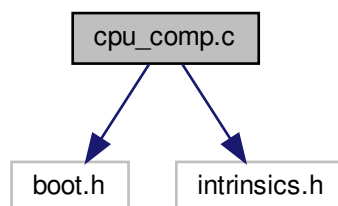
none.

7.392 cpu_comp.c File Reference

Bootloader cpu module source file.

```
#include "boot.h"  
#include <intrinsics.h>
```

Include dependency graph for ARMCM7_STM32H7/IAR/cpu_comp.c:



Functions

- void [CpuIrqDisable](#) (void)
Disable global interrupts.
- void [CpuIrqEnable](#) (void)
Enable global interrupts.

7.392.1 Detailed Description

Bootloader cpu module source file.

7.392.2 Function Documentation

7.392.2.1 CpuIrqDisable()

```
void CpuIrqDisable (
    void )
```

Disable global interrupts.

Returns

none.

7.392.2.2 CpuIrqEnable()

```
void CpuIrqEnable (
    void )
```

Enable global interrupts.

Returns

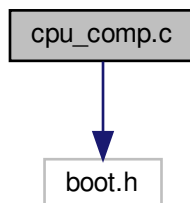
none.

7.393 cpu_comp.c File Reference

Bootloader cpu module source file.

```
#include "boot.h"
```

Include dependency graph for ARMCM7_STM32H7/Keil/cpu_comp.c:



Functions

- void [CpulrqDisable](#) (void)
Disable global interrupts.
- void [CpulrqEnable](#) (void)
Enable global interrupts.

7.393.1 Detailed Description

Bootloader cpu module source file.

7.393.2 Function Documentation

7.393.2.1 CpulrqDisable()

```
void CpuIrqDisable (  
    void )
```

Disable global interrupts.

Returns

none.

7.393.2.2 CpulrqEnable()

```
void CpuIrqEnable (  
    void )
```

Enable global interrupts.

Returns

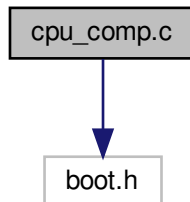
none.

7.394 cpu_comp.c File Reference

Bootloader cpu module source file.

```
#include "boot.h"
```

Include dependency graph for HCS12/CodeWarrior/cpu_comp.c:



Functions

- void [CpulrQDisable](#) (void)
Disable global interrupts.
- void [CpulrQEnable](#) (void)
Enable global interrupts.

7.394.1 Detailed Description

Bootloader cpu module source file.

7.394.2 Function Documentation

7.394.2.1 CpulrQDisable()

```
void CpuIrqDisable (  
    void )
```

Disable global interrupts.

Returns

none.

7.394.2.2 CpuIrqEnable()

```
void CpuIrqEnable (
    void )
```

Enable global interrupts.

Returns

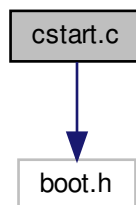
none.

7.395 cstart.c File Reference

Bootloader C startup source file.

```
#include "boot.h"
```

Include dependency graph for ARMC32_EFM32_Olimex_EM32G880F128STK_GCC/Boot/cstart.c:



Functions

- int [main](#) (void)

This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.

- void [reset_handler](#) (void)

Reset interrupt service routine. Configures the stack, initializes RAM and jumps to function main.

7.395.1 Detailed Description

Bootloader C startup source file.

7.395.2 Function Documentation

7.395.2.1 main()

```
int main (
    void )
```

This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.

Returns

Program return code.
none.
Program exit code.
None.

7.395.2.2 reset_handler()

```
void reset_handler (
    void )
```

Reset interrupt service routine. Configures the stack, initializes RAM and jumps to function main.

Returns

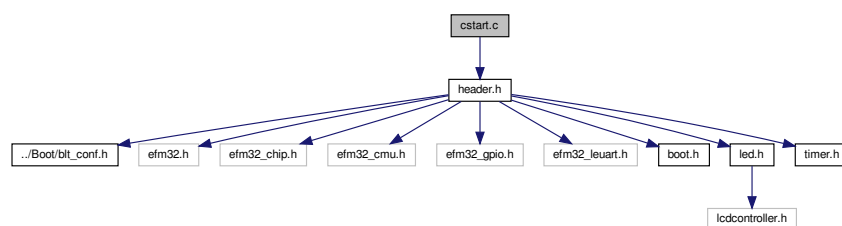
none.

7.396 cstart.c File Reference

Demo program C startup source file.

```
#include "header.h"
```

Include dependency graph for ARMCM3_EFM32_Olimex_EM32G880F128STK_GCC/Prog/cstart.c:



Functions

- int [main](#) (void)

This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.

- void [reset_handler](#) (void)

Reset interrupt service routine. Configures the stack, initializes RAM and jumps to function main.

7.396.1 Detailed Description

Demo program C startup source file.

7.396.2 Function Documentation

7.396.2.1 main()

```
int main (  
    void )
```

This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.

Returns

Program return code.
none.
Program exit code.
None.

7.396.2.2 reset_handler()

```
void reset_handler (  
    void )
```

Reset interrupt service routine. Configures the stack, initializes RAM and jumps to function main.

Returns

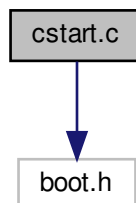
none.

7.397 cstart.c File Reference

Bootloader C startup source file.

```
#include "boot.h"
```

Include dependency graph for ARMC3_M3S_EK_M3S6965_GCC/Boot/cstart.c:



Functions

- int [main](#) (void)

This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.

- void [reset_handler](#) (void)

Reset interrupt service routine. Configures the stack, initializes RAM and jumps to function main.

7.397.1 Detailed Description

Bootloader C startup source file.

7.397.2 Function Documentation

7.397.2.1 main()

```
int main (  
    void )
```

This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.

Returns

Program return code.
none.
Program exit code.
None.

7.397.2.2 reset_handler()

```
void reset_handler (  
    void )
```

Reset interrupt service routine. Configures the stack, initializes RAM and jumps to function main.

Returns

none.

7.398.2.2 reset_handler()

```
void reset_handler (
    void )
```

Reset interrupt service routine. Configures the stack, initializes RAM and jumps to function main.

Returns

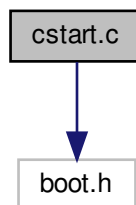
none.

7.399 cstart.c File Reference

Bootloader C startup source file.

```
#include "boot.h"
```

Include dependency graph for ARMC3M3_LM3S_EK_LM3S8962_GCC/Boot/cstart.c:



Functions

- int [main](#) (void)

This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.

- void [reset_handler](#) (void)

Reset interrupt service routine. Configures the stack, initializes RAM and jumps to function main.

7.399.1 Detailed Description

Bootloader C startup source file.

7.399.2 Function Documentation

7.399.2.1 main()

```
int main (
                                void )
```

This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.

Returns

Program return code.
none.
Program exit code.
None.

7.399.2.2 reset_handler()

```
void reset_handler (
    void )
```

Reset interrupt service routine. Configures the stack, initializes RAM and jumps to function main.

Returns

none.

7.400 cstart.c File Reference

Demo program C startup source file.

```
#include "header.h"
```

Include dependency graph for ARMCM3_LM3S_EK_LM3S8962_GCC/Prog/cstart.c:



Functions

- int [main](#) (void)
This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.
- void [reset_handler](#) (void)
Reset interrupt service routine. Configures the stack, initializes RAM and jumps to function main.

7.400.1 Detailed Description

Demo program C startup source file.

7.400.2 Function Documentation

7.400.2.1 main()

```
int main (
    void )
```

This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.

Returns

Program return code.
none.
Program exit code.
None.

7.400.2.2 reset_handler()

```
void reset_handler (
    void )
```

Reset interrupt service routine. Configures the stack, initializes RAM and jumps to function main.

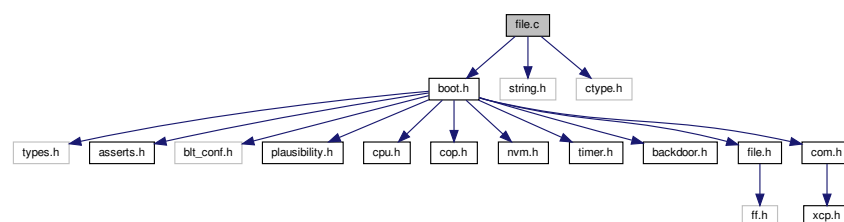
Returns

none.

7.401 file.c File Reference

Bootloader file system interface source file.

```
#include "boot.h"
#include <string.h>
#include <ctype.h>
Include dependency graph for file.c:
```



Data Structures

- struct [tFileEraseInfo](#)
Structure type with information for the memory erase operation.
- struct [tFatFsObjects](#)
Structure type for grouping FATFS related objects used by this module.

Enumerations

- enum [tFirmwareUpdateState](#) { [FIRMWARE_UPDATE_STATE_IDLE](#), [FIRMWARE_UPDATE_STATE_STARTING](#), [FIRMWARE_UPDATE_STATE_ERASING](#), [FIRMWARE_UPDATE_STATE_PROGRAMMING](#) }
Enumeration for the different internal module states.

Functions

- static [blt_char](#) [FileLibByteNibbleToChar](#) ([blt_int8u](#) nibble)
Helper function to convert a 4-bit value to a character that represents its value in hexadecimal format. Example: FileLibByteNibbleToChar(11) -> returns 'B'.
- static [blt_char](#) * [FileLibByteToHexString](#) ([blt_int8u](#) byte_val, [blt_char](#) *destination)
Helper function to convert a byte value to a string representing the value in hexadecimal format. Example: FileLibByteToHexString(28, strBuffer) -> returns "1C".
- static [blt_char](#) * [FileLibLongToIntString](#) ([blt_int32u](#) long_val, [blt_char](#) *destination)
Helper function to convert a 32-bit unsigned number to a string that represents its decimal value. Example: FileLibLongToIntString(1234, strBuffer) -> returns "1234".
- static [blt_int8u](#) [FileLibHexStringToByte](#) (const [blt_char](#) *hexstring)
Helper function to convert a sequence of 2 characters that represent a hexadecimal value to the actual byte value. Example: FileLibHexStringToByte("2f") -> returns 47.
- [blt_bool](#) [FileIsFirmwareUpdateRequestedHook](#) (void)
Callback that gets called to check whether a firmware update from local file storage should be started. This could for example be when a switch is pressed, when a certain file is found on the local file storage, etc.
- const [blt_char](#) * [FileGetFirmwareFilenameHook](#) (void)
Callback to obtain the filename of the firmware file that should be used during the firmware update from the local file storage. This hook function is called at the beginning of the firmware update from local storage sequence.
- void [FileFirmwareUpdateStartedHook](#) (void)
Callback that gets called to inform the application that a firmware update from local storage just started.
- void [FileFirmwareUpdateCompletedHook](#) (void)
Callback that gets called to inform the application that a firmware update was successfully completed.
- void [FileFirmwareUpdateErrorHook](#) ([blt_int8u](#) error_code)
Callback that gets called in case an error occurred during a firmware update. Refer to <file.h> for a list of available error codes.
- void [FileFirmwareUpdateLogHook](#) ([blt_char](#) *info_string)
Callback that gets called each time new log information becomes available during a firmware update.
- void [FileInit](#) (void)
Initializes the file system interface module. The initial firmware update state is set to idle and the file system is mounted as logical disk 0.
- [blt_bool](#) [FileIsIdle](#) (void)
This function checks if a firmware update through the locally attached storage is in progress or not (idle).
- [blt_bool](#) [FileHandleFirmwareUpdateRequest](#) (void)
This function checks if a firmware update through the locally attached storage is requested to be started and if so processes this request by transitioning from the IDLE to the STARTING state.
- void [FileTask](#) (void)

File system task function for managing the firmware updates from locally attached storage.

- [tSrecLineType](#) [FileSrecGetLineType](#) (const [blt_char](#) *line)
Inspects a line from a Motorola S-Record file to determine its type.
- [blt_bool](#) [FileSrecVerifyChecksum](#) (const [blt_char](#) *line)
Inspects an S1, S2 or S3 line from a Motorola S-Record file to determine if the checksum at the end is correct.
- [blt_int16s](#) [FileSrecParseLine](#) (const [blt_char](#) *line, [blt_addr](#) *address, [blt_int8u](#) *data)
Parses a line from a Motorola S-Record file and looks for S1, S2 or S3 lines with data. Note that if a null pointer is passed as the data parameter, then no data is extracted from the line.

Variables

- static [tFirmwareUpdateState](#) [firmwareUpdateState](#)
Local variable that holds the internal module state.
- static [tFatFsObjects](#) [fatFsObjects](#)
Local variable for the used FATFS objects in this module.
- static [tSrecLineParseObject](#) [lineParseObject](#)
Local variable for storing S-record line parsing results.
- static [tFileEraseInfo](#) [eraseInfo](#)
Local variable for storing information regarding the memory erase operation.
- static [blt_char](#) [loggingStr](#) [64]
Local character buffer for storing the string with log information.

7.401.1 Detailed Description

Bootloader file system interface source file.

7.401.2 Enumeration Type Documentation

7.401.2.1 tFirmwareUpdateState

```
enum tFirmwareUpdateState
```

Enumeration for the different internal module states.

Enumerator

FIRMWARE_UPDATE_STATE_IDLE	idle state
FIRMWARE_UPDATE_STATE_STARTING	starting state
FIRMWARE_UPDATE_STATE_ERASING	erasing state
FIRMWARE_UPDATE_STATE_PROGRAMMING	programming state

7.401.3 Function Documentation

7.401.3.1 FileFirmwareUpdateCompletedHook()

```
void FileFirmwareUpdateCompletedHook (  
    void )
```

Callback that gets called to inform the application that a firmware update was successfully completed.

Returns

none.

Referenced by FileTask().

7.401.3.2 FileFirmwareUpdateErrorHook()

```
void FileFirmwareUpdateErrorHook (  
    blt_int8u error_code )
```

Callback that gets called in case an error occurred during a firmware update. Refer to <[file.h](#)> for a list of available error codes.

Returns

none.

Referenced by FileTask().

7.401.3.3 FileFirmwareUpdateLogHook()

```
void FileFirmwareUpdateLogHook (  
    blt_char * info_string )
```

Callback that gets called each time new log information becomes available during a firmware update.

Parameters

<i>info_string</i>	Pointer to a character array with the log entry info.
--------------------	---

Returns

none.

Referenced by FileTask().

7.401.3.4 FileFirmwareUpdateStartedHook()

```
void FileFirmwareUpdateStartedHook (  
    void )
```

Callback that gets called to inform the application that a firmware update from local storage just started.

Returns

none.

Referenced by FileTask().

7.401.3.5 FileGetFirmwareFilenameHook()

```
const blt_char* FileGetFirmwareFilenameHook (  
    void )
```

Callback to obtain the filename of the firmware file that should be used during the firmware update from the local file storage. This hook function is called at the beginning of the firmware update from local storage sequence.

Returns

valid firmware filename with full path or BLT_NULL.

Referenced by FileTask().

7.401.3.6 FileHandleFirmwareUpdateRequest()

```
blt_bool FileHandleFirmwareUpdateRequest (  
    void )
```

This function checks if a firmware update through the locally attached storage is requested to be started and if so processes this request by transitioning from the IDLE to the STARTING state.

Returns

BLT_TRUE when a firmware update is requested, BLT_FALSE otherwise.

Referenced by BackDoorCheck(), and BackDoorInit().

7.401.3.7 FileInit()

```
void FileInit (
    void )
```

Initializes the file system interface module. The initial firmware update state is set to idle and the file system is mounted as logical disk 0.

Returns

none

Referenced by BootInit().

7.401.3.8 FileIsFirmwareUpdateRequestedHook()

```
blt_bool FileIsFirmwareUpdateRequestedHook (
    void )
```

Callback that gets called to check whether a firmware update from local file storage should be started. This could for example be when a switch is pressed, when a certain file is found on the local file storage, etc.

Returns

BLT_TRUE if a firmware update is requested, BLT_FALSE otherwise.

Referenced by FileHandleFirmwareUpdateRequest().

7.401.3.9 FileIsIdle()

```
blt_bool FileIsIdle (
    void )
```

This function checks if a firmware update through the locally attached storage is in progress or not (idle).

Returns

BLT_TRUE when in idle state, BLT_FALSE otherwise.

Referenced by BackDoorCheck(), and XcpCmdConnect().

7.401.3.10 FileLibByteNibbleToChar()

```
static blt_char FileLibByteNibbleToChar (
    blt_int8u nibble ) [static]
```

Helper function to convert a 4-bit value to a character that represents its value in hexadecimal format. Example: FileLibByteNibbleToChar(11) -> returns 'B'.

Parameters

<i>nibble</i>	4-bit value to convert.
---------------	-------------------------

Returns

The resulting byte value.

Referenced by FileLibByteToHexString().

7.401.3.11 FileLibByteToHexString()

```
static blt_char * FileLibByteToHexString (
    blt_int8u byte_val,
    blt_char * destination ) [static]
```

Helper function to convert a byte value to a string representing the value in hexadecimal format. Example: FileLibByteToHexString(28, strBuffer) -> returns "1C".

Parameters

<i>byte_val</i>	8-bit value to convert.
<i>destination</i>	Pointer to character buffer for storing the results.

Returns

The resulting string.

Referenced by FileTask().

7.401.3.12 FileLibHexStringToByte()

```
static blt_int8u FileLibHexStringToByte (
    const blt_char * hexstring ) [static]
```

Helper function to convert a sequence of 2 characters that represent a hexadecimal value to the actual byte value. Example: FileLibHexStringToByte("2f") -> returns 47.

Parameters

<i>hexstring</i>	String beginning with 2 characters that represent a hexa- decimal value.
------------------	--

Returns

The resulting byte value.

Referenced by FileSrecParseLine(), and FileSrecVerifyChecksum().

7.401.3.13 FileLibLongToIntString()

```
static blt_char * FileLibLongToIntString (
    blt_int32u long_val,
    blt_char * destination ) [static]
```

Helper function to convert a 32-bit unsigned number to a string that represents its decimal value. Example: File↵
LibLongToIntString(1234, strBuffer) -> returns "1234".

Parameters

<i>long_val</i>	32-bit value to convert.
<i>destination</i>	Pointer to character buffer for storing the results.

Returns

The resulting string.

Referenced by FileTask().

7.401.3.14 FileSrecGetLineType()

```
tSrecLineType FileSrecGetLineType (
    const blt_char * line )
```

Inspects a line from a Motorola S-Record file to determine its type.

Parameters

<i>line</i>	A line from the S-Record.
-------------	---------------------------

Returns

the S-Record line type.

Referenced by FileSrecParseLine().

7.401.3.15 FileSrecParseLine()

```
blt_int16s FileSrecParseLine (
    const blt_char * line,
    blt_addr * address,
    blt_int8u * data )
```

Parses a line from a Motorola S-Record file and looks for S1, S2 or S3 lines with data. Note that if a null pointer is passed as the data parameter, then no data is extracted from the line.

Parameters

<i>line</i>	A line from the S-Record.
<i>address</i>	Address found in the S-Record data line.
<i>data</i>	Byte array where the data bytes from the S-Record data line are stored.

Returns

The number of data bytes found on the S-record data line, 0 in case the line is not an S1, S2 or S3 line or ERROR_SREC_INVALID_CHECKSUM in case the checksum validation failed.

Referenced by FileTask().

7.401.3.16 FileSrecVerifyChecksum()

```
blt_bool FileSrecVerifyChecksum (
    const blt_char * line )
```

Inspects an S1, S2 or S3 line from a Motorola S-Record file to determine if the checksum at the end is correct.

Parameters

<i>line</i>	An S1, S2 or S3 line from the S-Record.
-------------	---

Returns

BLT_TRUE if the checksum is correct, BLT_FALSE otherwise.

Referenced by FileSrecParseLine().

7.401.3.17 FileTask()

```
void FileTask (
    void )
```

File system task function for managing the firmware updates from locally attached storage.

Returns

none.

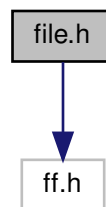
Referenced by BootTask().

7.402 file.h File Reference

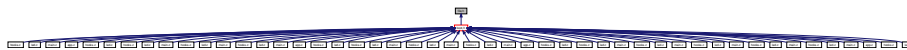
Bootloader file system interface header file.

```
#include "ff.h"
```

Include dependency graph for file.h:



This graph shows which files directly or indirectly include this file:

**Data Structures**

- struct [tSrecLineParseObject](#)
Structure type for grouping the parsing results of an S-record line.

Macros

- #define [FILE_ERROR_CANNOT_OPEN_FIRMWARE_FILE](#) (1)
Error code for not being able to open the firmware file.
- #define [FILE_ERROR_CANNOT_READ_FROM_FILE](#) (2)
Error code for not being able to read from the firmware file.
- #define [FILE_ERROR_INVALID_CHECKSUM_IN_FILE](#) (3)
Error code because in incorrect checksum was found in the firmware file.
- #define [FILE_ERROR_REWINDING_FILE_READ_POINTER](#) (4)
Error code because the file pointers read pointer could not be rewinded.
- #define [FILE_ERROR_CANNOT_ERASE_MEMORY](#) (5)

- Error code because an error occurred during the memory erase operation.*
 - `#define FILE_ERROR_CANNOT_PROGRAM_MEMORY` (6)
- Error code because an error occurred during the memory write operation.*
 - `#define FILE_ERROR_CANNOT_WRITE_CHECKSUM` (7)
- Error code because the program's checksum could not be written to memory.*
 - `#define MAX_CHARS_PER_LINE` (256)
- Maximum number of characters that can be on a line in the firmware file.*
 - `#define MAX_DATA_BYTES_PER_LINE` (`MAX_CHARS_PER_LINE/2`)
- Maximum number of data bytes that can be on a line in the firmware file (S-record).*
 - `#define ERROR_SREC_INVALID_CHECKSUM` (-1)
- Return code in case an invalid checksum was detected on an S-record line.*

Enumerations

- `enum tSrecLineType { LINE_TYPE_S1, LINE_TYPE_S2, LINE_TYPE_S3, LINE_TYPE_UNSUPPORTED }`
Enumeration for the different S-record line types.

Functions

- `void FileInit` (void)
Initializes the file system interface module. The initial firmware update state is set to idle and the file system is mounted as logical disk 0.
- `void FileTask` (void)
File system task function for managing the firmware updates from locally attached storage.
- `blt_bool FileIsIdle` (void)
This function checks if a firmware update through the locally attached storage is in progress or not (idle).
- `blt_bool FileHandleFirmwareUpdateRequest` (void)
This function checks if a firmware update through the locally attached storage is requested to be started and if so processes this request by transitioning from the IDLE to the STARTING state.
- `tSrecLineType FileSrecGetLineType` (const `blt_char` *line)
Inspects a line from a Motorola S-Record file to determine its type.
- `blt_bool FileSrecVerifyChecksum` (const `blt_char` *line)
Inspects an S1, S2 or S3 line from a Motorola S-Record file to determine if the checksum at the end is correct.
- `blt_int16s FileSrecParseLine` (const `blt_char` *line, `blt_addr` *address, `blt_int8u` *data)
Parses a line from a Motorola S-Record file and looks for S1, S2 or S3 lines with data. Note that if a null pointer is passed as the data parameter, then no data is extracted from the line.

7.402.1 Detailed Description

Bootloader file system interface header file.

7.402.2 Enumeration Type Documentation

7.402.2.1 tSrecLineType

```
enum tSrecLineType
```

Enumeration for the different S-record line types.

Enumerator

LINE_TYPE_S1	16-bit address line
LINE_TYPE_S2	24-bit address line
LINE_TYPE_S3	32-bit address line
LINE_TYPE_UNSUPPORTED	unsupported line

7.402.3 Function Documentation

7.402.3.1 FileHandleFirmwareUpdateRequest()

```
blt_bool FileHandleFirmwareUpdateRequest (  
    void )
```

This function checks if a firmware update through the locally attached storage is requested to be started and if so processes this request by transitioning from the IDLE to the STARTING state.

Returns

BLT_TRUE when a firmware update is requested, BLT_FALSE otherwise.

Referenced by BackDoorCheck(), and BackDoorInit().

7.402.3.2 FileInit()

```
void FileInit (  
    void )
```

Initializes the file system interface module. The initial firmware update state is set to idle and the file system is mounted as logical disk 0.

Returns

none

Referenced by BootInit().

7.402.3.3 FileIsIdle()

```
blt_bool FileIsIdle (
    void )
```

This function checks if a firmware update through the locally attached storage is in progress or not (idle).

Returns

BLT_TRUE when in idle state, BLT_FALSE otherwise.

Referenced by BackDoorCheck(), and XcpCmdConnect().

7.402.3.4 FileSrecGetLineType()

```
tSrecLineType FileSrecGetLineType (
    const blt_char * line )
```

Inspects a line from a Motorola S-Record file to determine its type.

Parameters

<i>line</i>	A line from the S-Record.
-------------	---------------------------

Returns

the S-Record line type.

Referenced by FileSrecParseLine().

7.402.3.5 FileSrecParseLine()

```
blt_int16s FileSrecParseLine (
    const blt_char * line,
    blt_addr * address,
    blt_int8u * data )
```

Parses a line from a Motorola S-Record file and looks for S1, S2 or S3 lines with data. Note that if a null pointer is passed as the data parameter, then no data is extracted from the line.

Parameters

<i>line</i>	A line from the S-Record.
<i>address</i>	Address found in the S-Record data line.
<i>data</i>	Byte array where the data bytes from the S-Record data line are stored.

Returns

The number of data bytes found on the S-record data line, 0 in case the line is not an S1, S2 or S3 line or `ERROR_SREC_INVALID_CHECKSUM` in case the checksum validation failed.

Referenced by `FileTask()`.

7.402.3.6 FileSrecVerifyChecksum()

```
blt_bool FileSrecVerifyChecksum (
    const blt_char * line )
```

Inspects an S1, S2 or S3 line from a Motorola S-Record file to determine if the checksum at the end is correct.

Parameters

<i>line</i>	An S1, S2 or S3 line from the S-Record.
-------------	---

Returns

`BLT_TRUE` if the checksum is correct, `BLT_FALSE` otherwise.

Referenced by `FileSrecParseLine()`.

7.402.3.7 FileTask()

```
void FileTask (
    void )
```

File system task function for managing the firmware updates from locally attached storage.

Returns

none.

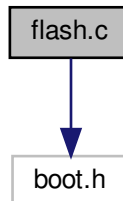
Referenced by `BootTask()`.

7.403 flash.c File Reference

Bootloader flash driver source file.

```
#include "boot.h"
```

Include dependency graph for _template/flash.c:



Data Structures

- struct [tFlashSector](#)
Flash sector descriptor type.
- struct [tFlashBlockInfo](#)
Structure type for grouping flash block information.

Macros

- #define [FLASH_INVALID_SECTOR_IDX](#) (0xff)
Value for an invalid sector entry index into flashLayout[].
- #define [FLASH_INVALID_ADDRESS](#) (0xffffffff)
Value for an invalid flash address.
- #define [FLASH_WRITE_BLOCK_SIZE](#) (512)
Standard size of a flash block for writing.
- #define [FLASH_TOTAL_SECTORS](#) (sizeof(flashLayout)/sizeof(flashLayout[0]))
Total numbers of sectors in array flashLayout[].
- #define [FLASH_END_ADDRESS](#)
End address of the bootloader programmable flash.
- #define [BOOT_FLASH_VECTOR_TABLE_CS_OFFSET](#) (0x188)
Offset into the user program's vector table where the checksum is located. For this target it is set to the end of the vector table. Note that the value can be overridden in blt_conf.h, because the size of the vector table could vary. When changing this value, don't forget to update the location of the checksum in the user program accordingly. Otherwise the checksum verification will always fail.

Functions

- static `blt_bool` `FlashInitBlock` (`tFlashBlockInfo` *block, `blt_addr` address)
Copies data currently in flash to the block->data and sets the base address.
- static `tFlashBlockInfo` * `FlashSwitchBlock` (`tFlashBlockInfo` *block, `blt_addr` base_addr)
Switches blocks by programming the current one and initializing the next.
- static `blt_bool` `FlashAddToBlock` (`tFlashBlockInfo` *block, `blt_addr` address, `blt_int8u` *data, `blt_int32u` len)
Programming is done per block. This function adds data to the block that is currently collecting data to be written to flash. If the address is outside of the current block, the current block is written to flash and a new block is initialized.
- static `blt_bool` `FlashWriteBlock` (`tFlashBlockInfo` *block)
Programs FLASH_WRITE_BLOCK_SIZE bytes to flash from the block->data array.
- static `blt_bool` `FlashEraseSectors` (`blt_int8u` first_sector_idx, `blt_int8u` last_sector_idx)
Erases the flash sectors from indices first_sector_idx up until last_sector_idx into the flashLayout[] array.
- static `blt_int8u` `FlashGetSectorIdx` (`blt_addr` address)
Determines the index into the flashLayout[] array of the flash sector that the specified address is in.
- void `FlashInit` (void)
Initializes the flash driver.
- void `FlashReinit` (void)
Reinitializes the flash driver.
- `blt_bool` `FlashWrite` (`blt_addr` addr, `blt_int32u` len, `blt_int8u` *data)
Writes the data to flash through a flash block manager. Note that this function also checks that no data is programmed outside the flash memory region, so the bootloader can never be overwritten.
- `blt_bool` `FlashErase` (`blt_addr` addr, `blt_int32u` len)
Erases the flash memory. Note that this function also checks that no data is erased outside the flash memory region, so the bootloader can never be erased.
- `blt_bool` `FlashWriteChecksum` (void)
Writes a checksum of the user program to non-volatile memory. This is performed once the entire user program has been programmed. Through the checksum, the bootloader can check if the programming session was completed, which indicates that a valid user programming is present and can be started.
- `blt_bool` `FlashVerifyChecksum` (void)
Verifies the checksum, which indicates that a valid user program is present and can be started.
- `blt_bool` `FlashDone` (void)
Finalizes the flash driver operations. There could still be data in the currently active block that needs to be flashed.
- `blt_addr` `FlashGetUserProgBaseAddress` (void)
Obtains the base address of the flash memory available to the user program. This is basically the first address in the flashLayout table.

Variables

- static const `tFlashSector` `flashLayout` []
If desired, it is possible to set BOOT_FLASH_CUSTOM_LAYOUT_ENABLE to > 0 in blt_conf.h and then implement your own version of the flashLayout[] table in a source-file with the name flash_layout.c. This way you customize the flash memory size reserved for the bootloader, without having to modify the flashLayout[] table in this file directly. This file will then include flash_layout.c so there is no need to compile it additionally with your project.
- static `tFlashBlockInfo` `blockInfo`
Local variable with information about the flash block that is currently being operated on.
- static `tFlashBlockInfo` `bootBlockInfo`
Local variable with information about the flash boot block.

7.403.1 Detailed Description

Bootloader flash driver source file.

7.403.2 Function Documentation

7.403.2.1 FlashAddToBlock()

```
static blt_bool FlashAddToBlock (
    tFlashBlockInfo * block,
    blt_addr address,
    blt_int8u * data,
    blt_int32u len ) [static]
```

Programming is done per block. This function adds data to the block that is currently collecting data to be written to flash. If the address is outside of the current block, the current block is written to flash and a new block is initialized.

Parameters

<i>block</i>	Pointer to flash block info structure to operate on.
<i>address</i>	Flash destination address.
<i>data</i>	Pointer to the byte array with data.
<i>len</i>	Number of bytes to add to the block.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

Referenced by FlashWrite().

7.403.2.2 FlashDone()

```
blt_bool FlashDone (
    void )
```

Finalizes the flash driver operations. There could still be data in the currently active block that needs to be flashed.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

Referenced by NvmDone().

7.403.2.3 FlashErase()

```
blt_bool FlashErase (
    blt_addr addr,
    blt_int32u len )
```

Erases the flash memory. Note that this function also checks that no data is erased outside the flash memory region, so the bootloader can never be erased.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

Referenced by NvmErase().

7.403.2.4 FlashEraseSectors()

```
static blt_bool FlashEraseSectors (  
    blt_int8u first_sector_idx,  
    blt_int8u last_sector_idx ) [static]
```

Erases the flash sectors from indices first_sector_idx up until last_sector_idx into the flashLayout[] array.

Parameters

<i>first_sector_idx</i>	First flash sector number index into flashLayout[].
<i>last_sector_idx</i>	Last flash sector number index into flashLayout[].

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

Referenced by FlashErase().

7.403.2.5 FlashGetSectorIdx()

```
static blt_int8u FlashGetSectorIdx (  
    blt_addr address ) [static]
```

Determines the index into the flashLayout[] array of the flash sector that the specified address is in.

Parameters

<i>address</i>	Address in the flash sector.
----------------	------------------------------

Returns

Flash sector index in flashLayout[] or FLASH_INVALID_SECTOR_IDX.

Referenced by FlashErase(), FlashWrite(), and FlashWriteBlock().

7.403.2.6 FlashGetUserProgBaseAddress()

```
blt_addr FlashGetUserProgBaseAddress (
    void )
```

Obtains the base address of the flash memory available to the user program. This is basically the first address in the flashLayout table.

Returns

Base address.

Referenced by NvmGetUserProgBaseAddress().

7.403.2.7 FlashInit()

```
void FlashInit (
    void )
```

Initializes the flash driver.

Returns

none.

Referenced by NvmInit().

7.403.2.8 FlashInitBlock()

```
static blt_bool FlashInitBlock (
    tFlashBlockInfo * block,
    blt_addr address ) [static]
```

Copies data currently in flash to the block->data and sets the base address.

Parameters

<i>block</i>	Pointer to flash block info structure to operate on.
<i>address</i>	Base address of the block data.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

Referenced by FlashAddToBlock(), and FlashSwitchBlock().

7.403.2.9 FlashReinit()

```
void FlashReinit (
    void )
```

Reinitializes the flash driver.

Returns

none.

Referenced by NvmReinit().

7.403.2.10 FlashSwitchBlock()

```
static tFlashBlockInfo * FlashSwitchBlock (
    tFlashBlockInfo * block,
    blt_addr base_addr ) [static]
```

Switches blocks by programming the current one and initializing the next.

Parameters

<i>block</i>	Pointer to flash block info structure to operate on.
<i>base_addr</i>	Base address of the next block.

Returns

The pointer of the block info struct that is now being used, or a NULL pointer in case of error.

Referenced by FlashAddToBlock().

7.403.2.11 FlashVerifyChecksum()

```
blt_bool FlashVerifyChecksum (
    void )
```

Verifies the checksum, which indicates that a valid user program is present and can be started.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

Referenced by NvmVerifyChecksum().

7.403.2.12 FlashWrite()

```
blt_bool FlashWrite (
    blt_addr addr,
    blt_int32u len,
    blt_int8u * data )
```

Writes the data to flash through a flash block manager. Note that this function also checks that no data is programmed outside the flash memory region, so the bootloader can never be overwritten.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.
<i>data</i>	Pointer to the data buffer.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

Referenced by FlashWriteChecksum(), and NvmWrite().

7.403.2.13 FlashWriteBlock()

```
static blt_bool FlashWriteBlock (
    tFlashBlockInfo * block ) [static]
```

Programs FLASH_WRITE_BLOCK_SIZE bytes to flash from the block->data array.

Parameters

<i>block</i>	Pointer to flash block info structure to operate on.
--------------	--

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

Referenced by FlashDone(), and FlashSwitchBlock().

7.403.2.14 FlashWriteChecksum()

```
blt_bool FlashWriteChecksum (  
    void )
```

Writes a checksum of the user program to non-volatile memory. This is performed once the entire user program has been programmed. Through the checksum, the bootloader can check if the programming session was completed, which indicates that a valid user programming is present and can be started.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

Referenced by NvmDone().

7.403.3 Variable Documentation

7.403.3.1 blockInfo

```
tFlashBlockInfo blockInfo [static]
```

Local variable with information about the flash block that is currently being operated on.

The smallest amount of flash that can be programmed is FLASH_WRITE_BLOCK_SIZE. A flash block manager is implemented in this driver and stores info in this variable. Whenever new data should be flashed, it is first added to a RAM buffer, which is part of this variable. Whenever the RAM buffer, which has the size of a flash block, is full or data needs to be written to a different block, the contents of the RAM buffer are programmed to flash. The flash block manager requires some software overhead, yet results in faster flash programming because data is first harvested, ideally until there is enough to program an entire flash block, before the flash device is actually operated on.

Referenced by FlashSwitchBlock().

7.403.3.2 bootBlockInfo

```
tFlashBlockInfo bootBlockInfo [static]
```

Local variable with information about the flash boot block.

The first block of the user program holds the vector table, which on the STM32 is also the where the checksum is written to. It is likely that the vector table is first flashed and then, at the end of the programming sequence, the checksum. This means that this flash block needs to be written to twice. Normally this is not a problem with flash memory, as long as you write the same values to those bytes that are not supposed to be changed and the locations where you do write to are still in the erased 0xFF state. Unfortunately, writing twice to flash this way, does not work reliably on all micros. This is why we need to have an extra block, the bootblock, placed under the management of the block manager. This way it is possible to implement functionality so that the bootblock is only written to once at the end of the programming sequence.

Referenced by FlashSwitchBlock().

7.403.3.3 flashLayout

```
const tFlashSector flashLayout[] [static]
```

Initial value:

```
=
{

    { 0x08008000, 0x04000, 2},
    { 0x0800C000, 0x04000, 3},

    { 0x08010000, 0x04000, 4},
    { 0x08014000, 0x04000, 5},
    { 0x08018000, 0x04000, 6},
    { 0x0801C000, 0x04000, 7},

    { 0x08020000, 0x04000, 8},
    { 0x08024000, 0x04000, 9},
    { 0x08028000, 0x04000, 10},
    { 0x0802C000, 0x04000, 11},
    { 0x08030000, 0x04000, 12},
    { 0x08034000, 0x04000, 13},
    { 0x08038000, 0x04000, 14},
    { 0x0803C000, 0x04000, 15},

}
```

If desired, it is possible to set `BOOT_FLASH_CUSTOM_LAYOUT_ENABLE` to `> 0` in `blt_conf.h` and then implement your own version of the `flashLayout[]` table in a source-file with the name `flash_layout.c`. This way you customize the flash memory size reserved for the bootloader, without having to modify the `flashLayout[]` table in this file directly. This file will then include `flash_layout.c` so there is no need to compile it additionally with your project.

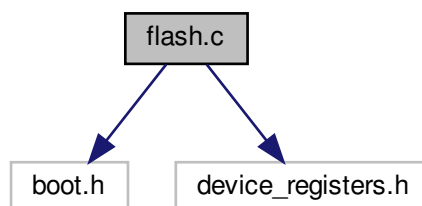
Array with the layout of the flash memory.

Also controls what part of the flash memory is reserved for the bootloader. If the bootloader size changes, the reserved sectors for the bootloader might need adjustment to make sure the bootloader doesn't get overwritten.

7.404 flash.c File Reference

Bootloader flash driver source file.

```
#include "boot.h"
#include "device_registers.h"
Include dependency graph for ARMCM0_S32K11/flash.c:
```



Data Structures

- struct [tFlashSector](#)
Flash sector descriptor type.
- struct [tFlashBlockInfo](#)
Structure type for grouping flash block information.

Macros

- #define [FLASH_INVALID_SECTOR_IDX](#) (0xff)
Value for an invalid sector entry index into flashLayout[].
- #define [FLASH_INVALID_ADDRESS](#) (0xffffffff)
Value for an invalid flash address.
- #define [FLASH_WRITE_BLOCK_SIZE](#) (256)
Standard size of a flash block for writing.
- #define [FLASH_ERASE_BLOCK_SIZE](#) (FEATURE_FLS_PF_BLOCK_SECTOR_SIZE)
Standard size of a flash block for erasing. This is either 2 or 4 kb depending on the microcontroller derivative.
- #define [FLASH_TOTAL_SECTORS](#) (sizeof(flashLayout)/sizeof(flashLayout[0]))
Total numbers of sectors in array flashLayout[].
- #define [FLASH_END_ADDRESS](#)
End address of the bootloader programmable flash.
- #define [FLASH_FTFC_CMD_PROGRAM_PHRASE](#) (0x07U)
FTFC program phrase command code.
- #define [FLASH_FTFC_CMD_ERASE_SECTOR](#) (0x09U)
FTFC erase sector command code.
- #define [BOOT_FLASH_VECTOR_TABLE_CS_OFFSET](#) (0xC0)
Offset into the user program's vector table where the checksum is located. Note that the value can be overridden in blt_conf.h, because the size of the vector table could vary. When changing this value, don't forget to update the location of the checksum in the user program accordingly. Otherwise the checksum verification will always fail.

Functions

- static [blt_bool](#) [FlashInitBlock](#) ([tFlashBlockInfo](#) *block, [blt_addr](#) address)
Copies data currently in flash to the block->data and sets the base address.
- static [tFlashBlockInfo](#) * [FlashSwitchBlock](#) ([tFlashBlockInfo](#) *block, [blt_addr](#) base_addr)
Switches blocks by programming the current one and initializing the next.
- static [blt_bool](#) [FlashAddToBlock](#) ([tFlashBlockInfo](#) *block, [blt_addr](#) address, [blt_int8u](#) *data, [blt_int32u](#) len)
Programming is done per block. This function adds data to the block that is currently collecting data to be written to flash. If the address is outside of the current block, the current block is written to flash and a new block is initialized.
- static [blt_bool](#) [FlashWriteBlock](#) ([tFlashBlockInfo](#) *block)
Programs FLASH_WRITE_BLOCK_SIZE bytes to flash from the block->data array.
- static [blt_bool](#) [FlashEraseSectors](#) ([blt_int8u](#) first_sector_idx, [blt_int8u](#) last_sector_idx)
Erases the flash sectors from indices first_sector_idx up until last_sector_idx into the flashLayout[] array.
- static [blt_int8u](#) [FlashGetSectorIdx](#) ([blt_addr](#) address)
Determines the index into the flashLayout[] array of the flash sector that the specified address is in.
- static START_FUNCTION_DECLARATION_RAMSECTION void [FlashCommandSequence](#) (void)
If desired, it is possible to set BOOT_FLASH_CUSTOM_LAYOUT_ENABLE to > 0 in blt_conf.h and then implement your own version of the flashLayout[] table in a source-file with the name flash_layout.c. This way you customize the flash memory size reserved for the bootloader, without having to modify the flashLayout[] table in this file directly. This file will then include flash_layout.c so there is no need to compile it additionally with your project.
- void [FlashInit](#) (void)

- Initializes the flash driver.*

 - `void FlashReinit (void)`

Reinitializes the flash driver.
- `blt_bool FlashWrite (blt_addr addr, blt_int32u len, blt_int8u *data)`

Writes the data to flash through a flash block manager. Note that this function also checks that no data is programmed outside the flash memory region, so the bootloader can never be overwritten.
- `blt_bool FlashErase (blt_addr addr, blt_int32u len)`

Erases the flash memory. Note that this function also checks that no data is erased outside the flash memory region, so the bootloader can never be erased.
- `blt_bool FlashWriteChecksum (void)`

Writes a checksum of the user program to non-volatile memory. This is performed once the entire user program has been programmed. Through the checksum, the bootloader can check if the programming session was completed, which indicates that a valid user programming is present and can be started.
- `blt_bool FlashVerifyChecksum (void)`

Verifies the checksum, which indicates that a valid user program is present and can be started.
- `blt_bool FlashDone (void)`

Finalizes the flash driver operations. There could still be data in the currently active block that needs to be flashed.
- `blt_addr FlashGetUserProgBaseAddress (void)`

Obtains the base address of the flash memory available to the user program. This is basically the first address in the flashLayout table.

Variables

- `static tFlashBlockInfo blockInfo`
- Local variable with information about the flash block that is currently being operated on.*
- `static tFlashBlockInfo bootBlockInfo`
- Local variable with information about the flash boot block.*

7.404.1 Detailed Description

Bootloader flash driver source file.

7.404.2 Function Documentation

7.404.2.1 FlashAddToBlock()

```
static blt_bool FlashAddToBlock (
    tFlashBlockInfo * block,
    blt_addr address,
    blt_int8u * data,
    blt_int32u len ) [static]
```

Programming is done per block. This function adds data to the block that is currently collecting data to be written to flash. If the address is outside of the current block, the current block is written to flash and a new block is initialized.

Parameters

<i>block</i>	Pointer to flash block info structure to operate on.
<i>address</i>	Flash destination address.
<i>data</i>	Pointer to the byte array with data.
<i>len</i>	Number of bytes to add to the block.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

Referenced by FlashWrite().

7.404.2.2 FlashCommandSequence()

```
static START_FUNCTION_DEFINITION_RAMSECTION void FlashCommandSequence (  
    void ) [static]
```

If desired, it is possible to set `BOOT_FLASH_CUSTOM_LAYOUT_ENABLE` to `> 0` in `blt_conf.h` and then implement your own version of the `flashLayout[]` table in a source-file with the name `flash_layout.c`. This way you customize the flash memory size reserved for the bootloader, without having to modify the `flashLayout[]` table in this file directly. This file will then include `flash_layout.c` so there is no need to compile it additionally with your project.

Use the FTFC module to run the flash command sequence. It is assumed that that command and its necessary parameters were already written to the correct FTFC registers.

Array with the layout of the flash memory.

Also controls what part of the flash memory is reserved for the bootloader. If the bootloader size changes, the reserved sectors for the bootloader might need adjustment to make sure the bootloader doesn't get overwritten.

Attention

This function needs to run from RAM. It is configured such that the C start-up code automatically copies it from ROM to RAM in function `init_data_bss()`, which is called by the reset handler.

Returns

None.

Referenced by FlashEraseSectors(), FlashGetSectorIdx(), and FlashWriteBlock().

7.404.2.3 FlashDone()

```
blt_bool FlashDone (  
    void )
```

Finalizes the flash driver operations. There could still be data in the currently active block that needs to be flashed.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.404.2.4 FlashErase()

```
blt_bool FlashErase (  
    blt_addr addr,  
    blt_int32u len )
```

Erases the flash memory. Note that this function also checks that no data is erased outside the flash memory region, so the bootloader can never be erased.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.404.2.5 FlashEraseSectors()

```
static blt_bool FlashEraseSectors (  
    blt_int8u first_sector_idx,  
    blt_int8u last_sector_idx ) [static]
```

Erases the flash sectors from indices *first_sector_idx* up until *last_sector_idx* into the *flashLayout[]* array.

Parameters

<i>first_sector_idx</i>	First flash sector number index into <i>flashLayout[]</i> .
<i>last_sector_idx</i>	Last flash sector number index into <i>flashLayout[]</i> .

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

Referenced by FlashErase().

7.404.2.6 FlashGetSectorIdx()

```
static blt_int8u FlashGetSectorIdx (  
    blt_addr address ) [static]
```

Determines the index into the flashLayout[] array of the flash sector that the specified address is in.

Parameters

<i>address</i>	Address in the flash sector.
----------------	------------------------------

Returns

Flash sector index in flashLayout[] or FLASH_INVALID_SECTOR_IDX.

Referenced by FlashErase(), FlashWrite(), and FlashWriteBlock().

7.404.2.7 FlashGetUserProgBaseAddress()

```
blt_addr FlashGetUserProgBaseAddress (  
    void )
```

Obtains the base address of the flash memory available to the user program. This is basically the first address in the flashLayout table.

Returns

Base address.

7.404.2.8 FlashInit()

```
void FlashInit (  
    void )
```

Initializes the flash driver.

Returns

none.

7.404.2.9 FlashInitBlock()

```
static blt_bool FlashInitBlock (  
    tFlashBlockInfo * block,  
    blt_addr address ) [static]
```

Copies data currently in flash to the block->data and sets the base address.

Parameters

<i>block</i>	Pointer to flash block info structure to operate on.
<i>address</i>	Base address of the block data.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

Referenced by FlashAddToBlock(), and FlashSwitchBlock().

7.404.2.10 FlashReinit()

```
void FlashReinit (
    void )
```

Reinitializes the flash driver.

Returns

none.

7.404.2.11 FlashSwitchBlock()

```
static tFlashBlockInfo * FlashSwitchBlock (
    tFlashBlockInfo * block,
    blt_addr base_addr ) [static]
```

Switches blocks by programming the current one and initializing the next.

Parameters

<i>block</i>	Pointer to flash block info structure to operate on.
<i>base_addr</i>	Base address of the next block.

Returns

The pointer of the block info struct that is now being used, or a NULL pointer in case of error.

Referenced by FlashAddToBlock().

7.404.2.12 FlashVerifyChecksum()

```
blt_bool FlashVerifyChecksum (
    void )
```

Verifies the checksum, which indicates that a valid user program is present and can be started.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.404.2.13 FlashWrite()

```
blt_bool FlashWrite (
    blt_addr addr,
    blt_int32u len,
    blt_int8u * data )
```

Writes the data to flash through a flash block manager. Note that this function also checks that no data is programmed outside the flash memory region, so the bootloader can never be overwritten.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.
<i>data</i>	Pointer to the data buffer.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

Referenced by FlashWriteChecksum().

7.404.2.14 FlashWriteBlock()

```
static blt_bool FlashWriteBlock (
    tFlashBlockInfo * block ) [static]
```

Programs FLASH_WRITE_BLOCK_SIZE bytes to flash from the block->data array.

Parameters

<i>block</i>	Pointer to flash block info structure to operate on.
--------------	--

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

Referenced by FlashDone(), and FlashSwitchBlock().

7.404.2.15 FlashWriteChecksum()

```
blt_bool FlashWriteChecksum (
    void )
```

Writes a checksum of the user program to non-volatile memory. This is performed once the entire user program has been programmed. Through the checksum, the bootloader can check if the programming session was completed, which indicates that a valid user programming is present and can be started.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.404.3 Variable Documentation

7.404.3.1 blockInfo

```
tFlashBlockInfo blockInfo [static]
```

Local variable with information about the flash block that is currently being operated on.

The smallest amount of flash that can be programmed is FLASH_WRITE_BLOCK_SIZE. A flash block manager is implemented in this driver and stores info in this variable. Whenever new data should be flashed, it is first added to a RAM buffer, which is part of this variable. Whenever the RAM buffer, which has the size of a flash block, is full or data needs to be written to a different block, the contents of the RAM buffer are programmed to flash. The flash block manager requires some software overhead, yet results in faster flash programming because data is first harvested, ideally until there is enough to program an entire flash block, before the flash device is actually operated on.

Referenced by FlashSwitchBlock().

7.404.3.2 bootBlockInfo

```
tFlashBlockInfo bootBlockInfo [static]
```

Local variable with information about the flash boot block.

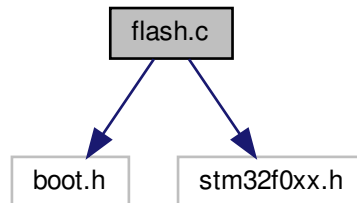
The first block of the user program holds the vector table, which on the STM32 is also the where the checksum is written to. It is likely that the vector table is first flashed and then, at the end of the programming sequence, the checksum. This means that this flash block needs to be written to twice. Normally this is not a problem with flash memory, as long as you write the same values to those bytes that are not supposed to be changed and the locations where you do write to are still in the erased 0xFF state. Unfortunately, writing twice to flash this way, does not work reliably on all micros. This is why we need to have an extra block, the bootblock, placed under the management of the block manager. This way it is possible to implement functionality so that the bootblock is only written to once at the end of the programming sequence.

Referenced by FlashSwitchBlock().

7.405 flash.c File Reference

Bootloader flash driver source file.

```
#include "boot.h"
#include "stm32f0xx.h"
Include dependency graph for ARMCM0_STM32F0/flash.c:
```



Data Structures

- struct [tFlashSector](#)
Flash sector descriptor type.
- struct [tFlashBlockInfo](#)
Structure type for grouping flash block information.

Macros

- #define [FLASH_INVALID_SECTOR](#) (0xff)
Value for an invalid flash sector.
- #define [FLASH_INVALID_ADDRESS](#) (0xffffffff)
Value for an invalid flash address.
- #define [FLASH_WRITE_BLOCK_SIZE](#) (512)
Standard size of a flash block for writing.
- #define [FLASH_TOTAL_SECTORS](#) (sizeof(flashLayout)/sizeof(flashLayout[0]))
Total numbers of sectors in array flashLayout[].
- #define [FLASH_END_ADDRESS](#)
End address of the bootloader programmable flash.
- #define [FLASH_ERASE_BLOCK_SIZE](#) (0x800)
Number of bytes to erase per erase operation.
- #define [BOOT_FLASH_VECTOR_TABLE_CS_OFFSET](#) (0xC0)
Offset into the user program's vector table where the checksum is located. For this target it is set to the end of the vector table. Note that the value can be overridden in blt_conf.h, because the size of the vector table could vary. When changing this value, don't forget to update the location of the checksum in the user program accordingly. Otherwise the checksum verification will always fail.

Functions

- static `blt_bool FlashInitBlock` (`tFlashBlockInfo *block`, `blt_addr` address)
Copies data currently in flash to the block->data and sets the base address.
- static `tFlashBlockInfo * FlashSwitchBlock` (`tFlashBlockInfo *block`, `blt_addr` base_addr)
Switches blocks by programming the current one and initializing the next.
- static `blt_bool FlashAddToBlock` (`tFlashBlockInfo *block`, `blt_addr` address, `blt_int8u *data`, `blt_int32u` len)
Programming is done per block. This function adds data to the block that is currently collecting data to be written to flash. If the address is outside of the current block, the current block is written to flash and a new block is initialized.
- static `blt_bool FlashWriteBlock` (`tFlashBlockInfo *block`)
Programs FLASH_WRITE_BLOCK_SIZE bytes to flash from the block->data array.
- static `blt_bool FlashEraseSectors` (`blt_int8u` first_sector, `blt_int8u` last_sector)
Erases the flash sectors from first_sector up until last_sector.
- static `blt_int8u FlashGetSector` (`blt_addr` address)
Determines the flash sector the address is in.
- static `blt_addr FlashGetSectorBaseAddr` (`blt_int8u` sector)
Determines the flash sector base address.
- static `blt_addr FlashGetSectorSize` (`blt_int8u` sector)
Determines the flash sector size.
- void `FlashInit` (void)
Initializes the flash driver.
- void `FlashReinit` (void)
Reinitializes the flash driver.
- `blt_bool FlashWrite` (`blt_addr` addr, `blt_int32u` len, `blt_int8u *data`)
Writes the data to flash through a flash block manager. Note that this function also checks that no data is programmed outside the flash memory region, so the bootloader can never be overwritten.
- `blt_bool FlashErase` (`blt_addr` addr, `blt_int32u` len)
Erases the flash memory. Note that this function also checks that no data is erased outside the flash memory region, so the bootloader can never be erased.
- `blt_bool FlashWriteChecksum` (void)
Writes a checksum of the user program to non-volatile memory. This is performed once the entire user program has been programmed. Through the checksum, the bootloader can check if the programming session was completed, which indicates that a valid user programming is present and can be started.
- `blt_bool FlashVerifyChecksum` (void)
Verifies the checksum, which indicates that a valid user program is present and can be started.
- `blt_bool FlashDone` (void)
Finalizes the flash driver operations. There could still be data in the currently active block that needs to be flashed.
- `blt_addr FlashGetUserProgBaseAddress` (void)
Obtains the base address of the flash memory available to the user program. This is basically the first address in the flashLayout table.

Variables

- static const `tFlashSector flashLayout` []
If desired, it is possible to set BOOT_FLASH_CUSTOM_LAYOUT_ENABLE to > 0 in blt_conf.h and then implement your own version of the flashLayout[] table in a source-file with the name flash_layout.c. This way you customize the flash memory size reserved for the bootloader, without having to modify the flashLayout[] table in this file directly. This file will then include flash_layout.c so there is no need to compile it additionally with your project.
- static `tFlashBlockInfo blockInfo`
Local variable with information about the flash block that is currently being operated on.
- static `tFlashBlockInfo bootBlockInfo`
Local variable with information about the flash boot block.

7.405.1 Detailed Description

Bootloader flash driver source file.

7.405.2 Function Documentation

7.405.2.1 FlashAddToBlock()

```
static blt_bool FlashAddToBlock (
    tFlashBlockInfo * block,
    blt_addr address,
    blt_int8u * data,
    blt_int32u len ) [static]
```

Programming is done per block. This function adds data to the block that is currently collecting data to be written to flash. If the address is outside of the current block, the current block is written to flash and a new block is initialized.

Parameters

<i>block</i>	Pointer to flash block info structure to operate on.
<i>address</i>	Flash destination address.
<i>data</i>	Pointer to the byte array with data.
<i>len</i>	Number of bytes to add to the block.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

Referenced by FlashWrite().

7.405.2.2 FlashDone()

```
blt_bool FlashDone (
    void )
```

Finalizes the flash driver operations. There could still be data in the currently active block that needs to be flashed.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.405.2.3 FlashErase()

```
blt_bool FlashErase (
    blt_addr addr,
    blt_int32u len )
```

Erases the flash memory. Note that this function also checks that no data is erased outside the flash memory region, so the bootloader can never be erased.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.405.2.4 FlashEraseSectors()

```
static blt_bool FlashEraseSectors (  
    blt_int8u first_sector,  
    blt_int8u last_sector ) [static]
```

Erases the flash sectors from first_sector up until last_sector.

Parameters

<i>first_sector</i>	First flash sector number.
<i>last_sector</i>	Last flash sector number.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

Referenced by FlashErase().

7.405.2.5 FlashGetSector()

```
static blt_int8u FlashGetSector (  
    blt_addr address ) [static]
```

Determines the flash sector the address is in.

Parameters

<i>address</i>	Address in the flash sector.
----------------	------------------------------

Returns

Flash sector number or FLASH_INVALID_SECTOR.

Referenced by FlashErase(), FlashWrite(), and FlashWriteBlock().

7.405.2.6 FlashGetSectorBaseAddr()

```
static blt_addr FlashGetSectorBaseAddr (  
    blt_int8u sector ) [static]
```

Determines the flash sector base address.

Parameters

<i>sector</i>	Sector to get the base address of.
---------------	------------------------------------

Returns

Flash sector base address or FLASH_INVALID_ADDRESS.

Referenced by FlashEraseSectors().

7.405.2.7 FlashGetSectorSize()

```
static blt_addr FlashGetSectorSize (  
    blt_int8u sector ) [static]
```

Determines the flash sector size.

Parameters

<i>sector</i>	Sector to get the size of.
---------------	----------------------------

Returns

Flash sector size or 0.

Referenced by FlashEraseSectors().

7.405.2.8 FlashGetUserProgBaseAddress()

```
blt_addr FlashGetUserProgBaseAddress (  
    void )
```

Obtains the base address of the flash memory available to the user program. This is basically the first address in the flashLayout table.

Returns

Base address.

7.405.2.9 FlashInit()

```
void FlashInit (
    void )
```

Initializes the flash driver.

Returns

none.

7.405.2.10 FlashInitBlock()

```
static blt_bool FlashInitBlock (
    tFlashBlockInfo * block,
    blt_addr address ) [static]
```

Copies data currently in flash to the block->data and sets the base address.

Parameters

<i>block</i>	Pointer to flash block info structure to operate on.
<i>address</i>	Base address of the block data.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

Referenced by FlashAddToBlock(), and FlashSwitchBlock().

7.405.2.11 FlashReinit()

```
void FlashReinit (
    void )
```

Reinitializes the flash driver.

Returns

none.

7.405.2.12 FlashSwitchBlock()

```
static tFlashBlockInfo * FlashSwitchBlock (
    tFlashBlockInfo * block,
    blt_addr base_addr ) [static]
```

Switches blocks by programming the current one and initializing the next.

Parameters

<i>block</i>	Pointer to flash block info structure to operate on.
<i>base_addr</i>	Base address of the next block.

Returns

The pointer of the block info struct that is no being used, or a NULL pointer in case of error.

Referenced by FlashAddToBlock().

7.405.2.13 FlashVerifyChecksum()

```
blt_bool FlashVerifyChecksum (
    void )
```

Verifies the checksum, which indicates that a valid user program is present and can be started.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.405.2.14 FlashWrite()

```
blt_bool FlashWrite (
    blt_addr addr,
    blt_int32u len,
    blt_int8u * data )
```

Writes the data to flash through a flash block manager. Note that this function also checks that no data is programmed outside the flash memory region, so the bootloader can never be overwritten.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.
<i>data</i>	Pointer to the data buffer.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

Referenced by FlashWriteChecksum().

7.405.2.15 FlashWriteBlock()

```
static blt_bool FlashWriteBlock (
    tFlashBlockInfo * block ) [static]
```

Programs FLASH_WRITE_BLOCK_SIZE bytes to flash from the block->data array.

Parameters

<i>block</i>	Pointer to flash block info structure to operate on.
--------------	--

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

Referenced by FlashDone(), and FlashSwitchBlock().

7.405.2.16 FlashWriteChecksum()

```
blt_bool FlashWriteChecksum (
    void )
```

Writes a checksum of the user program to non-volatile memory. This is performed once the entire user program has been programmed. Through the checksum, the bootloader can check if the programming session was completed, which indicates that a valid user programming is present and can be started.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.405.3 Variable Documentation

7.405.3.1 blockInfo

```
tFlashBlockInfo blockInfo [static]
```

Local variable with information about the flash block that is currently being operated on.

The smallest amount of flash that can be programmed is FLASH_WRITE_BLOCK_SIZE. A flash block manager is implemented in this driver and stores info in this variable. Whenever new data should be flashed, it is first added to a RAM buffer, which is part of this variable. Whenever the RAM buffer, which has the size of a flash block, is full or data needs to be written to a different block, the contents of the RAM buffer are programmed to flash. The flash block manager requires some software overhead, yet results in faster flash programming because data is first harvested, ideally until there is enough to program an entire flash block, before the flash device is actually operated on.

Referenced by FlashSwitchBlock().

7.405.3.2 bootBlockInfo

```
tFlashBlockInfo bootBlockInfo [static]
```

Local variable with information about the flash boot block.

The first block of the user program holds the vector table, which on the STM32 is also the where the checksum is written to. Is it likely that the vector table is first flashed and then, at the end of the programming sequence, the checksum. This means that this flash block need to be written to twice. Normally this is not a problem with flash memory, as long as you write the same values to those bytes that are not supposed to be changed and the locations where you do write to are still in the erased 0xFF state. Unfortunately, writing twice to flash this way, does not work reliably on all micros. This is why we need to have an extra block, the bootblock, placed under the management of the block manager. This way is it possible to implement functionality so that the bootblock is only written to once at the end of the programming sequence.

Referenced by FlashSwitchBlock().

7.405.3.3 flashLayout

```
const tFlashSector flashLayout[] [static]
```

If desired, it is possible to set `BOOT_FLASH_CUSTOM_LAYOUT_ENABLE` to `> 0` in `blt_conf.h` and then implement your own version of the `flashLayout[]` table in a source-file with the name `flash_layout.c`. This way you customize the flash memory size reserved for the bootloader, without having to modify the `flashLayout[]` table in this file directly. This file will then include `flash_layout.c` so there is no need to compile it additionally with your project.

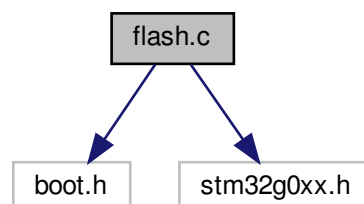
Array with the layout of the flash memory.

Also controls what part of the flash memory is reserved for the bootloader. If the bootloader size changes, the reserved sectors for the bootloader might need adjustment to make sure the bootloader doesn't get overwritten. The current flash layout does not reflect the minimum sector size of the physical flash (1 - 2kb), because this would make the table quit long and a waste of ROM. The minimum sector size is only really needed when erasing the flash. This can still be done in combination with macro `FLASH_ERASE_BLOCK_SIZE`. Note that the term sector here is used in a different meaning than in the controller's reference manual.

7.406 flash.c File Reference

Bootloader flash driver source file.

```
#include "boot.h"
#include "stm32g0xx.h"
Include dependency graph for ARMCM0_STM32G0/flash.c:
```



Data Structures

- struct [tFlashSector](#)
Flash sector descriptor type.
- struct [tFlashBlockInfo](#)
Structure type for grouping flash block information.

Macros

- #define [FLASH_INVALID_SECTOR](#) (0xff)
Value for an invalid sector number.
- #define [FLASH_INVALID_ADDRESS](#) (0xffffffff)
Value for an invalid flash address.
- #define [FLASH_WRITE_BLOCK_SIZE](#) (512)
Standard size of a flash block for writing.
- #define [FLASH_TOTAL_SECTORS](#) (sizeof(flashLayout)/sizeof(flashLayout[0]))
Total numbers of sectors in array flashLayout[].
- #define [FLASH_ADDRESS_MASK](#) (0x00ffffffu)
Mask for low-order bits of flash address range.
- #define [FLASH_START_ADDRESS](#) (flashLayout[0].sector_start & ~FLASH_ADDRESS_MASK)
Start address of the bootloader programmable flash.
- #define [FLASH_END_ADDRESS](#)
End address of the bootloader programmable flash.
- #define [FLASH_ERASE_BLOCK_SIZE](#) (0x800)
Hardware erase unit size (sectors must be multiples of this)
- #define [FLASH_ERASE_BLOCK_NUM](#)(adrs) (((adrs) - FLASH_START_ADDRESS) / FLASH_ERASE_BLOCK_SIZE)
Flash page (erase unit) number.
- #define [BOOT_FLASH_VECTOR_TABLE_CS_OFFSET](#) (0xBC)
Offset into the user program's vector table where the checksum is located. For this target it is set to the end of the vector table. Note that the value can be overridden in blt_conf.h, because the size of the vector table could vary. When changing this value, don't forget to update the location of the checksum in the user program accordingly. Otherwise the checksum verification will always fail.

Functions

- static [blt_bool](#) [FlashInitBlock](#) ([tFlashBlockInfo](#) *block, [blt_addr](#) address)
Copies data currently in flash to the block->data and sets the base address.
- static [tFlashBlockInfo](#) * [FlashSwitchBlock](#) ([tFlashBlockInfo](#) *block, [blt_addr](#) base_addr)
Switches blocks by programming the current one and initializing the next.
- static [blt_bool](#) [FlashAddToBlock](#) ([tFlashBlockInfo](#) *block, [blt_addr](#) address, [blt_int8u](#) *data, [blt_int32u](#) len)
Programming is done per block. This function adds data to the block that is currently collecting data to be written to flash. If the address is outside of the current block, the current block is written to flash and a new block is initialized.
- static [blt_bool](#) [FlashWriteBlock](#) ([tFlashBlockInfo](#) *block)
Programs FLASH_WRITE_BLOCK_SIZE bytes to flash from the block->data array.
- static [blt_bool](#) [FlashEraseSectors](#) ([blt_int8u](#) first_sector, [blt_int8u](#) last_sector)
Erases the flash sectors from first_sector up until last_sector.
- static [blt_int8u](#) [FlashGetSector](#) ([blt_addr](#) address)
Determines the flash sector the address is in.
- static [blt_addr](#) [FlashGetSectorBaseAddr](#) ([blt_int8u](#) sector)

- Determines the flash sector base address.*

 - static `blt_addr FlashGetSectorSize (blt_int8u sector)`

Determines the flash sector size.
 - void `FlashInit (void)`

Initializes the flash driver.
 - void `FlashReinit (void)`

Reinitializes the flash driver.
 - `blt_bool FlashWrite (blt_addr addr, blt_int32u len, blt_int8u *data)`

Writes the data to flash through a flash block manager. Note that this function also checks that no data is programmed outside the flash memory region, so the bootloader can never be overwritten.
 - `blt_bool FlashErase (blt_addr addr, blt_int32u len)`

Erases the flash memory. Note that this function also checks that no data is erased outside the flash memory region, so the bootloader can never be erased.
 - `blt_bool FlashWriteChecksum (void)`

Writes a checksum of the user program to non-volatile memory. This is performed once the entire user program has been programmed. Through the checksum, the bootloader can check if the programming session was completed, which indicates that a valid user programming is present and can be started.
 - `blt_bool FlashVerifyChecksum (void)`

Verifies the checksum, which indicates that a valid user program is present and can be started.
 - `blt_bool FlashDone (void)`

Finalizes the flash driver operations. There could still be data in the currently active block that needs to be flashed.
 - `blt_addr FlashGetUserProgBaseAddress (void)`

Obtains the base address of the flash memory available to the user program. This is basically the first address in the flashLayout table.

Variables

- static const `tFlashSector flashLayout []`

If desired, it is possible to set `BOOT_FLASH_CUSTOM_LAYOUT_ENABLE` to `> 0` in `blt_conf.h` and then implement your own version of the `flashLayout[]` table in a source-file with the name `flash_layout.c`. This way you customize the flash memory size reserved for the bootloader, without having to modify the `flashLayout[]` table in this file directly. This file will then include `flash_layout.c` so there is no need to compile it additionally with your project.
- static `tFlashBlockInfo blockInfo`

Local variable with information about the flash block that is currently being operated on.
- static `tFlashBlockInfo bootBlockInfo`

Local variable with information about the flash boot block.

7.406.1 Detailed Description

Bootloader flash driver source file.

7.406.2 Function Documentation

7.406.2.1 FlashAddToBlock()

```
static blt_bool FlashAddToBlock (
    tFlashBlockInfo * block,
    blt_addr address,
    blt_int8u * data,
    blt_int32u len ) [static]
```

Programming is done per block. This function adds data to the block that is currently collecting data to be written to flash. If the address is outside of the current block, the current block is written to flash and a new block is initialized.

Parameters

<i>block</i>	Pointer to flash block info structure to operate on.
<i>address</i>	Flash destination address.
<i>data</i>	Pointer to the byte array with data.
<i>len</i>	Number of bytes to add to the block.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

Referenced by FlashWrite().

7.406.2.2 FlashDone()

```
blt_bool FlashDone (  
    void )
```

Finalizes the flash driver operations. There could still be data in the currently active block that needs to be flashed.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.406.2.3 FlashErase()

```
blt_bool FlashErase (  
    blt_addr addr,  
    blt_int32u len )
```

Erases the flash memory. Note that this function also checks that no data is erased outside the flash memory region, so the bootloader can never be erased.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.406.2.4 FlashEraseSectors()

```
static blt_bool FlashEraseSectors (
    blt_int8u first_sector,
    blt_int8u last_sector ) [static]
```

Erases the flash sectors from `first_sector` up until `last_sector`.

Parameters

<i>first_sector</i>	First flash sector number.
<i>last_sector</i>	Last flash sector number.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

Referenced by FlashErase().

7.406.2.5 FlashGetSector()

```
static blt_int8u FlashGetSector (
    blt_addr address ) [static]
```

Determines the flash sector the address is in.

Parameters

<i>address</i>	Address in the flash sector.
----------------	------------------------------

Returns

Flash sector number or FLASH_INVALID_SECTOR.

Referenced by FlashErase(), FlashWrite(), and FlashWriteBlock().

7.406.2.6 FlashGetSectorBaseAddr()

```
static blt_addr FlashGetSectorBaseAddr (
    blt_int8u sector ) [static]
```

Determines the flash sector base address.

Parameters

<i>sector</i>	Sector to get the base address of.
---------------	------------------------------------

Returns

Flash sector base address or FLASH_INVALID_ADDRESS.

Referenced by FlashEraseSectors().

7.406.2.7 FlashGetSectorSize()

```
static blt_addr FlashGetSectorSize (  
    blt_int8u sector ) [static]
```

Determines the flash sector size.

Parameters

<i>sector</i>	Sector to get the size of.
---------------	----------------------------

Returns

Flash sector size or 0.

Referenced by FlashEraseSectors().

7.406.2.8 FlashGetUserProgBaseAddress()

```
blt_addr FlashGetUserProgBaseAddress (  
    void )
```

Obtains the base address of the flash memory available to the user program. This is basically the first address in the flashLayout table.

Returns

Base address.

7.406.2.9 FlashInit()

```
void FlashInit (  
    void )
```

Initializes the flash driver.

Returns

none.

7.406.2.10 FlashInitBlock()

```
static blt_bool FlashInitBlock (
    tFlashBlockInfo * block,
    blt_addr address ) [static]
```

Copies data currently in flash to the block->data and sets the base address.

Parameters

<i>block</i>	Pointer to flash block info structure to operate on.
<i>address</i>	Base address of the block data.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

Referenced by FlashAddToBlock(), and FlashSwitchBlock().

7.406.2.11 FlashReinit()

```
void FlashReinit (
    void )
```

Reinitializes the flash driver.

Returns

none.

7.406.2.12 FlashSwitchBlock()

```
static tFlashBlockInfo * FlashSwitchBlock (
    tFlashBlockInfo * block,
    blt_addr base_addr ) [static]
```

Switches blocks by programming the current one and initializing the next.

Parameters

<i>block</i>	Pointer to flash block info structure to operate on.
<i>base_addr</i>	Base address of the next block.

Returns

The pointer of the block info struct that is now being used, or a NULL pointer in case of error.

Referenced by FlashAddToBlock().

7.406.2.13 FlashVerifyChecksum()

```
blt_bool FlashVerifyChecksum (
    void )
```

Verifies the checksum, which indicates that a valid user program is present and can be started.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.406.2.14 FlashWrite()

```
blt_bool FlashWrite (
    blt_addr addr,
    blt_int32u len,
    blt_int8u * data )
```

Writes the data to flash through a flash block manager. Note that this function also checks that no data is programmed outside the flash memory region, so the bootloader can never be overwritten.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.
<i>data</i>	Pointer to the data buffer.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

Referenced by FlashWriteChecksum().

7.406.2.15 FlashWriteBlock()

```
static blt_bool FlashWriteBlock (
    tFlashBlockInfo * block ) [static]
```

Programs FLASH_WRITE_BLOCK_SIZE bytes to flash from the block->data array.

Parameters

<i>block</i>	Pointer to flash block info structure to operate on.
--------------	--

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

Referenced by FlashDone(), and FlashSwitchBlock().

7.406.2.16 FlashWriteChecksum()

```
blt_bool FlashWriteChecksum (  
    void )
```

Writes a checksum of the user program to non-volatile memory. This is performed once the entire user program has been programmed. Through the checksum, the bootloader can check if the programming session was completed, which indicates that a valid user programming is present and can be started.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.406.3 Variable Documentation

7.406.3.1 blockInfo

```
tFlashBlockInfo blockInfo [static]
```

Local variable with information about the flash block that is currently being operated on.

The smallest amount of flash that can be programmed is FLASH_WRITE_BLOCK_SIZE. A flash block manager is implemented in this driver and stores info in this variable. Whenever new data should be flashed, it is first added to a RAM buffer, which is part of this variable. Whenever the RAM buffer, which has the size of a flash block, is full or data needs to be written to a different block, the contents of the RAM buffer are programmed to flash. The flash block manager requires some software overhead, yet results in faster flash programming because data is first harvested, ideally until there is enough to program an entire flash block, before the flash device is actually operated on.

Referenced by FlashSwitchBlock().

7.406.3.2 bootBlockInfo

```
tFlashBlockInfo bootBlockInfo [static]
```

Local variable with information about the flash boot block.

The first block of the user program holds the vector table, which on the STM32 is also the where the checksum is written to. Is it likely that the vector table is first flashed and then, at the end of the programming sequence, the checksum. This means that this flash block need to be written to twice. Normally this is not a problem with flash memory, as long as you write the same values to those bytes that are not supposed to be changed and the locations where you do write to are still in the erased 0xFF state. Unfortunately, writing twice to flash this way, does not work reliably on all micros. This is why we need to have an extra block, the bootblock, placed under the management of the block manager. This way is it possible to implement functionality so that the bootblock is only written to once at the end of the programming sequence.

Referenced by FlashSwitchBlock().

7.406.3.3 flashLayout

```
const tFlashSector flashLayout[] [static]
```

If desired, it is possible to set `BOOT_FLASH_CUSTOM_LAYOUT_ENABLE` to `> 0` in `blt_conf.h` and then implement your own version of the `flashLayout[]` table in a source-file with the name `flash_layout.c`. This way you customize the flash memory size reserved for the bootloader, without having to modify the `flashLayout[]` table in this file directly. This file will then include `flash_layout.c` so there is no need to compile it additionally with your project.

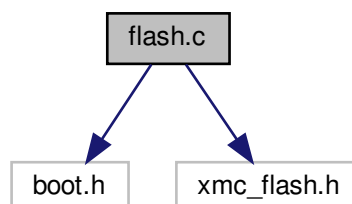
Array with the layout of the flash memory.

Also controls what part of the flash memory is reserved for the bootloader. If the bootloader size changes, the reserved sectors for the bootloader might need adjustment to make sure the bootloader doesn't get overwritten. The current flash layout does not reflect the minimum sector size of the physical flash (2 KiB), because this would make the table quite long and a waste of ROM. The minimum sector size is only really needed when erasing the flash. This can still be done in combination with macro `FLASH_ERASE_BLOCK_SIZE`. Note that the term sector here is used in a different meaning than in the controller's reference manual.

7.407 flash.c File Reference

Bootloader flash driver source file.

```
#include "boot.h"
#include "xmc_flash.h"
Include dependency graph for ARMCM0_XMC1/flash.c:
```



Data Structures

- struct [tFlashSector](#)
Flash sector descriptor type.
- struct [tFlashBlockInfo](#)
Structure type for grouping flash block information.

Macros

- #define [FLASH_INVALID_SECTOR](#) (0xff)
Value for an invalid flash sector.
- #define [FLASH_INVALID_ADDRESS](#) (0xffffffff)
Value for an invalid flash address.
- #define [FLASH_WRITE_BLOCK_SIZE](#) (256)
Standard size of a flash block for writing. It should be large enough so that the OpenBLT checksum fits in the first (boot) block). Note that this value is set to the hardware defined size of a XMC1 flash page.
- #define [FLASH_TOTAL_SECTORS](#) (sizeof(flashLayout)/sizeof(flashLayout[0]))
Total numbers of sectors in array flashLayout[].
- #define [FLASH_END_ADDRESS](#)
End address of the bootloader programmable flash.
- #define [BOOT_FLASH_VECTOR_TABLE_CS_OFFSET](#) (0x018)
Offset into the user program's vector table where the checksum is located. For this target it is set to the end of the vector table. Note that the value can be overridden in blt_conf.h, because the size of the vector table could vary. When changing this value, don't forget to update the location of the checksum in the user program accordingly. Otherwise the checksum verification will always fail.
- #define [FLASH_ERASE_TIME_MAX_MS](#) (10)
Maximum time for a sector erase operation as specified by the XCM1xxx data- sheet with an added 20% margin.
- #define [FLASH_PROGRAM_TIME_MAX_MS](#) (5)
Maximum time for a page program operation as specified by the XCM1xxx data- sheet with an added 20% margin.

Functions

- static [blt_bool](#) [FlashInitBlock](#) ([tFlashBlockInfo](#) *block, [blt_addr](#) address)
Copies data currently in flash to the block->data and sets the base address.
- static [tFlashBlockInfo](#) * [FlashSwitchBlock](#) ([tFlashBlockInfo](#) *block, [blt_addr](#) base_addr)
Switches blocks by programming the current one and initializing the next.
- static [blt_bool](#) [FlashAddToBlock](#) ([tFlashBlockInfo](#) *block, [blt_addr](#) address, [blt_int8u](#) *data, [blt_int32u](#) len)
Programming is done per block. This function adds data to the block that is currently collecting data to be written to flash. If the address is outside of the current block, the current block is written to flash and a new block is initialized.
- static [blt_bool](#) [FlashWriteBlock](#) ([tFlashBlockInfo](#) *block)
Programs FLASH_WRITE_BLOCK_SIZE bytes to flash from the block->data array.
- static [blt_bool](#) [FlashEraseSectors](#) ([blt_int8u](#) first_sector, [blt_int8u](#) last_sector)
Erases the flash sectors from first_sector up until last_sector.
- static [blt_int8u](#) [FlashGetSector](#) ([blt_addr](#) address)
Determines the flash sector the address is in.
- static [blt_addr](#) [FlashGetSectorBaseAddr](#) ([blt_int8u](#) sector)
Obtains the base address of the specified sector.
- void [FlashInit](#) (void)
Initializes the flash driver.
- void [FlashReinit](#) (void)

Reinitializes the flash driver.

- `blt_bool FlashWrite (blt_addr addr, blt_int32u len, blt_int8u *data)`
Writes the data to flash through a flash block manager. Note that this function also checks that no data is programmed outside the flash memory region, so the bootloader can never be overwritten.
- `blt_bool FlashErase (blt_addr addr, blt_int32u len)`
Erases the flash memory. Note that this function also checks that no data is erased outside the flash memory region, so the bootloader can never be erased.
- `blt_bool FlashWriteChecksum (void)`
Writes a checksum of the user program to non-volatile memory. This is performed once the entire user program has been programmed. Through the checksum, the bootloader can check if the programming session was completed, which indicates that a valid user programming is present and can be started.
- `blt_bool FlashVerifyChecksum (void)`
Verifies the checksum, which indicates that a valid user program is present and can be started.
- `blt_bool FlashDone (void)`
Finalizes the flash driver operations. There could still be data in the currently active block that needs to be flashed.
- `blt_addr FlashGetUserProgBaseAddress (void)`
Obtains the base address of the flash memory available to the user program. This is basically the first address in the flashLayout table.

Variables

- static const `tFlashSector flashLayout []`
If desired, it is possible to set `BOOT_FLASH_CUSTOM_LAYOUT_ENABLE` to `> 0` in `blt_conf.h` and then implement your own version of the `flashLayout[]` table in a source-file with the name `flash_layout.c`. This way you customize the flash memory size reserved for the bootloader, without having to modify the `flashLayout[]` table in this file directly. This file will then include `flash_layout.c` so there is no need to compile it additionally with your project.
- static `tFlashBlockInfo blockInfo`
Local variable with information about the flash block that is currently being operated on.
- static `tFlashBlockInfo bootBlockInfo`
Local variable with information about the flash boot block.

7.407.1 Detailed Description

Bootloader flash driver source file.

7.407.2 Function Documentation

7.407.2.1 FlashAddToBlock()

```
static blt_bool FlashAddToBlock (
    tFlashBlockInfo * block,
    blt_addr address,
    blt_int8u * data,
    blt_int32u len ) [static]
```

Programming is done per block. This function adds data to the block that is currently collecting data to be written to flash. If the address is outside of the current block, the current block is written to flash and a new block is initialized.

Parameters

<i>block</i>	Pointer to flash block info structure to operate on.
<i>address</i>	Flash destination address.
<i>data</i>	Pointer to the byte array with data.
<i>len</i>	Number of bytes to add to the block.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

Referenced by FlashWrite().

7.407.2.2 FlashDone()

```
blt_bool FlashDone (  
    void )
```

Finalizes the flash driver operations. There could still be data in the currently active block that needs to be flashed.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.407.2.3 FlashErase()

```
blt_bool FlashErase (  
    blt_addr addr,  
    blt_int32u len )
```

Erases the flash memory. Note that this function also checks that no data is erased outside the flash memory region, so the bootloader can never be erased.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.407.2.4 FlashEraseSectors()

```
static blt_bool FlashEraseSectors (
    blt_int8u first_sector,
    blt_int8u last_sector ) [static]
```

Erases the flash sectors from first_sector up until last_sector.

Parameters

<i>first_sector</i>	First flash sector number.
<i>last_sector</i>	Last flash sector number.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

Referenced by FlashErase().

7.407.2.5 FlashGetSector()

```
static blt_int8u FlashGetSector (
    blt_addr address ) [static]
```

Determines the flash sector the address is in.

Parameters

<i>address</i>	Address in the flash sector.
----------------	------------------------------

Returns

Flash sector number or FLASH_INVALID_SECTOR.

Referenced by FlashErase(), FlashWrite(), and FlashWriteBlock().

7.407.2.6 FlashGetSectorBaseAddr()

```
static blt_addr FlashGetSectorBaseAddr (
    blt_int8u sector ) [static]
```

Obtains the base address of the specified sector.

Parameters

<i>sector</i>	Sector to get the base address of.
---------------	------------------------------------

Returns

Base Base address of the sector if found, FLASH_INVALID_ADDRESS otherwise.

Referenced by FlashEraseSectors().

7.407.2.7 FlashGetUserProgBaseAddress()

```
b1t_addr FlashGetUserProgBaseAddress (
    void )
```

Obtains the base address of the flash memory available to the user program. This is basically the first address in the flashLayout table.

Returns

Base address.

7.407.2.8 FlashInit()

```
void FlashInit (
    void )
```

Initializes the flash driver.

Returns

none.

7.407.2.9 FlashInitBlock()

```
static b1t_bool FlashInitBlock (
    tFlashBlockInfo * block,
    b1t_addr address ) [static]
```

Copies data currently in flash to the block->data and sets the base address.

Parameters

<i>block</i>	Pointer to flash block info structure to operate on.
<i>address</i>	Base address of the block data.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

Referenced by FlashAddToBlock(), and FlashSwitchBlock().

7.407.2.10 FlashReinit()

```
void FlashReinit (
    void )
```

Reinitializes the flash driver.

Returns

none.

7.407.2.11 FlashSwitchBlock()

```
static tFlashBlockInfo * FlashSwitchBlock (
    tFlashBlockInfo * block,
    blt_addr base_addr ) [static]
```

Switches blocks by programming the current one and initializing the next.

Parameters

<i>block</i>	Pointer to flash block info structure to operate on.
<i>base_addr</i>	Base address of the next block.

Returns

The pointer of the block info struct that is no being used, or a NULL pointer in case of error.

Referenced by FlashAddToBlock().

7.407.2.12 FlashVerifyChecksum()

```
blt_bool FlashVerifyChecksum (
    void )
```

Verifies the checksum, which indicates that a valid user program is present and can be started.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.407.2.13 FlashWrite()

```
blt_bool FlashWrite (
    blt_addr addr,
    blt_int32u len,
    blt_int8u * data )
```

Writes the data to flash through a flash block manager. Note that this function also checks that no data is programmed outside the flash memory region, so the bootloader can never be overwritten.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.
<i>data</i>	Pointer to the data buffer.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

Referenced by FlashWriteChecksum().

7.407.2.14 FlashWriteBlock()

```
static blt_bool FlashWriteBlock (
    tFlashBlockInfo * block ) [static]
```

Programs FLASH_WRITE_BLOCK_SIZE bytes to flash from the block->data array.

Parameters

<i>block</i>	Pointer to flash block info structure to operate on.
--------------	--

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

Referenced by FlashDone(), and FlashSwitchBlock().

7.407.2.15 FlashWriteChecksum()

```
blt_bool FlashWriteChecksum (
    void )
```

Writes a checksum of the user program to non-volatile memory. This is performed once the entire user program has been programmed. Through the checksum, the bootloader can check if the programming session was completed, which indicates that a valid user programming is present and can be started.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.407.3 Variable Documentation

7.407.3.1 blockInfo

```
tFlashBlockInfo blockInfo [static]
```

Local variable with information about the flash block that is currently being operated on.

The smallest amount of flash that can be programmed is FLASH_WRITE_BLOCK_SIZE. A flash block manager is implemented in this driver and stores info in this variable. Whenever new data should be flashed, it is first added to a RAM buffer, which is part of this variable. Whenever the RAM buffer, which has the size of a flash block, is full or data needs to be written to a different block, the contents of the RAM buffer are programmed to flash. The flash block manager requires some software overhead, yet results in faster flash programming because data is first harvested, ideally until there is enough to program an entire flash block, before the flash device is actually operated on.

Referenced by FlashSwitchBlock().

7.407.3.2 bootBlockInfo

```
tFlashBlockInfo bootBlockInfo [static]
```

Local variable with information about the flash boot block.

The first block of the user program holds the vector table, which on the STM32 is also the where the checksum is written to. It is likely that the vector table is first flashed and then, at the end of the programming sequence, the checksum. This means that this flash block needs to be written to twice. Normally this is not a problem with flash memory, as long as you write the same values to those bytes that are not supposed to be changed and the locations where you do write to are still in the erased 0xFF state. Unfortunately, writing twice to flash this way, does not work reliably on all micros. This is why we need to have an extra block, the bootblock, placed under the management of the block manager. This way it is possible to implement functionality so that the bootblock is only written to once at the end of the programming sequence.

Referenced by FlashSwitchBlock().

7.407.3.3 flashLayout

```
const tFlashSector flashLayout[] [static]
```

If desired, it is possible to set BOOT_FLASH_CUSTOM_LAYOUT_ENABLE to > 0 in blt_conf.h and then implement your own version of the flashLayout[] table in a source-file with the name flash_layout.c. This way you customize the flash memory size reserved for the bootloader, without having to modify the flashLayout[] table in this file directly. This file will then include flash_layout.c so there is no need to compile it additionally with your project.

Array with the layout of the flash memory.

Also controls what part of the flash memory is reserved for the bootloader. If the bootloader size changes, the reserved sectors for the bootloader might need adjustment to make sure the bootloader doesn't get overwritten. Note that the table contains uncached addresses, because flash program/erase operations need to be performed on uncached addresses. This flash driver automatically translated cached to uncached addresses, so there is no need for the user to adjust this when calling this driver's API.

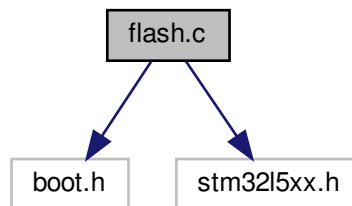
7.408 flash.c File Reference

Bootloader flash driver source file.

```
#include "boot.h"
```

```
#include "stm32l5xx.h"
```

Include dependency graph for ARMCM33_STM32L5/flash.c:



Data Structures

- struct [tFlashSector](#)
Flash sector descriptor type.
- struct [tFlashBlockInfo](#)
Structure type for grouping flash block information.

Macros

- #define [FLASH_INVALID_SECTOR_IDX](#) (0xff)
Value for an invalid sector entry index into flashLayout[].
- #define [FLASH_INVALID_ADDRESS](#) (0xffffffff)
Value for an invalid flash address.
- #define [FLASH_WRITE_BLOCK_SIZE](#) (512)
Standard size of a flash block for writing.
- #define [FLASH_TOTAL_SECTORS](#) (sizeof(flashLayout)/sizeof(flashLayout[0]))
Total numbers of sectors in array flashLayout[].
- #define [FLASH_END_ADDRESS](#)
End address of the bootloader programmable flash.
- #define [BOOT_FLASH_VECTOR_TABLE_CS_OFFSET](#) (0x1F4)
Offset into the user program's vector table where the checksum is located. For this target it is set to the end of the vector table. Note that the value can be overridden in blt_conf.h, because the size of the vector table could vary. When changing this value, don't forget to update the location of the checksum in the user program accordingly. Otherwise the checksum verification will always fail.

Functions

- static `blt_bool FlashInitBlock` (`tFlashBlockInfo *block`, `blt_addr` address)
Copies data currently in flash to the block->data and sets the base address.
- static `tFlashBlockInfo * FlashSwitchBlock` (`tFlashBlockInfo *block`, `blt_addr` base_addr)
Switches blocks by programming the current one and initializing the next.
- static `blt_bool FlashAddToBlock` (`tFlashBlockInfo *block`, `blt_addr` address, `blt_int8u *data`, `blt_int32u` len)
Programming is done per block. This function adds data to the block that is currently collecting data to be written to flash. If the address is outside of the current block, the current block is written to flash and a new block is initialized.
- static `blt_bool FlashWriteBlock` (`tFlashBlockInfo *block`)
Programs FLASH_WRITE_BLOCK_SIZE bytes to flash from the block->data array.
- static `blt_bool FlashEraseSectors` (`blt_int8u` first_sector_idx, `blt_int8u` last_sector_idx)
Erases the flash sectors from indices first_sector_idx up until last_sector_idx into the flashLayout[] array.
- static `blt_int8u FlashGetSectorIdx` (`blt_addr` address)
Determines the index into the flashLayout[] array of the flash sector that the specified address is in.
- static `blt_int32u FlashGetPageSize` (void)
Determines the size of a flash pages, defined by hardware. This also defines the minimal erase size.
- static `blt_bool FlashIsDualBankMode` (void)
Determines if the flash device is configured as dual bank mode or single bank mode.
- static `blt_int32u FlashGetBank` (`blt_addr` address)
Determines the flash bank that the address belongs to.
- static `blt_int32u FlashGetPage` (`blt_addr` address)
Determines the flash page that the address belongs to.
- void `FlashInit` (void)
Initializes the flash driver.
- void `FlashReinit` (void)
Reinitializes the flash driver.
- `blt_bool FlashWrite` (`blt_addr` addr, `blt_int32u` len, `blt_int8u *data`)
Writes the data to flash through a flash block manager. Note that this function also checks that no data is programmed outside the flash memory region, so the bootloader can never be overwritten.
- `blt_bool FlashErase` (`blt_addr` addr, `blt_int32u` len)
Erases the flash memory. Note that this function also checks that no data is erased outside the flash memory region, so the bootloader can never be erased.
- `blt_bool FlashWriteChecksum` (void)
Writes a checksum of the user program to non-volatile memory. This is performed once the entire user program has been programmed. Through the checksum, the bootloader can check if the programming session was completed, which indicates that a valid user programming is present and can be started.
- `blt_bool FlashVerifyChecksum` (void)
Verifies the checksum, which indicates that a valid user program is present and can be started.
- `blt_bool FlashDone` (void)
Finalizes the flash driver operations. There could still be data in the currently active block that needs to be flashed.
- `blt_addr FlashGetUserProgBaseAddress` (void)
Obtains the base address of the flash memory available to the user program. This is basically the first address in the flashLayout table.

Variables

- static const `tFlashSector flashLayout` []
If desired, it is possible to set BOOT_FLASH_CUSTOM_LAYOUT_ENABLE to > 0 in blt_conf.h and then implement your own version of the flashLayout[] table in a source-file with the name flash_layout.c. This way you customize the flash memory size reserved for the bootloader, without having to modify the flashLayout[] table in this file directly. This file will then include flash_layout.c so there is no need to compile it additionally with your project.
- static `tFlashBlockInfo blockInfo`
Local variable with information about the flash block that is currently being operated on.
- static `tFlashBlockInfo bootBlockInfo`
Local variable with information about the flash boot block.

7.408.1 Detailed Description

Bootloader flash driver source file.

7.408.2 Function Documentation

7.408.2.1 FlashAddToBlock()

```
static blt_bool FlashAddToBlock (
    tFlashBlockInfo * block,
    blt_addr address,
    blt_int8u * data,
    blt_int32u len ) [static]
```

Programming is done per block. This function adds data to the block that is currently collecting data to be written to flash. If the address is outside of the current block, the current block is written to flash and a new block is initialized.

Parameters

<i>block</i>	Pointer to flash block info structure to operate on.
<i>address</i>	Flash destination address.
<i>data</i>	Pointer to the byte array with data.
<i>len</i>	Number of bytes to add to the block.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

Referenced by FlashWrite().

7.408.2.2 FlashDone()

```
blt_bool FlashDone (
    void )
```

Finalizes the flash driver operations. There could still be data in the currently active block that needs to be flashed.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.408.2.3 FlashErase()

```
blt_bool FlashErase (
    blt_addr addr,
    blt_int32u len )
```

Erases the flash memory. Note that this function also checks that no data is erased outside the flash memory region, so the bootloader can never be erased.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.408.2.4 FlashEraseSectors()

```
static blt_bool FlashEraseSectors (
    blt_int8u first_sector_idx,
    blt_int8u last_sector_idx ) [static]
```

Erases the flash sectors from indices first_sector_idx up until last_sector_idx into the flashLayout[] array.

Parameters

<i>first_sector_idx</i>	First flash sector number index into flashLayout[].
<i>last_sector_idx</i>	Last flash sector number index into flashLayout[].

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

Referenced by FlashErase().

7.408.2.5 FlashGetBank()

```
static blt_int32u FlashGetBank (
    blt_addr address ) [static]
```

Determines the flash bank that the address belongs to.

Parameters

<i>address</i>	Flash memory address.
----------------	-----------------------

Returns

FLASH_BANK_1 if the address belongs to bank 1, FLASH_BANK_2 otherwise.

Referenced by FlashEraseSectors(), and FlashGetPage().

7.408.2.6 FlashGetPage()

```
static blt_int32u FlashGetPage (
    blt_addr address ) [static]
```

Determines the flash page that the address belongs to.

end of FlashGetBank

Parameters

<i>address</i>	Flash memory address.
----------------	-----------------------

Returns

Page number.

Referenced by FlashEraseSectors().

7.408.2.7 FlashGetPageSize()

```
static blt_int32u FlashGetPageSize (
    void ) [static]
```

Determines the size of a flash pages, defined by hardware. This also defines the minimal erase size.

Returns

Size of a flash page.

Referenced by FlashEraseSectors(), and FlashGetPage().

7.408.2.8 FlashGetSectorIdx()

```
static blt_int8u FlashGetSectorIdx (
    blt_addr address ) [static]
```

Determines the index into the flashLayout[] array of the flash sector that the specified address is in.

Parameters

<i>address</i>	Address in the flash sector.
----------------	------------------------------

Returns

Flash sector index in flashLayout[] or FLASH_INVALID_SECTOR_IDX.

Referenced by FlashErase(), FlashWrite(), and FlashWriteBlock().

7.408.2.9 FlashGetUserProgBaseAddress()

```
blt_addr FlashGetUserProgBaseAddress (
    void )
```

Obtains the base address of the flash memory available to the user program. This is basically the first address in the flashLayout table.

Returns

Base address.

7.408.2.10 FlashInit()

```
void FlashInit (
    void )
```

Initializes the flash driver.

Returns

none.

7.408.2.11 FlashInitBlock()

```
static blt_bool FlashInitBlock (
    tFlashBlockInfo * block,
    blt_addr address ) [static]
```

Copies data currently in flash to the block->data and sets the base address.

Parameters

<i>block</i>	Pointer to flash block info structure to operate on.
<i>address</i>	Base address of the block data.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

Referenced by FlashAddToBlock(), and FlashSwitchBlock().

7.408.2.12 FlashIsDualBankMode()

```
static blt_bool FlashIsDualBankMode (  
    void ) [static]
```

Determines if the flash device is configured as dual bank mode or single bank mode.

Returns

BLT_TRUE if configured as dual bank mode, BLT_FALSE for single bank mode.

Referenced by FlashGetBank(), FlashGetPage(), and FlashGetPageSize().

7.408.2.13 FlashReinit()

```
void FlashReinit (  
    void )
```

Reinitializes the flash driver.

Returns

none.

7.408.2.14 FlashSwitchBlock()

```
static tFlashBlockInfo * FlashSwitchBlock (  
    tFlashBlockInfo * block,  
    blt_addr base_addr ) [static]
```

Switches blocks by programming the current one and initializing the next.

Parameters

<i>block</i>	Pointer to flash block info structure to operate on.
<i>base_addr</i>	Base address of the next block.

Returns

The pointer of the block info struct that is now being used, or a NULL pointer in case of error.

Referenced by FlashAddToBlock().

7.408.2.15 FlashVerifyChecksum()

```
blt_bool FlashVerifyChecksum (
    void )
```

Verifies the checksum, which indicates that a valid user program is present and can be started.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.408.2.16 FlashWrite()

```
blt_bool FlashWrite (
    blt_addr addr,
    blt_int32u len,
    blt_int8u * data )
```

Writes the data to flash through a flash block manager. Note that this function also checks that no data is programmed outside the flash memory region, so the bootloader can never be overwritten.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.
<i>data</i>	Pointer to the data buffer.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

Referenced by FlashWriteChecksum().

7.408.2.17 FlashWriteBlock()

```
static blt_bool FlashWriteBlock (
    tFlashBlockInfo * block ) [static]
```

Programs FLASH_WRITE_BLOCK_SIZE bytes to flash from the block->data array.

Parameters

<i>block</i>	Pointer to flash block info structure to operate on.
--------------	--

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

Referenced by FlashDone(), and FlashSwitchBlock().

7.408.2.18 FlashWriteChecksum()

```
blt_bool FlashWriteChecksum (  
    void )
```

Writes a checksum of the user program to non-volatile memory. This is performed once the entire user program has been programmed. Through the checksum, the bootloader can check if the programming session was completed, which indicates that a valid user programming is present and can be started.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.408.3 Variable Documentation

7.408.3.1 blockInfo

```
tFlashBlockInfo blockInfo [static]
```

Local variable with information about the flash block that is currently being operated on.

The smallest amount of flash that can be programmed is FLASH_WRITE_BLOCK_SIZE. A flash block manager is implemented in this driver and stores info in this variable. Whenever new data should be flashed, it is first added to a RAM buffer, which is part of this variable. Whenever the RAM buffer, which has the size of a flash block, is full or data needs to be written to a different block, the contents of the RAM buffer are programmed to flash. The flash block manager requires some software overhead, yet results in faster flash programming because data is first harvested, ideally until there is enough to program an entire flash block, before the flash device is actually operated on.

Referenced by FlashSwitchBlock().

7.408.3.2 bootBlockInfo

```
tFlashBlockInfo bootBlockInfo [static]
```

Local variable with information about the flash boot block.

The first block of the user program holds the vector table, which on the STM32 is also the where the checksum is written to. Is it likely that the vector table is first flashed and then, at the end of the programming sequence, the checksum. This means that this flash block need to be written to twice. Normally this is not a problem with flash memory, as long as you write the same values to those bytes that are not supposed to be changed and the locations where you do write to are still in the erased 0xFF state. Unfortunately, writing twice to flash this way, does not work reliably on all micros. This is why we need to have an extra block, the bootblock, placed under the management of the block manager. This way is it possible to implement functionality so that the bootblock is only written to once at the end of the programming sequence.

Referenced by FlashSwitchBlock().

7.408.3.3 flashLayout

```
const tFlashSector flashLayout[] [static]
```

If desired, it is possible to set `BOOT_FLASH_CUSTOM_LAYOUT_ENABLE` to `> 0` in `blt_conf.h` and then implement your own version of the `flashLayout[]` table in a source-file with the name `flash_layout.c`. This way you customize the flash memory size reserved for the bootloader, without having to modify the `flashLayout[]` table in this file directly. This file will then include `flash_layout.c` so there is no need to compile it additionally with your project.

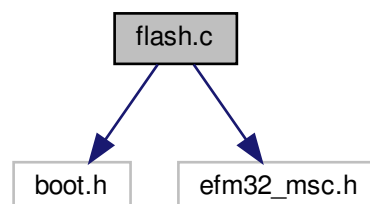
Array wit the layout of the flash memory.

Also controls what part of the flash memory is reserved for the bootloader. If the bootloader size changes, the reserved sectors for the bootloader might need adjustment to make sure the bootloader doesn't get overwritten.

7.409 flash.c File Reference

Bootloader flash driver source file.

```
#include "boot.h"  
#include "efm32_msc.h"  
Include dependency graph for ARMCM3_EFM32/flash.c:
```



Data Structures

- struct [tFlashSector](#)
Flash sector descriptor type.
- struct [tFlashBlockInfo](#)
Structure type for grouping flash block information.

Macros

- #define [FLASH_INVALID_SECTOR](#) (0xff)
Value for an invalid flash sector.
- #define [FLASH_INVALID_ADDRESS](#) (0xffffffff)
Value for an invalid flash address.
- #define [FLASH_WRITE_BLOCK_SIZE](#) (512)
Standard size of a flash block for writing.
- #define [FLASH_TOTAL_SECTORS](#) (sizeof(flashLayout)/sizeof(flashLayout[0]))
Total numbers of sectors in array flashLayout[].
- #define [FLASH_END_ADDRESS](#)
End address of the bootloader programmable flash.
- #define [BOOT_FLASH_VECTOR_TABLE_CS_OFFSET](#) (0x0B8)
Offset into the user program's vector table where the checksum is located. For this target it is set to the end of the vector table. Note that the value can be overridden in blt_conf.h, because the size of the vector table could vary. When changing this value, don't forget to update the location of the checksum in the user program accordingly. Otherwise the checksum verification will always fail.

Functions

- static [blt_bool](#) [FlashInitBlock](#) ([tFlashBlockInfo](#) *block, [blt_addr](#) address)
Copies data currently in flash to the block->data and sets the base address.
- static [tFlashBlockInfo](#) * [FlashSwitchBlock](#) ([tFlashBlockInfo](#) *block, [blt_addr](#) base_addr)
Switches blocks by programming the current one and initializing the next.
- static [blt_bool](#) [FlashAddToBlock](#) ([tFlashBlockInfo](#) *block, [blt_addr](#) address, [blt_int8u](#) *data, [blt_int32u](#) len)
Programming is done per block. This function adds data to the block that is currently collecting data to be written to flash. If the address is outside of the current block, the current block is written to flash and a new block is initialized.
- static [blt_bool](#) [FlashWriteBlock](#) ([tFlashBlockInfo](#) *block)
Programs FLASH_WRITE_BLOCK_SIZE bytes to flash from the block->data array.
- static [blt_bool](#) [FlashEraseSectors](#) ([blt_int8u](#) first_sector, [blt_int8u](#) last_sector)
Erases the flash sectors from first_sector up until last_sector.
- static [blt_int8u](#) [FlashGetSector](#) ([blt_addr](#) address)
Determines the flash sector the address is in.
- static [blt_addr](#) [FlashGetSectorBaseAddr](#) ([blt_int8u](#) sector)
Determines the flash sector base address.
- static [blt_addr](#) [FlashGetSectorSize](#) ([blt_int8u](#) sector)
Determines the flash sector size.
- static [blt_int32u](#) [FlashCalcPageSize](#) (void)
Determines the flash page size for the specific EFM32 derivative. This is the minimum erase size.
- void [FlashInit](#) (void)
Initializes the flash driver.
- void [FlashReinit](#) (void)
Reinitializes the flash driver.

- `blt_bool FlashWrite (blt_addr addr, blt_int32u len, blt_int8u *data)`
Writes the data to flash through a flash block manager. Note that this function also checks that no data is programmed outside the flash memory region, so the bootloader can never be overwritten.
- `blt_bool FlashErase (blt_addr addr, blt_int32u len)`
Erases the flash memory. Note that this function also checks that no data is erased outside the flash memory region, so the bootloader can never be erased.
- `blt_bool FlashWriteChecksum (void)`
Writes a checksum of the user program to non-volatile memory. This is performed once the entire user program has been programmed. Through the checksum, the bootloader can check if the programming session was completed, which indicates that a valid user programming is present and can be started.
- `blt_bool FlashVerifyChecksum (void)`
Verifies the checksum, which indicates that a valid user program is present and can be started.
- `blt_bool FlashDone (void)`
Finalizes the flash driver operations. There could still be data in the currently active block that needs to be flashed.
- `blt_addr FlashGetUserProgBaseAddress (void)`
Obtains the base address of the flash memory available to the user program. This is basically the first address in the flashLayout table.

Variables

- static const `tFlashSector flashLayout []`
If desired, it is possible to set `BOOT_FLASH_CUSTOM_LAYOUT_ENABLE` to `> 0` in `blt_conf.h` and then implement your own version of the `flashLayout[]` table in a source-file with the name `flash_layout.c`. This way you customize the flash memory size reserved for the bootloader, without having to modify the `flashLayout[]` table in this file directly. This file will then include `flash_layout.c` so there is no need to compile it additionally with your project.
- static `tFlashBlockInfo blockInfo`
Local variable with information about the flash block that is currently being operated on.
- static `tFlashBlockInfo bootBlockInfo`
Local variable with information about the flash boot block.

7.409.1 Detailed Description

Bootloader flash driver source file.

7.409.2 Function Documentation

7.409.2.1 FlashAddToBlock()

```
static blt_bool FlashAddToBlock (
    tFlashBlockInfo * block,
    blt_addr address,
    blt_int8u * data,
    blt_int32u len ) [static]
```

Programming is done per block. This function adds data to the block that is currently collecting data to be written to flash. If the address is outside of the current block, the current block is written to flash and a new block is initialized.

Parameters

<i>block</i>	Pointer to flash block info structure to operate on.
<i>address</i>	Flash destination address.
<i>data</i>	Pointer to the byte array with data.
<i>len</i>	Number of bytes to add to the block.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

Referenced by FlashWrite().

7.409.2.2 FlashCalcPageSize()

```
static blt_int32u FlashCalcPageSize (  
    void ) [static]
```

Determines the flash page size for the specific EFM32 derivative. This is the minimum erase size.

Returns

The flash page size.

Referenced by FlashEraseSectors().

7.409.2.3 FlashDone()

```
blt_bool FlashDone (  
    void )
```

Finalizes the flash driver operations. There could still be data in the currently active block that needs to be flashed.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.409.2.4 FlashErase()

```
blt_bool FlashErase (  
    blt_addr addr,  
    blt_int32u len )
```

Erases the flash memory. Note that this function also checks that no data is erased outside the flash memory region, so the bootloader can never be erased.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.409.2.5 FlashEraseSectors()

```
static blt_bool FlashEraseSectors (  
    blt_int8u first_sector,  
    blt_int8u last_sector ) [static]
```

Erases the flash sectors from first_sector up until last_sector.

Parameters

<i>first_sector</i>	First flash sector number.
<i>last_sector</i>	Last flash sector number.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

Referenced by FlashErase().

7.409.2.6 FlashGetSector()

```
static blt_int8u FlashGetSector (  
    blt_addr address ) [static]
```

Determines the flash sector the address is in.

Parameters

<i>address</i>	Address in the flash sector.
----------------	------------------------------

Returns

Flash sector number or FLASH_INVALID_SECTOR.

Referenced by FlashErase(), FlashWrite(), and FlashWriteBlock().

7.409.2.7 FlashGetSectorBaseAddr()

```
static blt_addr FlashGetSectorBaseAddr (
    blt_int8u sector ) [static]
```

Determines the flash sector base address.

Parameters

<i>sector</i>	Sector to get the base address of.
---------------	------------------------------------

Returns

Flash sector base address or FLASH_INVALID_ADDRESS.

Referenced by FlashEraseSectors().

7.409.2.8 FlashGetSectorSize()

```
static blt_addr FlashGetSectorSize (
    blt_int8u sector ) [static]
```

Determines the flash sector size.

Parameters

<i>sector</i>	Sector to get the size of.
---------------	----------------------------

Returns

Flash sector size or 0.

Referenced by FlashEraseSectors().

7.409.2.9 FlashGetUserProgBaseAddress()

```
blt_addr FlashGetUserProgBaseAddress (
    void )
```

Obtains the base address of the flash memory available to the user program. This is basically the first address in the flashLayout table.

Returns

Base address.

7.409.2.10 FlashInit()

```
void FlashInit (
    void )
```

Initializes the flash driver.

Returns

none.

7.409.2.11 FlashInitBlock()

```
static blt_bool FlashInitBlock (
    tFlashBlockInfo * block,
    blt_addr address ) [static]
```

Copies data currently in flash to the block->data and sets the base address.

Parameters

<i>block</i>	Pointer to flash block info structure to operate on.
<i>address</i>	Base address of the block data.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

Referenced by FlashAddToBlock(), and FlashSwitchBlock().

7.409.2.12 FlashReinit()

```
void FlashReinit (
    void )
```

Reinitializes the flash driver.

Returns

none.

7.409.2.13 FlashSwitchBlock()

```
static tFlashBlockInfo * FlashSwitchBlock (
    tFlashBlockInfo * block,
    blt_addr base_addr ) [static]
```

Switches blocks by programming the current one and initializing the next.

Parameters

<i>block</i>	Pointer to flash block info structure to operate on.
<i>base_addr</i>	Base address of the next block.

Returns

The pointer of the block info struct that is no being used, or a NULL pointer in case of error.

Referenced by FlashAddToBlock().

7.409.2.14 FlashVerifyChecksum()

```
blt_bool FlashVerifyChecksum (
    void )
```

Verifies the checksum, which indicates that a valid user program is present and can be started.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.409.2.15 FlashWrite()

```
blt_bool FlashWrite (
    blt_addr addr,
    blt_int32u len,
    blt_int8u * data )
```

Writes the data to flash through a flash block manager. Note that this function also checks that no data is programmed outside the flash memory region, so the bootloader can never be overwritten.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.
<i>data</i>	Pointer to the data buffer.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

Referenced by FlashWriteChecksum().

7.409.2.16 FlashWriteBlock()

```
static blt_bool FlashWriteBlock (
    tFlashBlockInfo * block ) [static]
```

Programs FLASH_WRITE_BLOCK_SIZE bytes to flash from the block->data array.

Parameters

<i>block</i>	Pointer to flash block info structure to operate on.
--------------	--

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

Referenced by FlashDone(), and FlashSwitchBlock().

7.409.2.17 FlashWriteChecksum()

```
blt_bool FlashWriteChecksum (
    void )
```

Writes a checksum of the user program to non-volatile memory. This is performed once the entire user program has been programmed. Through the checksum, the bootloader can check if the programming session was completed, which indicates that a valid user programming is present and can be started.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.409.3 Variable Documentation

7.409.3.1 blockInfo

```
tFlashBlockInfo blockInfo [static]
```

Local variable with information about the flash block that is currently being operated on.

The smallest amount of flash that can be programmed is FLASH_WRITE_BLOCK_SIZE. A flash block manager is implemented in this driver and stores info in this variable. Whenever new data should be flashed, it is first added to a RAM buffer, which is part of this variable. Whenever the RAM buffer, which has the size of a flash block, is full or data needs to be written to a different block, the contents of the RAM buffer are programmed to flash. The flash block manager requires some software overhead, yet results in faster flash programming because data is first harvested, ideally until there is enough to program an entire flash block, before the flash device is actually operated on.

Referenced by FlashSwitchBlock().

7.409.3.2 bootBlockInfo

```
tFlashBlockInfo bootBlockInfo [static]
```

Local variable with information about the flash boot block.

The first block of the user program holds the vector table, which on the STM32 is also the where the checksum is written to. Is it likely that the vector table is first flashed and then, at the end of the programming sequence, the checksum. This means that this flash block need to be written to twice. Normally this is not a problem with flash memory, as long as you write the same values to those bytes that are not supposed to be changed and the locations where you do write to are still in the erased 0xFF state. Unfortunately, writing twice to flash this way, does not work reliably on all micros. This is why we need to have an extra block, the bootblock, placed under the management of the block manager. This way is it possible to implement functionality so that the bootblock is only written to once at the end of the programming sequence.

Referenced by FlashSwitchBlock().

7.409.3.3 flashLayout

```
const tFlashSector flashLayout[] [static]
```

If desired, it is possible to set BOOT_FLASH_CUSTOM_LAYOUT_ENABLE to > 0 in blt_conf.h and then implement your own version of the flashLayout[] table in a source-file with the name flash_layout.c. This way you customize the flash memory size reserved for the bootloader, without having to modify the flashLayout[] table in this file directly. This file will then include flash_layout.c so there is no need to compile it additionally with your project.

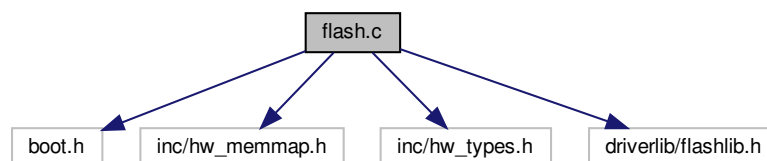
Array wit the layout of the flash memory.

Also controls what part of the flash memory is reserved for the bootloader. If the bootloader size changes, the reserved sectors for the bootloader might need adjustment to make sure the bootloader doesn't get overwritten. The current flash layout does not reflect the minimum sector size of the physical flash (1 - 2kb), because this would make the table quit long and a waste of ROM. The minimum sector size is only really needed when erasing the flash. This can still be done in combination with macro FLASH_ERASE_BLOCK_SIZE.

7.410 flash.c File Reference

Bootloader flash driver source file.

```
#include "boot.h"
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "driverlib/flashlib.h"
Include dependency graph for ARMCM3_LM3S/flash.c:
```



Data Structures

- struct [tFlashSector](#)
Flash sector descriptor type.
- struct [tFlashBlockInfo](#)
Structure type for grouping flash block information.

Macros

- #define [FLASH_INVALID_SECTOR](#) (0xff)
Value for an invalid flash sector.
- #define [FLASH_INVALID_ADDRESS](#) (0xffffffff)
Value for an invalid flash address.
- #define [FLASH_WRITE_BLOCK_SIZE](#) (512)
Standard size of a flash block for writing.
- #define [FLASH_TOTAL_SECTORS](#) (sizeof(flashLayout)/sizeof(flashLayout[0]))
Total numbers of sectors in array flashLayout[].
- #define [FLASH_END_ADDRESS](#)
End address of the bootloader programmable flash.
- #define [FLASH_ERASE_BLOCK_SIZE](#) (0x400)
Number of bytes to erase per erase operation.
- #define [BOOT_FLASH_VECTOR_TABLE_CS_OFFSET](#) (0xF0)
Offset into the user program's vector table where the checksum is located. For this target it is set to the end of the vector table. Note that the value can be overridden in blt_conf.h, because the size of the vector table could vary. When changing this value, don't forget to update the location of the checksum in the user program accordingly. Otherwise the checksum verification will always fail.

Functions

- static [blt_bool](#) [FlashInitBlock](#) ([tFlashBlockInfo](#) *block, [blt_addr](#) address)
Copies data currently in flash to the block->data and sets the base address.
- static [tFlashBlockInfo](#) * [FlashSwitchBlock](#) ([tFlashBlockInfo](#) *block, [blt_addr](#) base_addr)
Switches blocks by programming the current one and initializing the next.
- static [blt_bool](#) [FlashAddToBlock](#) ([tFlashBlockInfo](#) *block, [blt_addr](#) address, [blt_int8u](#) *data, [blt_int32u](#) len)
Programming is done per block. This function adds data to the block that is currently collecting data to be written to flash. If the address is outside of the current block, the current block is written to flash and a new block is initialized.
- static [blt_bool](#) [FlashWriteBlock](#) ([tFlashBlockInfo](#) *block)
Programs FLASH_WRITE_BLOCK_SIZE bytes to flash from the block->data array.
- static [blt_bool](#) [FlashEraseSectors](#) ([blt_int8u](#) first_sector, [blt_int8u](#) last_sector)
Erases the flash sectors from first_sector up until last_sector.
- static [blt_int8u](#) [FlashGetSector](#) ([blt_addr](#) address)
Determines the flash sector the address is in.
- static [blt_addr](#) [FlashGetSectorBaseAddr](#) ([blt_int8u](#) sector)
Determines the flash sector base address.
- static [blt_addr](#) [FlashGetSectorSize](#) ([blt_int8u](#) sector)
Determines the flash sector size.
- void [FlashInit](#) (void)
Initializes the flash driver.
- void [FlashReinit](#) (void)
Reinitializes the flash driver.

- `blt_bool FlashWrite (blt_addr addr, blt_int32u len, blt_int8u *data)`
Writes the data to flash through a flash block manager. Note that this function also checks that no data is programmed outside the flash memory region, so the bootloader can never be overwritten.
- `blt_bool FlashErase (blt_addr addr, blt_int32u len)`
Erases the flash memory. Note that this function also checks that no data is erased outside the flash memory region, so the bootloader can never be erased.
- `blt_bool FlashWriteChecksum (void)`
Writes a checksum of the user program to non-volatile memory. This is performed once the entire user program has been programmed. Through the checksum, the bootloader can check if the programming session was completed, which indicates that a valid user programming is present and can be started.
- `blt_bool FlashVerifyChecksum (void)`
Verifies the checksum, which indicates that a valid user program is present and can be started.
- `blt_bool FlashDone (void)`
Finalizes the flash driver operations. There could still be data in the currently active block that needs to be flashed.
- `blt_addr FlashGetUserProgBaseAddress (void)`
Obtains the base address of the flash memory available to the user program. This is basically the first address in the flashLayout table.

Variables

- static const `tFlashSector flashLayout []`
If desired, it is possible to set `BOOT_FLASH_CUSTOM_LAYOUT_ENABLE` to `> 0` in `blt_conf.h` and then implement your own version of the `flashLayout[]` table in a source-file with the name `flash_layout.c`. This way you customize the flash memory size reserved for the bootloader, without having to modify the `flashLayout[]` table in this file directly. This file will then include `flash_layout.c` so there is no need to compile it additionally with your project.
- static `tFlashBlockInfo blockInfo`
Local variable with information about the flash block that is currently being operated on.
- static `tFlashBlockInfo bootBlockInfo`
Local variable with information about the flash boot block.

7.410.1 Detailed Description

Bootloader flash driver source file.

7.410.2 Function Documentation

7.410.2.1 FlashAddToBlock()

```
static blt_bool FlashAddToBlock (
    tFlashBlockInfo * block,
    blt_addr address,
    blt_int8u * data,
    blt_int32u len ) [static]
```

Programming is done per block. This function adds data to the block that is currently collecting data to be written to flash. If the address is outside of the current block, the current block is written to flash and a new block is initialized.

Parameters

<i>block</i>	Pointer to flash block info structure to operate on.
<i>address</i>	Flash destination address.
<i>data</i>	Pointer to the byte array with data.
<i>len</i>	Number of bytes to add to the block.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

Referenced by FlashWrite().

7.410.2.2 FlashDone()

```
blt_bool FlashDone (  
    void )
```

Finalizes the flash driver operations. There could still be data in the currently active block that needs to be flashed.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.410.2.3 FlashErase()

```
blt_bool FlashErase (  
    blt_addr addr,  
    blt_int32u len )
```

Erases the flash memory. Note that this function also checks that no data is erased outside the flash memory region, so the bootloader can never be erased.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.410.2.4 FlashEraseSectors()

```
static blt_bool FlashEraseSectors (
    blt_int8u first_sector,
    blt_int8u last_sector ) [static]
```

Erases the flash sectors from `first_sector` up until `last_sector`.

Parameters

<i>first_sector</i>	First flash sector number.
<i>last_sector</i>	Last flash sector number.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

Referenced by FlashErase().

7.410.2.5 FlashGetSector()

```
static blt_int8u FlashGetSector (
    blt_addr address ) [static]
```

Determines the flash sector the address is in.

Parameters

<i>address</i>	Address in the flash sector.
----------------	------------------------------

Returns

Flash sector number or FLASH_INVALID_SECTOR.

Referenced by FlashErase(), FlashWrite(), and FlashWriteBlock().

7.410.2.6 FlashGetSectorBaseAddr()

```
static blt_addr FlashGetSectorBaseAddr (
    blt_int8u sector ) [static]
```

Determines the flash sector base address.

Parameters

<i>sector</i>	Sector to get the base address of.
---------------	------------------------------------

Returns

Flash sector base address or FLASH_INVALID_ADDRESS.

Referenced by FlashEraseSectors().

7.410.2.7 FlashGetSectorSize()

```
static blt_addr FlashGetSectorSize (  
    blt_int8u sector ) [static]
```

Determines the flash sector size.

Parameters

<i>sector</i>	Sector to get the size of.
---------------	----------------------------

Returns

Flash sector size or 0.

Referenced by FlashEraseSectors().

7.410.2.8 FlashGetUserProgBaseAddress()

```
blt_addr FlashGetUserProgBaseAddress (  
    void )
```

Obtains the base address of the flash memory available to the user program. This is basically the first address in the flashLayout table.

Returns

Base address.

7.410.2.9 FlashInit()

```
void FlashInit (  
    void )
```

Initializes the flash driver.

Returns

none.

7.410.2.10 FlashInitBlock()

```
static blt_bool FlashInitBlock (
    tFlashBlockInfo * block,
    blt_addr address ) [static]
```

Copies data currently in flash to the block->data and sets the base address.

Parameters

<i>block</i>	Pointer to flash block info structure to operate on.
<i>address</i>	Base address of the block data.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

Referenced by FlashAddToBlock(), and FlashSwitchBlock().

7.410.2.11 FlashReinit()

```
void FlashReinit (
    void )
```

Reinitializes the flash driver.

Returns

none.

7.410.2.12 FlashSwitchBlock()

```
static tFlashBlockInfo * FlashSwitchBlock (
    tFlashBlockInfo * block,
    blt_addr base_addr ) [static]
```

Switches blocks by programming the current one and initializing the next.

Parameters

<i>block</i>	Pointer to flash block info structure to operate on.
<i>base_addr</i>	Base address of the next block.

Returns

The pointer of the block info struct that is no being used, or a NULL pointer in case of error.

Referenced by FlashAddToBlock().

7.410.2.13 FlashVerifyChecksum()

```
blt_bool FlashVerifyChecksum (
    void )
```

Verifies the checksum, which indicates that a valid user program is present and can be started.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.410.2.14 FlashWrite()

```
blt_bool FlashWrite (
    blt_addr addr,
    blt_int32u len,
    blt_int8u * data )
```

Writes the data to flash through a flash block manager. Note that this function also checks that no data is programmed outside the flash memory region, so the bootloader can never be overwritten.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.
<i>data</i>	Pointer to the data buffer.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

Referenced by FlashWriteChecksum().

7.410.2.15 FlashWriteBlock()

```
static blt_bool FlashWriteBlock (
    tFlashBlockInfo * block ) [static]
```

Programs FLASH_WRITE_BLOCK_SIZE bytes to flash from the block->data array.

Parameters

<i>block</i>	Pointer to flash block info structure to operate on.
--------------	--

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

Referenced by FlashDone(), and FlashSwitchBlock().

7.410.2.16 FlashWriteChecksum()

```
blt_bool FlashWriteChecksum (  
    void )
```

Writes a checksum of the user program to non-volatile memory. This is performed once the entire user program has been programmed. Through the checksum, the bootloader can check if the programming session was completed, which indicates that a valid user programming is present and can be started.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.410.3 Variable Documentation

7.410.3.1 blockInfo

```
tFlashBlockInfo blockInfo [static]
```

Local variable with information about the flash block that is currently being operated on.

The smallest amount of flash that can be programmed is FLASH_WRITE_BLOCK_SIZE. A flash block manager is implemented in this driver and stores info in this variable. Whenever new data should be flashed, it is first added to a RAM buffer, which is part of this variable. Whenever the RAM buffer, which has the size of a flash block, is full or data needs to be written to a different block, the contents of the RAM buffer are programmed to flash. The flash block manager requires some software overhead, yet results in faster flash programming because data is first harvested, ideally until there is enough to program an entire flash block, before the flash device is actually operated on.

Referenced by FlashSwitchBlock().

7.410.3.2 bootBlockInfo

```
tFlashBlockInfo bootBlockInfo [static]
```

Local variable with information about the flash boot block.

The first block of the user program holds the vector table, which on the STM32 is also the where the checksum is written to. Is it likely that the vector table is first flashed and then, at the end of the programming sequence, the checksum. This means that this flash block need to be written to twice. Normally this is not a problem with flash memory, as long as you write the same values to those bytes that are not supposed to be changed and the locations where you do write to are still in the erased 0xFF state. Unfortunately, writing twice to flash this way, does not work reliably on all micros. This is why we need to have an extra block, the bootblock, placed under the management of the block manager. This way is it possible to implement functionality so that the bootblock is only written to once at the end of the programming sequence.

Referenced by FlashSwitchBlock().

7.410.3.3 flashLayout

```
const tFlashSector flashLayout[] [static]
```

If desired, it is possible to set BOOT_FLASH_CUSTOM_LAYOUT_ENABLE to > 0 in blt_conf.h and then implement your own version of the flashLayout[] table in a source-file with the name flash_layout.c. This way you customize the flash memory size reserved for the bootloader, without having to modify the flashLayout[] table in this file directly. This file will then include flash_layout.c so there is no need to compile it additionally with your project.

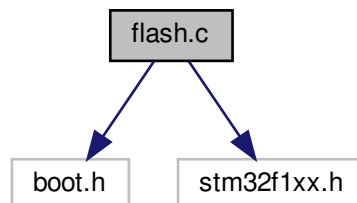
Array with the layout of the flash memory.

Also controls what part of the flash memory is reserved for the bootloader. If the bootloader size changes, the reserved sectors for the bootloader might need adjustment to make sure the bootloader doesn't get overwritten. The current flash layout does not reflect the minimum sector size of the physical flash (1 - 2kb), because this would make the table quite long and a waste of ROM. The minimum sector size is only really needed when erasing the flash. This can still be done in combination with macro FLASH_ERASE_BLOCK_SIZE.

7.411 flash.c File Reference

Bootloader flash driver source file.

```
#include "boot.h"
#include "stm32f1xx.h"
Include dependency graph for ARMCM3_STM32F1/flash.c:
```



Data Structures

- struct [tFlashSector](#)
Flash sector descriptor type.
- struct [tFlashBlockInfo](#)
Structure type for grouping flash block information.

Macros

- #define [FLASH_INVALID_ADDRESS](#) (0xffffffff)
Value for an invalid flash address.
- #define [FLASH_WRITE_BLOCK_SIZE](#) (512)
Standard size of a flash block for writing.
- #define [FLASH_TOTAL_SEGMENTS](#) (sizeof(flashLayout)/sizeof(flashLayout[0]))
Total numbers of segments in array flashLayout[].
- #define [FLASH_LAST_SEGMENT_IDX](#) (FLASH_TOTAL_SEGMENTS-1)
Index of the last segment in array flashLayout[].
- #define [FLASH_START_ADDRESS](#) (flashLayout[0].sector_start)
Start address of the bootloader programmable flash.
- #define [FLASH_END_ADDRESS](#)
End address of the bootloader programmable flash.
- #define [FLASH_ERASE_BLOCK_SIZE](#) (FLASH_PAGE_SIZE)
Number of bytes to erase per erase operation.
- #define [BOOT_FLASH_VECTOR_TABLE_CS_OFFSET](#) (0x150)
Offset into the user program's vector table where the checksum is located. For this target it is set to the end of the vector table. Note that the value can be overridden in blt_conf.h, because the size of the vector table could vary. When changing this value, don't forget to update the location of the checksum in the user program accordingly. Otherwise the checksum verification will always fail.

Functions

- static [blt_bool](#) [FlashInitBlock](#) ([tFlashBlockInfo](#) *block, [blt_addr](#) address)
Copies data currently in flash to the block->data and sets the base address.
- static [tFlashBlockInfo](#) * [FlashSwitchBlock](#) ([tFlashBlockInfo](#) *block, [blt_addr](#) base_addr)
Switches blocks by programming the current one and initializing the next.
- static [blt_bool](#) [FlashAddToBlock](#) ([tFlashBlockInfo](#) *block, [blt_addr](#) address, [blt_int8u](#) *data, [blt_int32u](#) len)
Programming is done per block. This function adds data to the block that is currently collecting data to be written to flash. If the address is outside of the current block, the current block is written to flash and a new block is initialized.
- static [blt_bool](#) [FlashWriteBlock](#) ([tFlashBlockInfo](#) *block)
Programs FLASH_WRITE_BLOCK_SIZE bytes to flash from the block->data array.
- void [FlashInit](#) (void)
Initializes the flash driver.
- void [FlashReinit](#) (void)
Reinitializes the flash driver.
- [blt_bool](#) [FlashWrite](#) ([blt_addr](#) addr, [blt_int32u](#) len, [blt_int8u](#) *data)
Writes the data to flash through a flash block manager. Note that this function also checks that no data is programmed outside the flash memory region, so the bootloader can never be overwritten.
- [blt_bool](#) [FlashErase](#) ([blt_addr](#) addr, [blt_int32u](#) len)
Erases the flash memory. Note that this function also checks that no data is erased outside the flash memory region, so the bootloader can never be erased.
- [blt_bool](#) [FlashWriteChecksum](#) (void)

Writes a checksum of the user program to non-volatile memory. This is performed once the entire user program has been programmed. Through the checksum, the bootloader can check if the programming session was completed, which indicates that a valid user programming is present and can be started.

- [blt_bool FlashVerifyChecksum](#) (void)

Verifies the checksum, which indicates that a valid user program is present and can be started.

- [blt_bool FlashDone](#) (void)

Finalizes the flash driver operations. There could still be data in the currently active block that needs to be flashed.

- [blt_addr FlashGetUserProgBaseAddress](#) (void)

Obtains the base address of the flash memory available to the user program. This is basically the first address in the flashLayout table.

Variables

- static const [tFlashSector flashLayout](#) []

If desired, it is possible to set `BOOT_FLASH_CUSTOM_LAYOUT_ENABLE` to `> 0` in `blt_conf.h` and then implement your own version of the `flashLayout[]` table in a source-file with the name `flash_layout.c`. This way you customize the flash memory size reserved for the bootloader, without having to modify the `flashLayout[]` table in this file directly. This file will then include `flash_layout.c` so there is no need to compile it additionally with your project.

- static [tFlashBlockInfo blockInfo](#)

Local variable with information about the flash block that is currently being operated on.

- static [tFlashBlockInfo bootBlockInfo](#)

Local variable with information about the flash boot block.

7.411.1 Detailed Description

Bootloader flash driver source file.

7.411.2 Function Documentation

7.411.2.1 FlashAddToBlock()

```
static blt_bool FlashAddToBlock (
    tFlashBlockInfo * block,
    blt_addr address,
    blt_int8u * data,
    blt_int32u len ) [static]
```

Programming is done per block. This function adds data to the block that is currently collecting data to be written to flash. If the address is outside of the current block, the current block is written to flash and a new block is initialized.

Parameters

<i>block</i>	Pointer to flash block info structure to operate on.
<i>address</i>	Flash destination address.
<i>data</i>	Pointer to the byte array with data.
<i>len</i>	Number of bytes to add to the block.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

Referenced by FlashWrite().

7.411.2.2 FlashDone()

```
blt_bool FlashDone (  
    void )
```

Finalizes the flash driver operations. There could still be data in the currently active block that needs to be flashed.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.411.2.3 FlashErase()

```
blt_bool FlashErase (  
    blt_addr addr,  
    blt_int32u len )
```

Erases the flash memory. Note that this function also checks that no data is erased outside the flash memory region, so the bootloader can never be erased.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.411.2.4 FlashGetUserProgBaseAddress()

```
blt_addr FlashGetUserProgBaseAddress (  
    void )
```

Obtains the base address of the flash memory available to the user program. This is basically the first address in the flashLayout table.

Returns

Base address.

7.411.2.5 FlashInit()

```
void FlashInit (
    void )
```

Initializes the flash driver.

Returns

none.

7.411.2.6 FlashInitBlock()

```
static blt_bool FlashInitBlock (
    tFlashBlockInfo * block,
    blt_addr address ) [static]
```

Copies data currently in flash to the block->data and sets the base address.

Parameters

<i>block</i>	Pointer to flash block info structure to operate on.
<i>address</i>	Base address of the block data.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

Referenced by FlashAddToBlock(), and FlashSwitchBlock().

7.411.2.7 FlashReinit()

```
void FlashReinit (
    void )
```

Reinitializes the flash driver.

Returns

none.

7.411.2.8 FlashSwitchBlock()

```
static tFlashBlockInfo * FlashSwitchBlock (
    tFlashBlockInfo * block,
    blt_addr base_addr ) [static]
```

Switches blocks by programming the current one and initializing the next.

Parameters

<i>block</i>	Pointer to flash block info structure to operate on.
<i>base_addr</i>	Base address of the next block.

Returns

The pointer of the block info struct that is no being used, or a NULL pointer in case of error.

Referenced by FlashAddToBlock().

7.411.2.9 FlashVerifyChecksum()

```
blt_bool FlashVerifyChecksum (  
    void )
```

Verifies the checksum, which indicates that a valid user program is present and can be started.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.411.2.10 FlashWrite()

```
blt_bool FlashWrite (  
    blt_addr addr,  
    blt_int32u len,  
    blt_int8u * data )
```

Writes the data to flash through a flash block manager. Note that this function also checks that no data is programmed outside the flash memory region, so the bootloader can never be overwritten.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.
<i>data</i>	Pointer to the data buffer.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

Referenced by FlashWriteChecksum().

7.411.2.11 FlashWriteBlock()

```
static blt_bool FlashWriteBlock (
    tFlashBlockInfo * block ) [static]
```

Programs FLASH_WRITE_BLOCK_SIZE bytes to flash from the block->data array.

Parameters

<i>block</i>	Pointer to flash block info structure to operate on.
--------------	--

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

Referenced by FlashDone(), and FlashSwitchBlock().

7.411.2.12 FlashWriteChecksum()

```
blt_bool FlashWriteChecksum (
    void )
```

Writes a checksum of the user program to non-volatile memory. This is performed once the entire user program has been programmed. Through the checksum, the bootloader can check if the programming session was completed, which indicates that a valid user programming is present and can be started.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.411.3 Variable Documentation

7.411.3.1 blockInfo

```
tFlashBlockInfo blockInfo [static]
```

Local variable with information about the flash block that is currently being operated on.

The smallest amount of flash that can be programmed is FLASH_WRITE_BLOCK_SIZE. A flash block manager is implemented in this driver and stores info in this variable. Whenever new data should be flashed, it is first added to a RAM buffer, which is part of this variable. Whenever the RAM buffer, which has the size of a flash block, is full or data needs to be written to a different block, the contents of the RAM buffer are programmed to flash. The flash block manager requires some software overhead, yet results in faster flash programming because data is first harvested, ideally until there is enough to program an entire flash block, before the flash device is actually operated on.

Referenced by FlashSwitchBlock().

7.411.3.2 bootBlockInfo

```
tFlashBlockInfo bootBlockInfo [static]
```

Local variable with information about the flash boot block.

The first block of the user program holds the vector table, which on the STM32 is also the where the checksum is written to. Is it likely that the vector table is first flashed and then, at the end of the programming sequence, the checksum. This means that this flash block need to be written to twice. Normally this is not a problem with flash memory, as long as you write the same values to those bytes that are not supposed to be changed and the locations where you do write to are still in the erased 0xFF state. Unfortunately, writing twice to flash this way, does not work reliably on all micros. This is why we need to have an extra block, the bootblock, placed under the management of the block manager. This way is it possible to implement functionality so that the bootblock is only written to once at the end of the programming sequence.

Referenced by FlashSwitchBlock().

7.411.3.3 flashLayout

```
const tFlashSector flashLayout[] [static]
```

If desired, it is possible to set `BOOT_FLASH_CUSTOM_LAYOUT_ENABLE` to `> 0` in `blt_conf.h` and then implement your own version of the `flashLayout[]` table in a source-file with the name `flash_layout.c`. This way you customize the flash memory size reserved for the bootloader, without having to modify the `flashLayout[]` table in this file directly. This file will then include `flash_layout.c` so there is no need to compile it additionally with your project.

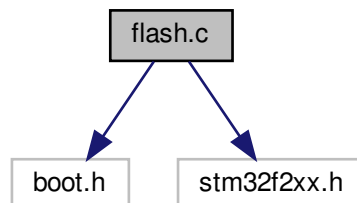
Array with the layout of the flash memory.

Also controls what part of the flash memory is reserved for the bootloader. If the bootloader size changes, the reserved sectors for the bootloader might need adjustment to make sure the bootloader doesn't get overwritten. The current flash layout does not reflect the minimum sector size of the physical flash (1 - 2kb), because this would make the table quite long and a waste of ROM. The minimum sector size is only really needed when erasing the flash. This can still be done in combination with macro `FLASH_ERASE_BLOCK_SIZE`.

7.412 flash.c File Reference

Bootloader flash driver source file.

```
#include "boot.h"
#include "stm32f2xx.h"
Include dependency graph for ARMCM3_STM32F2/flash.c:
```



Data Structures

- struct [tFlashSector](#)
Flash sector descriptor type.
- struct [tFlashBlockInfo](#)
Structure type for grouping flash block information.

Macros

- #define [FLASH_INVALID_SECTOR](#) (0xff)
Value for an invalid flash sector.
- #define [FLASH_INVALID_ADDRESS](#) (0xffffffff)
Value for an invalid flash address.
- #define [FLASH_WRITE_BLOCK_SIZE](#) (512)
Standard size of a flash block for writing.
- #define [FLASH_TOTAL_SECTORS](#) (sizeof(flashLayout)/sizeof(flashLayout[0]))
Total numbers of sectors in array flashLayout[].
- #define [FLASH_END_ADDRESS](#)
End address of the bootloader programmable flash.
- #define [BOOT_FLASH_VECTOR_TABLE_CS_OFFSET](#) (0x184)
Offset into the user program's vector table where the checksum is located. For this target it is set to the end of the vector table. Note that the value can be overridden in blt_conf.h, because the size of the vector table could vary. When changing this value, don't forget to update the location of the checksum in the user program accordingly. Otherwise the checksum verification will always fail.

Functions

- static [blt_bool](#) [FlashInitBlock](#) ([tFlashBlockInfo](#) *block, [blt_addr](#) address)
Copies data currently in flash to the block->data and sets the base address.
- static [tFlashBlockInfo](#) * [FlashSwitchBlock](#) ([tFlashBlockInfo](#) *block, [blt_addr](#) base_addr)
Switches blocks by programming the current one and initializing the next.
- static [blt_bool](#) [FlashAddToBlock](#) ([tFlashBlockInfo](#) *block, [blt_addr](#) address, [blt_int8u](#) *data, [blt_int32u](#) len)
Programming is done per block. This function adds data to the block that is currently collecting data to be written to flash. If the address is outside of the current block, the current block is written to flash and a new block is initialized.
- static [blt_bool](#) [FlashWriteBlock](#) ([tFlashBlockInfo](#) *block)
Programs FLASH_WRITE_BLOCK_SIZE bytes to flash from the block->data array.
- static [blt_bool](#) [FlashEraseSectors](#) ([blt_int8u](#) first_sector, [blt_int8u](#) last_sector)
Erases the flash sectors from first_sector up until last_sector.
- static [blt_int8u](#) [FlashGetSector](#) ([blt_addr](#) address)
Determines the flash sector the address is in.
- void [FlashInit](#) (void)
Initializes the flash driver.
- void [FlashReinit](#) (void)
Reinitializes the flash driver.
- [blt_bool](#) [FlashWrite](#) ([blt_addr](#) addr, [blt_int32u](#) len, [blt_int8u](#) *data)
Writes the data to flash through a flash block manager. Note that this function also checks that no data is programmed outside the flash memory region, so the bootloader can never be overwritten.
- [blt_bool](#) [FlashErase](#) ([blt_addr](#) addr, [blt_int32u](#) len)
Erases the flash memory. Note that this function also checks that no data is erased outside the flash memory region, so the bootloader can never be erased.
- [blt_bool](#) [FlashWriteChecksum](#) (void)

Writes a checksum of the user program to non-volatile memory. This is performed once the entire user program has been programmed. Through the checksum, the bootloader can check if the programming session was completed, which indicates that a valid user programming is present and can be started.

- [blt_bool FlashVerifyChecksum](#) (void)

Verifies the checksum, which indicates that a valid user program is present and can be started.

- [blt_bool FlashDone](#) (void)

Finalizes the flash driver operations. There could still be data in the currently active block that needs to be flashed.

- [blt_addr FlashGetUserProgBaseAddress](#) (void)

Obtains the base address of the flash memory available to the user program. This is basically the first address in the flashLayout table.

Variables

- static const [tFlashSector flashLayout](#) []

If desired, it is possible to set `BOOT_FLASH_CUSTOM_LAYOUT_ENABLE` to `> 0` in `blt_conf.h` and then implement your own version of the `flashLayout[]` table in a source-file with the name `flash_layout.c`. This way you customize the flash memory size reserved for the bootloader, without having to modify the `flashLayout[]` table in this file directly. This file will then include `flash_layout.c` so there is no need to compile it additionally with your project.

- static [tFlashBlockInfo blockInfo](#)

Local variable with information about the flash block that is currently being operated on.

- static [tFlashBlockInfo bootBlockInfo](#)

Local variable with information about the flash boot block.

7.412.1 Detailed Description

Bootloader flash driver source file.

7.412.2 Function Documentation

7.412.2.1 FlashAddToBlock()

```
static blt_bool FlashAddToBlock (
    tFlashBlockInfo * block,
    blt_addr address,
    blt_int8u * data,
    blt_int32u len ) [static]
```

Programming is done per block. This function adds data to the block that is currently collecting data to be written to flash. If the address is outside of the current block, the current block is written to flash and a new block is initialized.

Parameters

<i>block</i>	Pointer to flash block info structure to operate on.
<i>address</i>	Flash destination address.
<i>data</i>	Pointer to the byte array with data.
<i>len</i>	Number of bytes to add to the block.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

Referenced by FlashWrite().

7.412.2.2 FlashDone()

```
blt_bool FlashDone (
    void )
```

Finalizes the flash driver operations. There could still be data in the currently active block that needs to be flashed.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.412.2.3 FlashErase()

```
blt_bool FlashErase (
    blt_addr addr,
    blt_int32u len )
```

Erases the flash memory. Note that this function also checks that no data is erased outside the flash memory region, so the bootloader can never be erased.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.412.2.4 FlashEraseSectors()

```
static blt_bool FlashEraseSectors (
    blt_int8u first_sector,
    blt_int8u last_sector ) [static]
```

Erases the flash sectors from first_sector up until last_sector.

Parameters

<i>first_sector</i>	First flash sector number.
<i>last_sector</i>	Last flash sector number.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

Referenced by FlashErase().

7.412.2.5 FlashGetSector()

```
static blt_int8u FlashGetSector (
    blt_addr address ) [static]
```

Determines the flash sector the address is in.

Parameters

<i>address</i>	Address in the flash sector.
----------------	------------------------------

Returns

Flash sector number or FLASH_INVALID_SECTOR.

Referenced by FlashErase(), and FlashWrite().

7.412.2.6 FlashGetUserProgBaseAddress()

```
blt_addr FlashGetUserProgBaseAddress (
    void )
```

Obtains the base address of the flash memory available to the user program. This is basically the first address in the flashLayout table.

Returns

Base address.

7.412.2.7 FlashInit()

```
void FlashInit (
    void )
```

Initializes the flash driver.

Returns

none.

7.412.2.8 FlashInitBlock()

```
static blt_bool FlashInitBlock (
    tFlashBlockInfo * block,
    blt_addr address ) [static]
```

Copies data currently in flash to the block->data and sets the base address.

Parameters

<i>block</i>	Pointer to flash block info structure to operate on.
<i>address</i>	Base address of the block data.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

Referenced by FlashAddToBlock(), and FlashSwitchBlock().

7.412.2.9 FlashReinit()

```
void FlashReinit (
    void )
```

Reinitializes the flash driver.

Returns

none.

7.412.2.10 FlashSwitchBlock()

```
static tFlashBlockInfo * FlashSwitchBlock (
    tFlashBlockInfo * block,
    blt_addr base_addr ) [static]
```

Switches blocks by programming the current one and initializing the next.

Parameters

<i>block</i>	Pointer to flash block info structure to operate on.
<i>base_addr</i>	Base address of the next block.

Returns

The pointer of the block info struct that is no being used, or a NULL pointer in case of error.

Referenced by FlashAddToBlock().

7.412.2.11 FlashVerifyChecksum()

```
blt_bool FlashVerifyChecksum (  
    void )
```

Verifies the checksum, which indicates that a valid user program is present and can be started.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.412.2.12 FlashWrite()

```
blt_bool FlashWrite (  
    blt_addr addr,  
    blt_int32u len,  
    blt_int8u * data )
```

Writes the data to flash through a flash block manager. Note that this function also checks that no data is programmed outside the flash memory region, so the bootloader can never be overwritten.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.
<i>data</i>	Pointer to the data buffer.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

Referenced by FlashWriteChecksum().

7.412.2.13 FlashWriteBlock()

```
static blt_bool FlashWriteBlock (
    tFlashBlockInfo * block ) [static]
```

Programs FLASH_WRITE_BLOCK_SIZE bytes to flash from the block->data array.

Parameters

<i>block</i>	Pointer to flash block info structure to operate on.
--------------	--

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

Referenced by FlashDone(), and FlashSwitchBlock().

7.412.2.14 FlashWriteChecksum()

```
blt_bool FlashWriteChecksum (
    void )
```

Writes a checksum of the user program to non-volatile memory. This is performed once the entire user program has been programmed. Through the checksum, the bootloader can check if the programming session was completed, which indicates that a valid user programming is present and can be started.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.412.3 Variable Documentation

7.412.3.1 blockInfo

```
tFlashBlockInfo blockInfo [static]
```

Local variable with information about the flash block that is currently being operated on.

The smallest amount of flash that can be programmed is FLASH_WRITE_BLOCK_SIZE. A flash block manager is implemented in this driver and stores info in this variable. Whenever new data should be flashed, it is first added to a RAM buffer, which is part of this variable. Whenever the RAM buffer, which has the size of a flash block, is full or data needs to be written to a different block, the contents of the RAM buffer are programmed to flash. The flash block manager requires some software overhead, yet results in faster flash programming because data is first harvested, ideally until there is enough to program an entire flash block, before the flash device is actually operated on.

Referenced by FlashSwitchBlock().

7.412.3.2 bootBlockInfo

```
tFlashBlockInfo bootBlockInfo [static]
```

Local variable with information about the flash boot block.

The first block of the user program holds the vector table, which on the STM32 is also the where the checksum is written to. Is it likely that the vector table is first flashed and then, at the end of the programming sequence, the checksum. This means that this flash block need to be written to twice. Normally this is not a problem with flash memory, as long as you write the same values to those bytes that are not supposed to be changed and the locations where you do write to are still in the erased 0xFF state. Unfortunately, writing twice to flash this way, does not work reliably on all micros. This is why we need to have an extra block, the bootblock, placed under the management of the block manager. This way is it possible to implement functionality so that the bootblock is only written to once at the end of the programming sequence.

Referenced by FlashSwitchBlock().

7.412.3.3 flashLayout

```
const tFlashSector flashLayout[] [static]
```

Initial value:

```
=
{

    { 0x08008000, 0x04000, 2},
    { 0x0800c000, 0x04000, 3},
    { 0x08010000, 0x10000, 4},

    { 0x08020000, 0x20000, 5},

    { 0x08040000, 0x20000, 6},
    { 0x08060000, 0x20000, 7},

    { 0x08080000, 0x20000, 8},
    { 0x080A0000, 0x20000, 9},
    { 0x080C0000, 0x20000, 10},
    { 0x080E0000, 0x20000, 11},

}
```

If desired, it is possible to set `BOOT_FLASH_CUSTOM_LAYOUT_ENABLE` to `> 0` in `blt_conf.h` and then implement your own version of the `flashLayout[]` table in a source-file with the name `flash_layout.c`. This way you customize the flash memory size reserved for the bootloader, without having to modify the `flashLayout[]` table in this file directly. This file will then include `flash_layout.c` so there is no need to compile it additionally with your project.

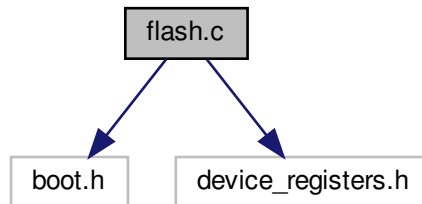
Array wit the layout of the flash memory.

Also controls what part of the flash memory is reserved for the bootloader. If the bootloader size changes, the reserved sectors for the bootloader might need adjustment to make sure the bootloader doesn't get overwritten.

7.413 flash.c File Reference

Bootloader flash driver source file.

```
#include "boot.h"
#include "device_registers.h"
Include dependency graph for ARMCM4_S32K14/flash.c:
```



Data Structures

- struct [tFlashSector](#)
Flash sector descriptor type.
- struct [tFlashBlockInfo](#)
Structure type for grouping flash block information.

Macros

- #define [FLASH_INVALID_SECTOR_IDX](#) (0xff)
Value for an invalid sector entry index into flashLayout[].
- #define [FLASH_INVALID_ADDRESS](#) (0xffffffff)
Value for an invalid flash address.
- #define [FLASH_WRITE_BLOCK_SIZE](#) (1024)
Standard size of a flash block for writing.
- #define [FLASH_ERASE_BLOCK_SIZE](#) (FEATURE_FLS_PF_BLOCK_SECTOR_SIZE)
Standard size of a flash block for erasing. This is either 2 or 4 kb depending on the microcontroller derivative.
- #define [FLASH_TOTAL_SECTORS](#) (sizeof(flashLayout)/sizeof(flashLayout[0]))
Total numbers of sectors in array flashLayout[].
- #define [FLASH_END_ADDRESS](#)
End address of the bootloader programmable flash.
- #define [FLASH_FTFC_CMD_PROGRAM_PHRASE](#) (0x07U)
FTFC program phrase command code.
- #define [FLASH_FTFC_CMD_ERASE_SECTOR](#) (0x09U)
FTFC erase sector command code.
- #define [BOOT_FLASH_VECTOR_TABLE_CS_OFFSET](#) (0x3F8)
Offset into the user program's vector table where the checksum is located. For this target it is set to the second to last entry (254) in the vector table. Note that the value can be overridden in blt_conf.h, because the size of the vector table could vary. When changing this value, don't forget to update the location of the checksum in the user program accordingly. Otherwise the checksum verification will always fail.

Functions

- static [blt_bool](#) [FlashInitBlock](#) ([tFlashBlockInfo](#) *block, [blt_addr](#) address)
Copies data currently in flash to the block->data and sets the base address.
- static [tFlashBlockInfo](#) * [FlashSwitchBlock](#) ([tFlashBlockInfo](#) *block, [blt_addr](#) base_addr)
Switches blocks by programming the current one and initializing the next.
- static [blt_bool](#) [FlashAddToBlock](#) ([tFlashBlockInfo](#) *block, [blt_addr](#) address, [blt_int8u](#) *data, [blt_int32u](#) len)
Programming is done per block. This function adds data to the block that is currently collecting data to be written to flash. If the address is outside of the current block, the current block is written to flash and a new block is initialized.
- static [blt_bool](#) [FlashWriteBlock](#) ([tFlashBlockInfo](#) *block)
Programs FLASH_WRITE_BLOCK_SIZE bytes to flash from the block->data array.
- static [blt_bool](#) [FlashEraseSectors](#) ([blt_int8u](#) first_sector_idx, [blt_int8u](#) last_sector_idx)
Erases the flash sectors from indices first_sector_idx up until last_sector_idx into the flashLayout[] array.
- static [blt_int8u](#) [FlashGetSectorIdx](#) ([blt_addr](#) address)
Determines the index into the flashLayout[] array of the flash sector that the specified address is in.
- static [START_FUNCTION_DECLARATION_RAMSECTION](#) void [FlashCommandSequence](#) (void)
If desired, it is possible to set BOOT_FLASH_CUSTOM_LAYOUT_ENABLE to > 0 in blt_conf.h and then implement your own version of the flashLayout[] table in a source-file with the name flash_layout.c. This way you customize the flash memory size reserved for the bootloader, without having to modify the flashLayout[] table in this file directly. This file will then include flash_layout.c so there is no need to compile it additionally with your project.
- void [FlashInit](#) (void)
Initializes the flash driver.
- void [FlashReinit](#) (void)
Reinitializes the flash driver.
- [blt_bool](#) [FlashWrite](#) ([blt_addr](#) addr, [blt_int32u](#) len, [blt_int8u](#) *data)
Writes the data to flash through a flash block manager. Note that this function also checks that no data is programmed outside the flash memory region, so the bootloader can never be overwritten.
- [blt_bool](#) [FlashErase](#) ([blt_addr](#) addr, [blt_int32u](#) len)
Erases the flash memory. Note that this function also checks that no data is erased outside the flash memory region, so the bootloader can never be erased.
- [blt_bool](#) [FlashWriteChecksum](#) (void)
Writes a checksum of the user program to non-volatile memory. This is performed once the entire user program has been programmed. Through the checksum, the bootloader can check if the programming session was completed, which indicates that a valid user programming is present and can be started.
- [blt_bool](#) [FlashVerifyChecksum](#) (void)
Verifies the checksum, which indicates that a valid user program is present and can be started.
- [blt_bool](#) [FlashDone](#) (void)
Finalizes the flash driver operations. There could still be data in the currently active block that needs to be flashed.
- [blt_addr](#) [FlashGetUserProgBaseAddress](#) (void)
Obtains the base address of the flash memory available to the user program. This is basically the first address in the flashLayout table.

Variables

- static [tFlashBlockInfo](#) [blockInfo](#)
Local variable with information about the flash block that is currently being operated on.
- static [tFlashBlockInfo](#) [bootBlockInfo](#)
Local variable with information about the flash boot block.

7.413.1 Detailed Description

Bootloader flash driver source file.

7.413.2 Function Documentation

7.413.2.1 FlashAddToBlock()

```
static blt_bool FlashAddToBlock (
    tFlashBlockInfo * block,
    blt_addr address,
    blt_int8u * data,
    blt_int32u len ) [static]
```

Programming is done per block. This function adds data to the block that is currently collecting data to be written to flash. If the address is outside of the current block, the current block is written to flash and a new block is initialized.

Parameters

<i>block</i>	Pointer to flash block info structure to operate on.
<i>address</i>	Flash destination address.
<i>data</i>	Pointer to the byte array with data.
<i>len</i>	Number of bytes to add to the block.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

Referenced by FlashWrite().

7.413.2.2 FlashCommandSequence()

```
static START_FUNCTION_DEFINITION_RAMSECTION void FlashCommandSequence (
    void ) [static]
```

If desired, it is possible to set `BOOT_FLASH_CUSTOM_LAYOUT_ENABLE` to `> 0` in `blt_conf.h` and then implement your own version of the `flashLayout[]` table in a source-file with the name `flash_layout.c`. This way you customize the flash memory size reserved for the bootloader, without having to modify the `flashLayout[]` table in this file directly. This file will then include `flash_layout.c` so there is no need to compile it additionally with your project.

Use the FTFC module to run the flash command sequence. It is assumed that that command and its necessary parameters were already written to the correct FTFC registers.

Array with the layout of the flash memory.

Also controls what part of the flash memory is reserved for the bootloader. If the bootloader size changes, the reserved sectors for the bootloader might need adjustment to make sure the bootloader doesn't get overwritten.

Attention

This function needs to run from RAM. It is configured such that the C start-up code automatically copies it from ROM to RAM in function `init_data_bss()`, which is called by the reset handler.

Returns

None.

Referenced by `FlashEraseSectors()`, `FlashGetSectorIdx()`, and `FlashWriteBlock()`.

7.413.2.3 FlashDone()

```
blt_bool FlashDone (
    void )
```

Finalizes the flash driver operations. There could still be data in the currently active block that needs to be flashed.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.413.2.4 FlashErase()

```
blt_bool FlashErase (
    blt_addr addr,
    blt_int32u len )
```

Erases the flash memory. Note that this function also checks that no data is erased outside the flash memory region, so the bootloader can never be erased.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.413.2.5 FlashEraseSectors()

```
static blt_bool FlashEraseSectors (
    blt_int8u first_sector_idx,
    blt_int8u last_sector_idx ) [static]
```

Erases the flash sectors from indices *first_sector_idx* up until *last_sector_idx* into the *flashLayout[]* array.

Parameters

<i>first_sector_idx</i>	First flash sector number index into <i>flashLayout[]</i> .
<i>last_sector_idx</i>	Last flash sector number index into <i>flashLayout[]</i> .

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

Referenced by FlashErase().

7.413.2.6 FlashGetSectorIdx()

```
static blt_int8u FlashGetSectorIdx (  
    blt_addr address ) [static]
```

Determines the index into the flashLayout[] array of the flash sector that the specified address is in.

Parameters

<i>address</i>	Address in the flash sector.
----------------	------------------------------

Returns

Flash sector index in flashLayout[] or FLASH_INVALID_SECTOR_IDX.

Referenced by FlashErase(), FlashWrite(), and FlashWriteBlock().

7.413.2.7 FlashGetUserProgBaseAddress()

```
blt_addr FlashGetUserProgBaseAddress (  
    void )
```

Obtains the base address of the flash memory available to the user program. This is basically the first address in the flashLayout table.

Returns

Base address.

7.413.2.8 FlashInit()

```
void FlashInit (  
    void )
```

Initializes the flash driver.

Returns

none.

7.413.2.9 FlashInitBlock()

```
static blt_bool FlashInitBlock (  
    tFlashBlockInfo * block,  
    blt_addr address ) [static]
```

Copies data currently in flash to the block->data and sets the base address.

Parameters

<i>block</i>	Pointer to flash block info structure to operate on.
<i>address</i>	Base address of the block data.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

Referenced by FlashAddToBlock(), and FlashSwitchBlock().

7.413.2.10 FlashReinit()

```
void FlashReinit (
    void )
```

Reinitializes the flash driver.

Returns

none.

7.413.2.11 FlashSwitchBlock()

```
static tFlashBlockInfo * FlashSwitchBlock (
    tFlashBlockInfo * block,
    blt_addr base_addr ) [static]
```

Switches blocks by programming the current one and initializing the next.

Parameters

<i>block</i>	Pointer to flash block info structure to operate on.
<i>base_addr</i>	Base address of the next block.

Returns

The pointer of the block info struct that is now being used, or a NULL pointer in case of error.

Referenced by FlashAddToBlock().

7.413.2.12 FlashVerifyChecksum()

```
blt_bool FlashVerifyChecksum (
    void )
```

Verifies the checksum, which indicates that a valid user program is present and can be started.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.413.2.13 FlashWrite()

```
blt_bool FlashWrite (
    blt_addr addr,
    blt_int32u len,
    blt_int8u * data )
```

Writes the data to flash through a flash block manager. Note that this function also checks that no data is programmed outside the flash memory region, so the bootloader can never be overwritten.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.
<i>data</i>	Pointer to the data buffer.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

Referenced by FlashWriteChecksum().

7.413.2.14 FlashWriteBlock()

```
static blt_bool FlashWriteBlock (
    tFlashBlockInfo * block ) [static]
```

Programs FLASH_WRITE_BLOCK_SIZE bytes to flash from the block->data array.

Parameters

<i>block</i>	Pointer to flash block info structure to operate on.
--------------	--

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

Referenced by FlashDone(), and FlashSwitchBlock().

7.413.2.15 FlashWriteChecksum()

```
blt_bool FlashWriteChecksum (  
    void )
```

Writes a checksum of the user program to non-volatile memory. This is performed once the entire user program has been programmed. Through the checksum, the bootloader can check if the programming session was completed, which indicates that a valid user programming is present and can be started.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.413.3 Variable Documentation

7.413.3.1 blockInfo

```
tFlashBlockInfo blockInfo [static]
```

Local variable with information about the flash block that is currently being operated on.

The smallest amount of flash that can be programmed is FLASH_WRITE_BLOCK_SIZE. A flash block manager is implemented in this driver and stores info in this variable. Whenever new data should be flashed, it is first added to a RAM buffer, which is part of this variable. Whenever the RAM buffer, which has the size of a flash block, is full or data needs to be written to a different block, the contents of the RAM buffer are programmed to flash. The flash block manager requires some software overhead, yet results in faster flash programming because data is first harvested, ideally until there is enough to program an entire flash block, before the flash device is actually operated on.

Referenced by FlashSwitchBlock().

7.413.3.2 bootBlockInfo

```
tFlashBlockInfo bootBlockInfo [static]
```

Local variable with information about the flash boot block.

The first block of the user program holds the vector table, which on the STM32 is also the where the checksum is written to. It is likely that the vector table is first flashed and then, at the end of the programming sequence, the checksum. This means that this flash block needs to be written to twice. Normally this is not a problem with flash memory, as long as you write the same values to those bytes that are not supposed to be changed and the locations where you do write to are still in the erased 0xFF state. Unfortunately, writing twice to flash this way, does not work reliably on all micros. This is why we need to have an extra block, the bootblock, placed under the management of the block manager. This way it is possible to implement functionality so that the bootblock is only written to once at the end of the programming sequence.

Referenced by FlashSwitchBlock().

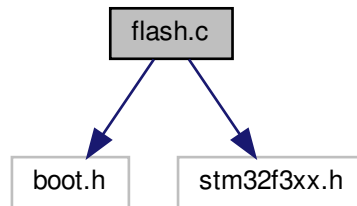
7.414 flash.c File Reference

Bootloader flash driver source file.

```
#include "boot.h"
```

```
#include "stm32f3xx.h"
```

Include dependency graph for ARMCM4_STM32F3/flash.c:



Data Structures

- struct [tFlashSector](#)
Flash sector descriptor type.
- struct [tFlashBlockInfo](#)
Structure type for grouping flash block information.

Macros

- #define [FLASH_INVALID_ADDRESS](#) (0xffffffff)
Value for an invalid flash address.
- #define [FLASH_WRITE_BLOCK_SIZE](#) (512)
Standard size of a flash block for writing.
- #define [FLASH_ERASE_SECTOR_SIZE](#) (2048)
Standard size of a flash sector for erasing.
- #define [FLASH_TOTAL_SEGMENTS](#) (sizeof(flashLayout)/sizeof(flashLayout[0]))
Total numbers of segments in array flashLayout[].
- #define [FLASH_LAST_SEGMENT_IDX](#) (FLASH_TOTAL_SEGMENTS-1)
Index of the last segment in array flashLayout[].
- #define [FLASH_START_ADDRESS](#) (flashLayout[0].sector_start)
Start address of the bootloader programmable flash.
- #define [FLASH_END_ADDRESS](#)
End address of the bootloader programmable flash.
- #define [BOOT_FLASH_VECTOR_TABLE_CS_OFFSET](#) (0x188)
Offset into the user program's vector table where the checksum is located. For this target it is set to the end of the vector table. Note that the value can be overridden in blt_conf.h, because the size of the vector table could vary. When changing this value, don't forget to update the location of the checksum in the user program accordingly. Otherwise the checksum verification will always fail.

Functions

- static `blt_bool FlashInitBlock` (`tFlashBlockInfo *block`, `blt_addr` address)
Copies data currently in flash to the block->data and sets the base address.
- static `tFlashBlockInfo * FlashSwitchBlock` (`tFlashBlockInfo *block`, `blt_addr` base_addr)
Switches blocks by programming the current one and initializing the next.
- static `blt_bool FlashAddToBlock` (`tFlashBlockInfo *block`, `blt_addr` address, `blt_int8u *data`, `blt_int32u` len)
Programming is done per block. This function adds data to the block that is currently collecting data to be written to flash. If the address is outside of the current block, the current block is written to flash and a new block is initialized.
- static `blt_bool FlashWriteBlock` (`tFlashBlockInfo *block`)
Programs FLASH_WRITE_BLOCK_SIZE bytes to flash from the block->data array.
- void `FlashInit` (void)
Initializes the flash driver.
- void `FlashReinit` (void)
Reinitializes the flash driver.
- `blt_bool FlashWrite` (`blt_addr` addr, `blt_int32u` len, `blt_int8u *data`)
Writes the data to flash through a flash block manager. Note that this function also checks that no data is programmed outside the flash memory region, so the bootloader can never be overwritten.
- `blt_bool FlashErase` (`blt_addr` addr, `blt_int32u` len)
Erases the flash memory. Note that this function also checks that no data is erased outside the flash memory region, so the bootloader can never be erased.
- `blt_bool FlashWriteChecksum` (void)
Writes a checksum of the user program to non-volatile memory. This is performed once the entire user program has been programmed. Through the checksum, the bootloader can check if the programming session was completed, which indicates that a valid user programming is present and can be started.
- `blt_bool FlashVerifyChecksum` (void)
Verifies the checksum, which indicates that a valid user program is present and can be started.
- `blt_bool FlashDone` (void)
Finalizes the flash driver operations. There could still be data in the currently active block that needs to be flashed.
- `blt_addr FlashGetUserProgBaseAddress` (void)
Obtains the base address of the flash memory available to the user program. This is basically the first address in the flashLayout table.

Variables

- static const `tFlashSector flashLayout` []
If desired, it is possible to set BOOT_FLASH_CUSTOM_LAYOUT_ENABLE to > 0 in blt_conf.h and then implement your own version of the flashLayout[] table in a source-file with the name flash_layout.c. This way you customize the flash memory size reserved for the bootloader, without having to modify the flashLayout[] table in this file directly. This file will then include flash_layout.c so there is no need to compile it additionally with your project.
- static `tFlashBlockInfo blockInfo`
Local variable with information about the flash block that is currently being operated on.
- static `tFlashBlockInfo bootBlockInfo`
Local variable with information about the flash boot block.

7.414.1 Detailed Description

Bootloader flash driver source file.

7.414.2 Function Documentation

7.414.2.1 FlashAddToBlock()

```
static blt_bool FlashAddToBlock (
    tFlashBlockInfo * block,
    blt_addr address,
    blt_int8u * data,
    blt_int32u len ) [static]
```

Programming is done per block. This function adds data to the block that is currently collecting data to be written to flash. If the address is outside of the current block, the current block is written to flash and a new block is initialized.

Parameters

<i>block</i>	Pointer to flash block info structure to operate on.
<i>address</i>	Flash destination address.
<i>data</i>	Pointer to the byte array with data.
<i>len</i>	Number of bytes to add to the block.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

Referenced by FlashWrite().

7.414.2.2 FlashDone()

```
blt_bool FlashDone (
    void )
```

Finalizes the flash driver operations. There could still be data in the currently active block that needs to be flashed.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.414.2.3 FlashErase()

```
blt_bool FlashErase (
    blt_addr addr,
    blt_int32u len )
```

Erases the flash memory. Note that this function also checks that no data is erased outside the flash memory region, so the bootloader can never be erased.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.414.2.4 FlashGetUserProgBaseAddress()

```
bld_addr FlashGetUserProgBaseAddress (
    void )
```

Obtains the base address of the flash memory available to the user program. This is basically the first address in the flashLayout table.

Returns

Base address.

7.414.2.5 FlashInit()

```
void FlashInit (
    void )
```

Initializes the flash driver.

Returns

none.

7.414.2.6 FlashInitBlock()

```
static bld_bool FlashInitBlock (
    tFlashBlockInfo * block,
    bld_addr address ) [static]
```

Copies data currently in flash to the block->data and sets the base address.

Parameters

<i>block</i>	Pointer to flash block info structure to operate on.
<i>address</i>	Base address of the block data.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

Referenced by FlashAddToBlock(), and FlashSwitchBlock().

7.414.2.7 FlashReinit()

```
void FlashReinit (
    void )
```

Reinitializes the flash driver.

Returns

none.

7.414.2.8 FlashSwitchBlock()

```
static tFlashBlockInfo * FlashSwitchBlock (
    tFlashBlockInfo * block,
    blt_addr base_addr ) [static]
```

Switches blocks by programming the current one and initializing the next.

Parameters

<i>block</i>	Pointer to flash block info structure to operate on.
<i>base_addr</i>	Base address of the next block.

Returns

The pointer of the block info struct that is no being used, or a NULL pointer in case of error.

Referenced by FlashAddToBlock().

7.414.2.9 FlashVerifyChecksum()

```
blt_bool FlashVerifyChecksum (
    void )
```

Verifies the checksum, which indicates that a valid user program is present and can be started.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.414.2.10 FlashWrite()

```
blt_bool FlashWrite (
    blt_addr addr,
    blt_int32u len,
    blt_int8u * data )
```

Writes the data to flash through a flash block manager. Note that this function also checks that no data is programmed outside the flash memory region, so the bootloader can never be overwritten.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.
<i>data</i>	Pointer to the data buffer.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

Referenced by FlashWriteChecksum().

7.414.2.11 FlashWriteBlock()

```
static blt_bool FlashWriteBlock (
    tFlashBlockInfo * block ) [static]
```

Programs FLASH_WRITE_BLOCK_SIZE bytes to flash from the block->data array.

Parameters

<i>block</i>	Pointer to flash block info structure to operate on.
--------------	--

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

Referenced by FlashDone(), and FlashSwitchBlock().

7.414.2.12 FlashWriteChecksum()

```
blt_bool FlashWriteChecksum (  
    void )
```

Writes a checksum of the user program to non-volatile memory. This is performed once the entire user program has been programmed. Through the checksum, the bootloader can check if the programming session was completed, which indicates that a valid user programming is present and can be started.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.414.3 Variable Documentation

7.414.3.1 blockInfo

```
tFlashBlockInfo blockInfo [static]
```

Local variable with information about the flash block that is currently being operated on.

The smallest amount of flash that can be programmed is FLASH_WRITE_BLOCK_SIZE. A flash block manager is implemented in this driver and stores info in this variable. Whenever new data should be flashed, it is first added to a RAM buffer, which is part of this variable. Whenever the RAM buffer, which has the size of a flash block, is full or data needs to be written to a different block, the contents of the RAM buffer are programmed to flash. The flash block manager requires some software overhead, yet results in faster flash programming because data is first harvested, ideally until there is enough to program an entire flash block, before the flash device is actually operated on.

Referenced by FlashSwitchBlock().

7.414.3.2 bootBlockInfo

```
tFlashBlockInfo bootBlockInfo [static]
```

Local variable with information about the flash boot block.

The first block of the user program holds the vector table, which on the STM32 is also the where the checksum is written to. It is likely that the vector table is first flashed and then, at the end of the programming sequence, the checksum. This means that this flash block needs to be written to twice. Normally this is not a problem with flash memory, as long as you write the same values to those bytes that are not supposed to be changed and the locations where you do write to are still in the erased 0xFF state. Unfortunately, writing twice to flash this way, does not work reliably on all micros. This is why we need to have an extra block, the bootblock, placed under the management of the block manager. This way it is possible to implement functionality so that the bootblock is only written to once at the end of the programming sequence.

Referenced by FlashSwitchBlock().

7.414.3.3 flashLayout

```
const tFlashSector flashLayout[] [static]
```

If desired, it is possible to set `BOOT_FLASH_CUSTOM_LAYOUT_ENABLE` to `> 0` in `blt_conf.h` and then implement your own version of the `flashLayout[]` table in a source-file with the name `flash_layout.c`. This way you customize the flash memory size reserved for the bootloader, without having to modify the `flashLayout[]` table in this file directly. This file will then include `flash_layout.c` so there is no need to compile it additionally with your project.

Array with the layout of the flash memory.

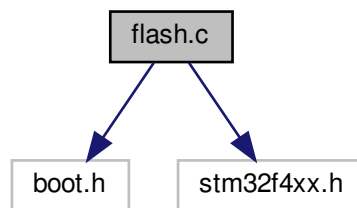
Also controls what part of the flash memory is reserved for the bootloader. If the bootloader size changes, the reserved sectors for the bootloader might need adjustment to make sure the bootloader doesn't get overwritten.

7.415 flash.c File Reference

Bootloader flash driver source file.

```
#include "boot.h"  
#include "stm32f4xx.h"
```

Include dependency graph for `ARMCM4_STM32F4/flash.c`:



Data Structures

- struct `tFlashSector`
Flash sector descriptor type.
- struct `tFlashBlockInfo`
Structure type for grouping flash block information.

Macros

- `#define FLASH_INVALID_SECTOR (0xff)`
Value for an invalid flash sector.
- `#define FLASH_INVALID_ADDRESS (0xffffffff)`
Value for an invalid flash address.
- `#define FLASH_WRITE_BLOCK_SIZE (512)`
Standard size of a flash block for writing.
- `#define FLASH_TOTAL_SECTORS (sizeof(flashLayout)/sizeof(flashLayout[0]))`
Total numbers of sectors in array flashLayout[].
- `#define FLASH_END_ADDRESS`
End address of the bootloader programmable flash.
- `#define BOOT_FLASH_VECTOR_TABLE_CS_OFFSET (0x188)`
Offset into the user program's vector table where the checksum is located. For this target it is set to the end of the vector table. Note that the value can be overridden in blt_conf.h, because the size of the vector table could vary. When changing this value, don't forget to update the location of the checksum in the user program accordingly. Otherwise the checksum verification will always fail.

Functions

- `static blt_bool FlashInitBlock (tFlashBlockInfo *block, blt_addr address)`
Copies data currently in flash to the block->data and sets the base address.
- `static tFlashBlockInfo * FlashSwitchBlock (tFlashBlockInfo *block, blt_addr base_addr)`
Switches blocks by programming the current one and initializing the next.
- `static blt_bool FlashAddToBlock (tFlashBlockInfo *block, blt_addr address, blt_int8u *data, blt_int32u len)`
Programming is done per block. This function adds data to the block that is currently collecting data to be written to flash. If the address is outside of the current block, the current block is written to flash and a new block is initialized.
- `static blt_bool FlashWriteBlock (tFlashBlockInfo *block)`
Programs FLASH_WRITE_BLOCK_SIZE bytes to flash from the block->data array.
- `static blt_bool FlashEraseSectors (blt_int8u first_sector, blt_int8u last_sector)`
Erases the flash sectors from first_sector up until last_sector.
- `static blt_int8u FlashGetSector (blt_addr address)`
Determines the flash sector the address is in.
- `void FlashInit (void)`
Initializes the flash driver.
- `void FlashReinit (void)`
Reinitializes the flash driver.
- `blt_bool FlashWrite (blt_addr addr, blt_int32u len, blt_int8u *data)`
Writes the data to flash through a flash block manager. Note that this function also checks that no data is programmed outside the flash memory region, so the bootloader can never be overwritten.
- `blt_bool FlashErase (blt_addr addr, blt_int32u len)`
Erases the flash memory. Note that this function also checks that no data is erased outside the flash memory region, so the bootloader can never be erased.
- `blt_bool FlashWriteChecksum (void)`
Writes a checksum of the user program to non-volatile memory. This is performed once the entire user program has been programmed. Through the checksum, the bootloader can check if the programming session was completed, which indicates that a valid user programming is present and can be started.
- `blt_bool FlashVerifyChecksum (void)`
Verifies the checksum, which indicates that a valid user program is present and can be started.
- `blt_bool FlashDone (void)`
Finalizes the flash driver operations. There could still be data in the currently active block that needs to be flashed.
- `blt_addr FlashGetUserProgBaseAddress (void)`
Obtains the base address of the flash memory available to the user program. This is basically the first address in the flashLayout table.

Variables

- static const [tFlashSector flashLayout \[\]](#)

If desired, it is possible to set `BOOT_FLASH_CUSTOM_LAYOUT_ENABLE` to `> 0` in `blt_conf.h` and then implement your own version of the `flashLayout[]` table in a source-file with the name `flash_layout.c`. This way you customize the flash memory size reserved for the bootloader, without having to modify the `flashLayout[]` table in this file directly. This file will then include `flash_layout.c` so there is no need to compile it additionally with your project.

- static [tFlashBlockInfo blockInfo](#)

Local variable with information about the flash block that is currently being operated on.

- static [tFlashBlockInfo bootBlockInfo](#)

Local variable with information about the flash boot block.

7.415.1 Detailed Description

Bootloader flash driver source file.

7.415.2 Function Documentation

7.415.2.1 FlashAddToBlock()

```
static blt\_bool FlashAddToBlock (  
    tFlashBlockInfo * block,  
    blt\_addr address,  
    blt\_int8u * data,  
    blt\_int32u len ) [static]
```

Programming is done per block. This function adds data to the block that is currently collecting data to be written to flash. If the address is outside of the current block, the current block is written to flash and a new block is initialized.

Parameters

<i>block</i>	Pointer to flash block info structure to operate on.
<i>address</i>	Flash destination address.
<i>data</i>	Pointer to the byte array with data.
<i>len</i>	Number of bytes to add to the block.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

Referenced by `FlashWrite()`.

7.415.2.2 FlashDone()

```
blt_bool FlashDone (
    void )
```

Finalizes the flash driver operations. There could still be data in the currently active block that needs to be flashed.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.415.2.3 FlashErase()

```
blt_bool FlashErase (
    blt_addr addr,
    blt_int32u len )
```

Erases the flash memory. Note that this function also checks that no data is erased outside the flash memory region, so the bootloader can never be erased.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.415.2.4 FlashEraseSectors()

```
static blt_bool FlashEraseSectors (
    blt_int8u first_sector,
    blt_int8u last_sector ) [static]
```

Erases the flash sectors from *first_sector* up until *last_sector*.

Parameters

<i>first_sector</i>	First flash sector number.
<i>last_sector</i>	Last flash sector number.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

Referenced by FlashErase().

7.415.2.5 FlashGetSector()

```
static blt_int8u FlashGetSector (
    blt_addr address ) [static]
```

Determines the flash sector the address is in.

Parameters

<i>address</i>	Address in the flash sector.
----------------	------------------------------

Returns

Flash sector number or FLASH_INVALID_SECTOR.

Referenced by FlashErase(), and FlashWrite().

7.415.2.6 FlashGetUserProgBaseAddress()

```
blt_addr FlashGetUserProgBaseAddress (
    void )
```

Obtains the base address of the flash memory available to the user program. This is basically the first address in the flashLayout table.

Returns

Base address.

7.415.2.7 FlashInit()

```
void FlashInit (
    void )
```

Initializes the flash driver.

Returns

none.

7.415.2.8 FlashInitBlock()

```
static blt_bool FlashInitBlock (
    tFlashBlockInfo * block,
    blt_addr address ) [static]
```

Copies data currently in flash to the block->data and sets the base address.

Parameters

<i>block</i>	Pointer to flash block info structure to operate on.
<i>address</i>	Base address of the block data.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

Referenced by FlashAddToBlock(), and FlashSwitchBlock().

7.415.2.9 FlashReinit()

```
void FlashReinit (
    void )
```

Reinitializes the flash driver.

Returns

none.

7.415.2.10 FlashSwitchBlock()

```
static tFlashBlockInfo * FlashSwitchBlock (
    tFlashBlockInfo * block,
    blt_addr base_addr ) [static]
```

Switches blocks by programming the current one and initializing the next.

Parameters

<i>block</i>	Pointer to flash block info structure to operate on.
<i>base_addr</i>	Base address of the next block.

Returns

The pointer of the block info struct that is no being used, or a NULL pointer in case of error.

Referenced by FlashAddToBlock().

7.415.2.11 FlashVerifyChecksum()

```
blt_bool FlashVerifyChecksum (
    void )
```

Verifies the checksum, which indicates that a valid user program is present and can be started.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.415.2.12 FlashWrite()

```
blt_bool FlashWrite (
    blt_addr addr,
    blt_int32u len,
    blt_int8u * data )
```

Writes the data to flash through a flash block manager. Note that this function also checks that no data is programmed outside the flash memory region, so the bootloader can never be overwritten.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.
<i>data</i>	Pointer to the data buffer.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

Referenced by FlashWriteChecksum().

7.415.2.13 FlashWriteBlock()

```
static blt_bool FlashWriteBlock (
    tFlashBlockInfo * block ) [static]
```

Programs FLASH_WRITE_BLOCK_SIZE bytes to flash from the block->data array.

Parameters

<i>block</i>	Pointer to flash block info structure to operate on.
--------------	--

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

Referenced by FlashDone(), and FlashSwitchBlock().

7.415.2.14 FlashWriteChecksum()

```
blt_bool FlashWriteChecksum (  
    void )
```

Writes a checksum of the user program to non-volatile memory. This is performed once the entire user program has been programmed. Through the checksum, the bootloader can check if the programming session was completed, which indicates that a valid user programming is present and can be started.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.415.3 Variable Documentation

7.415.3.1 blockInfo

```
tFlashBlockInfo blockInfo [static]
```

Local variable with information about the flash block that is currently being operated on.

The smallest amount of flash that can be programmed is FLASH_WRITE_BLOCK_SIZE. A flash block manager is implemented in this driver and stores info in this variable. Whenever new data should be flashed, it is first added to a RAM buffer, which is part of this variable. Whenever the RAM buffer, which has the size of a flash block, is full or data needs to be written to a different block, the contents of the RAM buffer are programmed to flash. The flash block manager requires some software overhead, yet results in faster flash programming because data is first harvested, ideally until there is enough to program an entire flash block, before the flash device is actually operated on.

Referenced by FlashSwitchBlock().

7.415.3.2 bootBlockInfo

```
tFlashBlockInfo bootBlockInfo [static]
```

Local variable with information about the flash boot block.

The first block of the user program holds the vector table, which on the STM32 is also the where the checksum is written to. It is likely that the vector table is first flashed and then, at the end of the programming sequence, the checksum. This means that this flash block needs to be written to twice. Normally this is not a problem with flash memory, as long as you write the same values to those bytes that are not supposed to be changed and the locations where you do write to are still in the erased 0xFF state. Unfortunately, writing twice to flash this way, does not work reliably on all micros. This is why we need to have an extra block, the bootblock, placed under the management of the block manager. This way it is possible to implement functionality so that the bootblock is only written to once at the end of the programming sequence.

Referenced by FlashSwitchBlock().

7.415.3.3 flashLayout

```
const tFlashSector flashLayout[] [static]
```

If desired, it is possible to set `BOOT_FLASH_CUSTOM_LAYOUT_ENABLE` to `> 0` in `blt_conf.h` and then implement your own version of the `flashLayout[]` table in a source-file with the name `flash_layout.c`. This way you customize the flash memory size reserved for the bootloader, without having to modify the `flashLayout[]` table in this file directly. This file will then include `flash_layout.c` so there is no need to compile it additionally with your project.

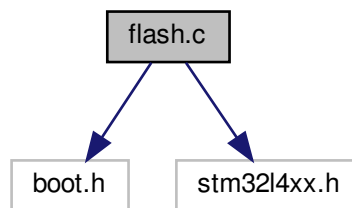
Array with the layout of the flash memory.

Also controls what part of the flash memory is reserved for the bootloader. If the bootloader size changes, the reserved sectors for the bootloader might need adjustment to make sure the bootloader doesn't get overwritten.

7.416 flash.c File Reference

Bootloader flash driver source file.

```
#include "boot.h"
#include "stm32l4xx.h"
Include dependency graph for ARMC4_STM32L4/flash.c:
```



Data Structures

- struct `tFlashSector`
Flash sector descriptor type.
- struct `tFlashBlockInfo`
Structure type for grouping flash block information.

Macros

- `#define FLASH_INVALID_ADDRESS (0xffffffff)`
Value for an invalid flash address.
- `#define FLASH_WRITE_BLOCK_SIZE (512)`
Standard size of a flash block for writing.
- `#define FLASH_ERASE_SECTOR_SIZE (2048)`
Standard size of a flash sector for erasing.
- `#define FLASH_TOTAL_SEGMENTS (sizeof(flashLayout)/sizeof(flashLayout[0]))`
Total numbers of segments in array flashLayout[].
- `#define FLASH_LAST_SEGMENT_IDX (FLASH_TOTAL_SEGMENTS-1)`
Index of the last segment in array flashLayout[].
- `#define FLASH_START_ADDRESS (flashLayout[0].sector_start)`
Start address of the bootloader programmable flash.
- `#define FLASH_END_ADDRESS`
End address of the bootloader programmable flash.
- `#define BOOT_FLASH_VECTOR_TABLE_CS_OFFSET (0x188)`
Offset into the user program's vector table where the checksum is located. For this target it is set to the end of the vector table. Note that the value can be overridden in blt_conf.h, because the size of the vector table could vary. When changing this value, don't forget to update the location of the checksum in the user program accordingly. Otherwise the checksum verification will always fail.

Functions

- static `blt_bool FlashInitBlock (tFlashBlockInfo *block, blt_addr address)`
Copies data currently in flash to the block->data and sets the base address.
- static `tFlashBlockInfo * FlashSwitchBlock (tFlashBlockInfo *block, blt_addr base_addr)`
Switches blocks by programming the current one and initializing the next.
- static `blt_bool FlashAddToBlock (tFlashBlockInfo *block, blt_addr address, blt_int8u *data, blt_int32u len)`
Programming is done per block. This function adds data to the block that is currently collecting data to be written to flash. If the address is outside of the current block, the current block is written to flash and a new block is initialized.
- static `blt_bool FlashWriteBlock (tFlashBlockInfo *block)`
Programs FLASH_WRITE_BLOCK_SIZE bytes to flash from the block->data array.
- static `blt_int32u FlashGetPage (blt_addr address)`
Gets the page number of the address relative to the bank.
- static `blt_int32u FlashGetBank (blt_addr address)`
Obtains the bank of the given address. The 1024kb version of the flash device contains 2 banks that can be swapped. This feature breaks the link between a bank number and flash addresses. This function obtains the bank number that is currently at the given address.
- void `FlashInit (void)`
Initializes the flash driver.
- void `FlashReinit (void)`
Reinitializes the flash driver.
- `blt_bool FlashWrite (blt_addr addr, blt_int32u len, blt_int8u *data)`
Writes the data to flash through a flash block manager. Note that this function also checks that no data is programmed outside the flash memory region, so the bootloader can never be overwritten.
- `blt_bool FlashErase (blt_addr addr, blt_int32u len)`
Erases the flash memory. Note that this function also checks that no data is erased outside the flash memory region, so the bootloader can never be erased. Note that the term sector used by this flash driver is equivalent to the term page in the STM32L4x reference manual.
- `blt_bool FlashWriteChecksum (void)`

Writes a checksum of the user program to non-volatile memory. This is performed once the entire user program has been programmed. Through the checksum, the bootloader can check if the programming session was completed, which indicates that a valid user programming is present and can be started.

- [blt_bool FlashVerifyChecksum](#) (void)

Verifies the checksum, which indicates that a valid user program is present and can be started.

- [blt_bool FlashDone](#) (void)

Finalizes the flash driver operations. There could still be data in the currently active block that needs to be flashed.

- [blt_addr FlashGetUserProgBaseAddress](#) (void)

Obtains the base address of the flash memory available to the user program. This is basically the first address in the flashLayout table.

Variables

- static const [tFlashSector flashLayout](#) []

If desired, it is possible to set `BOOT_FLASH_CUSTOM_LAYOUT_ENABLE` to `> 0` in `blt_conf.h` and then implement your own version of the `flashLayout[]` table in a source-file with the name `flash_layout.c`. This way you customize the flash memory size reserved for the bootloader, without having to modify the `flashLayout[]` table in this file directly. This file will then include `flash_layout.c` so there is no need to compile it additionally with your project.

- static [tFlashBlockInfo blockInfo](#)

Local variable with information about the flash block that is currently being operated on.

- static [tFlashBlockInfo bootBlockInfo](#)

Local variable with information about the flash boot block.

7.416.1 Detailed Description

Bootloader flash driver source file.

7.416.2 Function Documentation

7.416.2.1 FlashAddToBlock()

```
static blt_bool FlashAddToBlock (
    tFlashBlockInfo * block,
    blt_addr address,
    blt_int8u * data,
    blt_int32u len ) [static]
```

Programming is done per block. This function adds data to the block that is currently collecting data to be written to flash. If the address is outside of the current block, the current block is written to flash and a new block is initialized.

Parameters

<i>block</i>	Pointer to flash block info structure to operate on.
<i>address</i>	Flash destination address.
<i>data</i>	Pointer to the byte array with data.
<i>len</i>	Number of bytes to add to the block.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

Referenced by FlashWrite().

7.416.2.2 FlashDone()

```
blt_bool FlashDone (  
    void )
```

Finalizes the flash driver operations. There could still be data in the currently active block that needs to be flashed.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.416.2.3 FlashErase()

```
blt_bool FlashErase (  
    blt_addr addr,  
    blt_int32u len )
```

Erases the flash memory. Note that this function also checks that no data is erased outside the flash memory region, so the bootloader can never be erased. Note that the term sector used by this flash driver is equivalent to the term page in the STM32L4x reference manual.

Erases the flash memory. Note that this function also checks that no data is erased outside the flash memory region, so the bootloader can never be erased.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.416.2.4 FlashGetBank()

```
static blt_int32u FlashGetBank (  
    blt_addr address ) [static]
```


Obtains the bank of the given address. The 1024kb version of the flash device contains 2 banks that can be swapped. This feature breaks the link between a bank number and flash addresses. This function obtains the bank number that is currently at the given address.

Parameters

<i>address</i>	Address in the flash bank.
----------------	----------------------------

Returns

The flash bank of the given address: FLASH_BANK_1 or FLASH_BANK_2.

Referenced by FlashErase().

7.416.2.5 FlashGetPage()

```
static blt_int32u FlashGetPage (
    blt_addr address ) [static]
```

Gets the page number of the address relative to the bank.

Parameters

<i>address</i>	Address in the flash bank.
----------------	----------------------------

Returns

The page of the given address: 0..255.

Referenced by FlashErase().

7.416.2.6 FlashGetUserProgBaseAddress()

```
blt_addr FlashGetUserProgBaseAddress (
    void )
```

Obtains the base address of the flash memory available to the user program. This is basically the first address in the flashLayout table.

Returns

Base address.

7.416.2.7 FlashInit()

```
void FlashInit (
    void )
```

Initializes the flash driver.

Returns

none.

7.416.2.8 FlashInitBlock()

```
static blt_bool FlashInitBlock (
    tFlashBlockInfo * block,
    blt_addr address ) [static]
```

Copies data currently in flash to the block->data and sets the base address.

Parameters

<i>block</i>	Pointer to flash block info structure to operate on.
<i>address</i>	Base address of the block data.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

Referenced by FlashAddToBlock(), and FlashSwitchBlock().

7.416.2.9 FlashReinit()

```
void FlashReinit (
    void )
```

Reinitializes the flash driver.

Returns

none.

7.416.2.10 FlashSwitchBlock()

```
static tFlashBlockInfo * FlashSwitchBlock (
    tFlashBlockInfo * block,
    blt_addr base_addr ) [static]
```

Switches blocks by programming the current one and initializing the next.

Parameters

<i>block</i>	Pointer to flash block info structure to operate on.
<i>base_addr</i>	Base address of the next block.

Returns

The pointer of the block info struct that is no being used, or a NULL pointer in case of error.

Referenced by FlashAddToBlock().

7.416.2.11 FlashVerifyChecksum()

```
blt_bool FlashVerifyChecksum (
    void )
```

Verifies the checksum, which indicates that a valid user program is present and can be started.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.416.2.12 FlashWrite()

```
blt_bool FlashWrite (
    blt_addr addr,
    blt_int32u len,
    blt_int8u * data )
```

Writes the data to flash through a flash block manager. Note that this function also checks that no data is programmed outside the flash memory region, so the bootloader can never be overwritten.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.
<i>data</i>	Pointer to the data buffer.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

Referenced by FlashWriteChecksum().

7.416.2.13 FlashWriteBlock()

```
static blt_bool FlashWriteBlock (
    tFlashBlockInfo * block ) [static]
```

Programs FLASH_WRITE_BLOCK_SIZE bytes to flash from the block->data array.

Parameters

<i>block</i>	Pointer to flash block info structure to operate on.
--------------	--

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

Referenced by FlashDone(), and FlashSwitchBlock().

7.416.2.14 FlashWriteChecksum()

```
blt_bool FlashWriteChecksum (
    void )
```

Writes a checksum of the user program to non-volatile memory. This is performed once the entire user program has been programmed. Through the checksum, the bootloader can check if the programming session was completed, which indicates that a valid user programming is present and can be started.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.416.3 Variable Documentation

7.416.3.1 blockInfo

```
tFlashBlockInfo blockInfo [static]
```

Local variable with information about the flash block that is currently being operated on.

The smallest amount of flash that can be programmed is FLASH_WRITE_BLOCK_SIZE. A flash block manager is implemented in this driver and stores info in this variable. Whenever new data should be flashed, it is first added to a RAM buffer, which is part of this variable. Whenever the RAM buffer, which has the size of a flash block, is full or data needs to be written to a different block, the contents of the RAM buffer are programmed to flash. The flash block manager requires some software overhead, yet results in faster flash programming because data is first harvested, ideally until there is enough to program an entire flash block, before the flash device is actually operated on.

Referenced by FlashSwitchBlock().

7.416.3.2 bootBlockInfo

```
tFlashBlockInfo bootBlockInfo [static]
```

Local variable with information about the flash boot block.

The first block of the user program holds the vector table, which on the STM32 is also the where the checksum is written to. Is it likely that the vector table is first flashed and then, at the end of the programming sequence, the checksum. This means that this flash block need to be written to twice. Normally this is not a problem with flash memory, as long as you write the same values to those bytes that are not supposed to be changed and the locations where you do write to are still in the erased 0xFF state. Unfortunately, writing twice to flash this way, does not work reliably on all micros. This is why we need to have an extra block, the bootblock, placed under the management of the block manager. This way is it possible to implement functionality so that the bootblock is only written to once at the end of the programming sequence.

Referenced by FlashSwitchBlock().

7.416.3.3 flashLayout

```
const tFlashSector flashLayout[] [static]
```

If desired, it is possible to set `BOOT_FLASH_CUSTOM_LAYOUT_ENABLE` to `> 0` in `blt_conf.h` and then implement your own version of the `flashLayout[]` table in a source-file with the name `flash_layout.c`. This way you customize the flash memory size reserved for the bootloader, without having to modify the `flashLayout[]` table in this file directly. This file will then include `flash_layout.c` so there is no need to compile it additionally with your project.

Array with the layout of the flash memory. Note that the current flash driver supports the STM32L4x1, STM32L4x5 and STM32L4x6 derivatives in the STM32L4 family of microcontrollers.

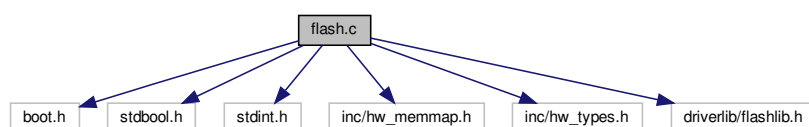
Also controls what part of the flash memory is reserved for the bootloader. If the bootloader size changes, the reserved sectors for the bootloader might need adjustment to make sure the bootloader doesn't get overwritten.

7.417 flash.c File Reference

Bootloader flash driver source file.

```
#include "boot.h"
#include <stdbool.h>
#include <stdint.h>
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "driverlib/flashlib.h"
```

Include dependency graph for ARMCM4_TM4C/flash.c:



Data Structures

- struct [tFlashSector](#)
Flash sector descriptor type.
- struct [tFlashBlockInfo](#)
Structure type for grouping flash block information.

Macros

- #define [FLASH_INVALID_SECTOR](#) (0xff)
Value for an invalid flash sector.
- #define [FLASH_INVALID_ADDRESS](#) (0xffffffff)
Value for an invalid flash address.
- #define [FLASH_WRITE_BLOCK_SIZE](#) (1024)
Standard size of a flash block for writing.
- #define [FLASH_TOTAL_SECTORS](#) (sizeof(flashLayout)/sizeof(flashLayout[0]))
Total numbers of sectors in array flashLayout[].
- #define [FLASH_END_ADDRESS](#)
End address of the bootloader programmable flash.
- #define [FLASH_ERASE_BLOCK_SIZE](#) (0x400)
Number of bytes to erase per erase operation.
- #define [BOOT_FLASH_VECTOR_TABLE_CS_OFFSET](#) (0x26C)
Offset into the user program's vector table where the checksum is located. For this target it is set to the end of the vector table. Note that the value can be overridden in blt_conf.h, because the size of the vector table could vary. When changing this value, don't forget to update the location of the checksum in the user program accordingly. Otherwise the checksum verification will always fail.

Functions

- static [blt_bool](#) [FlashInitBlock](#) ([tFlashBlockInfo](#) *block, [blt_addr](#) address)
Copies data currently in flash to the block->data and sets the base address.
- static [tFlashBlockInfo](#) * [FlashSwitchBlock](#) ([tFlashBlockInfo](#) *block, [blt_addr](#) base_addr)
Switches blocks by programming the current one and initializing the next.
- static [blt_bool](#) [FlashAddToBlock](#) ([tFlashBlockInfo](#) *block, [blt_addr](#) address, [blt_int8u](#) *data, [blt_int32u](#) len)
Programming is done per block. This function adds data to the block that is currently collecting data to be written to flash. If the address is outside of the current block, the current block is written to flash and a new block is initialized.
- static [blt_bool](#) [FlashWriteBlock](#) ([tFlashBlockInfo](#) *block)
Programs FLASH_WRITE_BLOCK_SIZE bytes to flash from the block->data array.
- static [blt_bool](#) [FlashEraseSectors](#) ([blt_int8u](#) first_sector, [blt_int8u](#) last_sector)
Erases the flash sectors from first_sector up until last_sector.
- static [blt_int8u](#) [FlashGetSector](#) ([blt_addr](#) address)
Determines the flash sector the address is in.
- static [blt_addr](#) [FlashGetSectorBaseAddr](#) ([blt_int8u](#) sector)
Determines the flash sector base address.
- static [blt_addr](#) [FlashGetSectorSize](#) ([blt_int8u](#) sector)
Determines the flash sector size.
- void [FlashInit](#) (void)
Initializes the flash driver.
- void [FlashReinit](#) (void)
Reinitializes the flash driver.

- `blt_bool FlashWrite (blt_addr addr, blt_int32u len, blt_int8u *data)`
Writes the data to flash through a flash block manager. Note that this function also checks that no data is programmed outside the flash memory region, so the bootloader can never be overwritten.
- `blt_bool FlashErase (blt_addr addr, blt_int32u len)`
Erases the flash memory. Note that this function also checks that no data is erased outside the flash memory region, so the bootloader can never be erased.
- `blt_bool FlashWriteChecksum (void)`
Writes a checksum of the user program to non-volatile memory. This is performed once the entire user program has been programmed. Through the checksum, the bootloader can check if the programming session was completed, which indicates that a valid user programming is present and can be started.
- `blt_bool FlashVerifyChecksum (void)`
Verifies the checksum, which indicates that a valid user program is present and can be started.
- `blt_bool FlashDone (void)`
Finalizes the flash driver operations. There could still be data in the currently active block that needs to be flashed.
- `blt_addr FlashGetUserProgBaseAddress (void)`
Obtains the base address of the flash memory available to the user program. This is basically the first address in the flashLayout table.

Variables

- static const `tFlashSector flashLayout []`
If desired, it is possible to set `BOOT_FLASH_CUSTOM_LAYOUT_ENABLE` to `> 0` in `blt_conf.h` and then implement your own version of the `flashLayout[]` table in a source-file with the name `flash_layout.c`. This way you customize the flash memory size reserved for the bootloader, without having to modify the `flashLayout[]` table in this file directly. This file will then include `flash_layout.c` so there is no need to compile it additionally with your project.
- static `tFlashBlockInfo blockInfo`
Local variable with information about the flash block that is currently being operated on.
- static `tFlashBlockInfo bootBlockInfo`
Local variable with information about the flash boot block.

7.417.1 Detailed Description

Bootloader flash driver source file.

7.417.2 Function Documentation

7.417.2.1 FlashAddToBlock()

```
static blt_bool FlashAddToBlock (
    tFlashBlockInfo * block,
    blt_addr address,
    blt_int8u * data,
    blt_int32u len ) [static]
```

Programming is done per block. This function adds data to the block that is currently collecting data to be written to flash. If the address is outside of the current block, the current block is written to flash and a new block is initialized.

Parameters

<i>block</i>	Pointer to flash block info structure to operate on.
<i>address</i>	Flash destination address.
<i>data</i>	Pointer to the byte array with data.
<i>len</i>	Number of bytes to add to the block.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

Referenced by FlashWrite().

7.417.2.2 FlashDone()

```
blt_bool FlashDone (  
    void )
```

Finalizes the flash driver operations. There could still be data in the currently active block that needs to be flashed.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.417.2.3 FlashErase()

```
blt_bool FlashErase (  
    blt_addr addr,  
    blt_int32u len )
```

Erases the flash memory. Note that this function also checks that no data is erased outside the flash memory region, so the bootloader can never be erased.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.417.2.4 FlashEraseSectors()

```
static blt_bool FlashEraseSectors (  
    blt_int8u first_sector,  
    blt_int8u last_sector ) [static]
```

Erases the flash sectors from first_sector up until last_sector.

Parameters

<i>first_sector</i>	First flash sector number.
<i>last_sector</i>	Last flash sector number.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

Referenced by FlashErase().

7.417.2.5 FlashGetSector()

```
static blt_int8u FlashGetSector (  
    blt_addr address ) [static]
```

Determines the flash sector the address is in.

Parameters

<i>address</i>	Address in the flash sector.
----------------	------------------------------

Returns

Flash sector number or FLASH_INVALID_SECTOR.

Referenced by FlashErase(), FlashWrite(), and FlashWriteBlock().

7.417.2.6 FlashGetSectorBaseAddr()

```
static blt_addr FlashGetSectorBaseAddr (  
    blt_int8u sector ) [static]
```

Determines the flash sector base address.

Parameters

<i>sector</i>	Sector to get the base address of.
---------------	------------------------------------

Returns

Flash sector base address or FLASH_INVALID_ADDRESS.

Referenced by FlashEraseSectors().

7.417.2.7 FlashGetSectorSize()

```
static blt_addr FlashGetSectorSize (  
    blt_int8u sector ) [static]
```

Determines the flash sector size.

Parameters

<i>sector</i>	Sector to get the size of.
---------------	----------------------------

Returns

Flash sector size or 0.

Referenced by FlashEraseSectors().

7.417.2.8 FlashGetUserProgBaseAddress()

```
blt_addr FlashGetUserProgBaseAddress (  
    void )
```

Obtains the base address of the flash memory available to the user program. This is basically the first address in the flashLayout table.

Returns

Base address.

7.417.2.9 FlashInit()

```
void FlashInit (  
    void )
```

Initializes the flash driver.

Returns

none.

7.417.2.10 FlashInitBlock()

```
static blt_bool FlashInitBlock (
    tFlashBlockInfo * block,
    blt_addr address ) [static]
```

Copies data currently in flash to the block->data and sets the base address.

Parameters

<i>block</i>	Pointer to flash block info structure to operate on.
<i>address</i>	Base address of the block data.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

Referenced by FlashAddToBlock(), and FlashSwitchBlock().

7.417.2.11 FlashReinit()

```
void FlashReinit (
    void )
```

Reinitializes the flash driver.

Returns

none.

7.417.2.12 FlashSwitchBlock()

```
static tFlashBlockInfo * FlashSwitchBlock (
    tFlashBlockInfo * block,
    blt_addr base_addr ) [static]
```

Switches blocks by programming the current one and initializing the next.

Parameters

<i>block</i>	Pointer to flash block info structure to operate on.
<i>base_addr</i>	Base address of the next block.

Returns

The pointer of the block info struct that is no being used, or a NULL pointer in case of error.

Referenced by FlashAddToBlock().

7.417.2.13 FlashVerifyChecksum()

```
blt_bool FlashVerifyChecksum (  
    void )
```

Verifies the checksum, which indicates that a valid user program is present and can be started.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.417.2.14 FlashWrite()

```
blt_bool FlashWrite (  
    blt_addr addr,  
    blt_int32u len,  
    blt_int8u * data )
```

Writes the data to flash through a flash block manager. Note that this function also checks that no data is programmed outside the flash memory region, so the bootloader can never be overwritten.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.
<i>data</i>	Pointer to the data buffer.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

Referenced by FlashWriteChecksum().

7.417.2.15 FlashWriteBlock()

```
static blt_bool FlashWriteBlock (  
    tFlashBlockInfo * block ) [static]
```

Programs FLASH_WRITE_BLOCK_SIZE bytes to flash from the block->data array.

Parameters

<i>block</i>	Pointer to flash block info structure to operate on.
--------------	--

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

Referenced by FlashDone(), and FlashSwitchBlock().

7.417.2.16 FlashWriteChecksum()

```
blt_bool FlashWriteChecksum (  
    void )
```

Writes a checksum of the user program to non-volatile memory. This is performed once the entire user program has been programmed. Through the checksum, the bootloader can check if the programming session was completed, which indicates that a valid user programming is present and can be started.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.417.3 Variable Documentation**7.417.3.1 blockInfo**

```
tFlashBlockInfo blockInfo [static]
```

Local variable with information about the flash block that is currently being operated on.

The smallest amount of flash that can be programmed is FLASH_WRITE_BLOCK_SIZE. A flash block manager is implemented in this driver and stores info in this variable. Whenever new data should be flashed, it is first added to a RAM buffer, which is part of this variable. Whenever the RAM buffer, which has the size of a flash block, is full or data needs to be written to a different block, the contents of the RAM buffer are programmed to flash. The flash block manager requires some software overhead, yet results in faster flash programming because data is first harvested, ideally until there is enough to program an entire flash block, before the flash device is actually operated on.

Referenced by FlashSwitchBlock().

7.417.3.2 bootBlockInfo

```
tFlashBlockInfo bootBlockInfo [static]
```

Local variable with information about the flash boot block.

The first block of the user program holds the vector table, which on the STM32 is also the where the checksum is written to. Is it likely that the vector table is first flashed and then, at the end of the programming sequence, the checksum. This means that this flash block need to be written to twice. Normally this is not a problem with flash memory, as long as you write the same values to those bytes that are not supposed to be changed and the locations where you do write to are still in the erased 0xFF state. Unfortunately, writing twice to flash this way, does not work reliably on all micros. This is why we need to have an extra block, the bootblock, placed under the management of the block manager. This way is it possible to implement functionality so that the bootblock is only written to once at the end of the programming sequence.

Referenced by FlashSwitchBlock().

7.417.3.3 flashLayout

```
const tFlashSector flashLayout[] [static]
```

If desired, it is possible to set `BOOT_FLASH_CUSTOM_LAYOUT_ENABLE` to `> 0` in `blt_conf.h` and then implement your own version of the `flashLayout[]` table in a source-file with the name `flash_layout.c`. This way you customize the flash memory size reserved for the bootloader, without having to modify the `flashLayout[]` table in this file directly. This file will then include `flash_layout.c` so there is no need to compile it additionally with your project.

Array with the layout of the flash memory.

Also controls what part of the flash memory is reserved for the bootloader. If the bootloader size changes, the reserved sectors for the bootloader might need adjustment to make sure the bootloader doesn't get overwritten. The current flash layout does not reflect the minimum sector size of the physical flash (1 - 2kb), because this would make the table quite long and a waste of ROM. The minimum sector size is only really needed when erasing the flash. This can still be done in combination with macro `FLASH_ERASE_BLOCK_SIZE`.

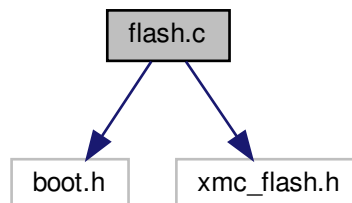
7.418 flash.c File Reference

Bootloader flash driver source file.

```
#include "boot.h"
```

```
#include "xmc_flash.h"
```

Include dependency graph for ARMCM4_XMC4/flash.c:



Data Structures

- struct [tFlashSector](#)
Flash sector descriptor type.
- struct [tFlashBlockInfo](#)
Structure type for grouping flash block information.

Macros

- #define [FLASH_INVALID_SECTOR](#) (0xff)
Value for an invalid flash sector.
- #define [FLASH_INVALID_ADDRESS](#) (0xffffffff)
Value for an invalid flash address.
- #define [FLASH_WRITE_BLOCK_SIZE](#) (1024)
Standard size of a flash block for writing. It should be large enough so that the OpenBLT checksum fits in the first (boot) block)
- #define [FLASH_TOTAL_SECTORS](#) (sizeof(flashLayout)/sizeof(flashLayout[0]))
Total numbers of sectors in array flashLayout[].
- #define [FLASH_END_ADDRESS](#)
End address of the bootloader programmable flash.
- #define [BOOT_FLASH_VECTOR_TABLE_CS_OFFSET](#) (0x200)
Offset into the user program's vector table where the checksum is located. For this target it is set to the end of the vector table. Note that the value can be overridden in blt_conf.h, because the size of the vector table could vary. When changing this value, don't forget to update the location of the checksum in the user program accordingly. Otherwise the checksum verification will always fail.
- #define [FLASH_WRITE_PAGE_SIZE](#) (256)
Minimum amount of bytes that can be programmed to flash at a time. It is hardware dependent.
- #define [FLASH_UNCACHED_BASE_ADDR](#) (0x0C000000U)
Base address in the memory map for uncached flash. It is hardware dependent.
- #define [FLASH_CACHED_BASE_ADDR](#) (0x08000000U)
Base address in the memory map for cached flash. It is hardware dependent.
- #define [FLASH_ERASE_TIME_MAX_MS](#) (6600)
Maximum time for a sector erase operation as specified by the XCM4xxx data- sheet with an added 20% margin.
- #define [FLASH_PROGRAM_TIME_MAX_MS](#) (13)
Maximum time for a page program operation as specified by the XCM4xxx data- sheet with an added 20% margin.

Functions

- static [blt_bool](#) [FlashInitBlock](#) ([tFlashBlockInfo](#) *block, [blt_addr](#) address)
Copies data currently in flash to the block->data and sets the base address.
- static [tFlashBlockInfo](#) * [FlashSwitchBlock](#) ([tFlashBlockInfo](#) *block, [blt_addr](#) base_addr)
Switches blocks by programming the current one and initializing the next.
- static [blt_bool](#) [FlashAddToBlock](#) ([tFlashBlockInfo](#) *block, [blt_addr](#) address, [blt_int8u](#) *data, [blt_int32u](#) len)
Programming is done per block. This function adds data to the block that is currently collecting data to be written to flash. If the address is outside of the current block, the current block is written to flash and a new block is initialized.
- static [blt_bool](#) [FlashWriteBlock](#) ([tFlashBlockInfo](#) *block)
Programs FLASH_WRITE_BLOCK_SIZE bytes to flash from the block->data array.
- static [blt_bool](#) [FlashEraseSectors](#) ([blt_int8u](#) first_sector, [blt_int8u](#) last_sector)
Erases the flash sectors from first_sector up until last_sector.
- static [blt_int8u](#) [FlashGetSector](#) ([blt_addr](#) address)

- Determines the flash sector the address is in.*

 - static `blt_addr FlashGetSectorBaseAddr (blt_int8u sector)`

Obtains the base address of the specified sector.
 - static `blt_addr FlashTranslateToNonCachedAddress (blt_addr address)`

The XMC4xxx has its PFLASH accessible in the memory map in two regions. One is the non-cached region starting at FLASH_UNCACHED_BASE_ADDR and the other is the cached region starting at FLASH_CACHED_BASE_ADDR. Flash erase and programming operations need to operate on addresses in the non-cached region. It is possible that the caller of this driver's API functions, specifies memory addresses in the cached region. This function automatically translates the memory address from cached to non-cached.
 - void `FlashInit (void)`

Initializes the flash driver.
 - void `FlashReinit (void)`

Reinitializes the flash driver.
 - `blt_bool FlashWrite (blt_addr addr, blt_int32u len, blt_int8u *data)`

Writes the data to flash through a flash block manager. Note that this function also checks that no data is programmed outside the flash memory region, so the bootloader can never be overwritten.
 - `blt_bool FlashErase (blt_addr addr, blt_int32u len)`

Erases the flash memory. Note that this function also checks that no data is erased outside the flash memory region, so the bootloader can never be erased.
 - `blt_bool FlashWriteChecksum (void)`

Writes a checksum of the user program to non-volatile memory. This is performed once the entire user program has been programmed. Through the checksum, the bootloader can check if the programming session was completed, which indicates that a valid user programming is present and can be started.
 - `blt_bool FlashVerifyChecksum (void)`

Verifies the checksum, which indicates that a valid user program is present and can be started.
 - `blt_bool FlashDone (void)`

Finalizes the flash driver operations. There could still be data in the currently active block that needs to be flashed.
 - `blt_addr FlashGetUserProgBaseAddress (void)`

Obtains the base address of the flash memory available to the user program. This is basically the first address in the flashLayout table.

Variables

- static const `tFlashSector flashLayout []`

If desired, it is possible to set BOOT_FLASH_CUSTOM_LAYOUT_ENABLE to > 0 in blt_conf.h and then implement your own version of the flashLayout[] table in a source-file with the name flash_layout.c. This way you customize the flash memory size reserved for the bootloader, without having to modify the flashLayout[] table in this file directly. This file will then include flash_layout.c so there is no need to compile it additionally with your project.
- static `tFlashBlockInfo blockInfo`

Local variable with information about the flash block that is currently being operated on.
- static `tFlashBlockInfo bootBlockInfo`

Local variable with information about the flash boot block.

7.418.1 Detailed Description

Bootloader flash driver source file.

7.418.2 Function Documentation

7.418.2.1 FlashAddToBlock()

```
static blt_bool FlashAddToBlock (
    tFlashBlockInfo * block,
    blt_addr address,
    blt_int8u * data,
    blt_int32u len ) [static]
```

Programming is done per block. This function adds data to the block that is currently collecting data to be written to flash. If the address is outside of the current block, the current block is written to flash and a new block is initialized.

Parameters

<i>block</i>	Pointer to flash block info structure to operate on.
<i>address</i>	Flash destination address.
<i>data</i>	Pointer to the byte array with data.
<i>len</i>	Number of bytes to add to the block.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

Referenced by FlashWrite().

7.418.2.2 FlashDone()

```
blt_bool FlashDone (
    void )
```

Finalizes the flash driver operations. There could still be data in the currently active block that needs to be flashed.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.418.2.3 FlashErase()

```
blt_bool FlashErase (
    blt_addr addr,
    blt_int32u len )
```

Erases the flash memory. Note that this function also checks that no data is erased outside the flash memory region, so the bootloader can never be erased.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.418.2.4 FlashEraseSectors()

```
static blt_bool FlashEraseSectors (  
    blt_int8u first_sector,  
    blt_int8u last_sector ) [static]
```

Erases the flash sectors from first_sector up until last_sector.

Parameters

<i>first_sector</i>	First flash sector number.
<i>last_sector</i>	Last flash sector number.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

Referenced by FlashErase().

7.418.2.5 FlashGetSector()

```
static blt_int8u FlashGetSector (  
    blt_addr address ) [static]
```

Determines the flash sector the address is in.

Parameters

<i>address</i>	Address in the flash sector.
----------------	------------------------------

Returns

Flash sector number or FLASH_INVALID_SECTOR.

Referenced by FlashErase(), FlashWrite(), and FlashWriteBlock().

7.418.2.6 FlashGetSectorBaseAddr()

```
static blt_addr FlashGetSectorBaseAddr (
    blt_int8u sector ) [static]
```

Obtains the base address of the specified sector.

Parameters

<i>sector</i>	Sector to get the base address of.
---------------	------------------------------------

Returns

Base Base address of the sector if found, FLASH_INVALID_ADDRESS otherwise.

Referenced by FlashEraseSectors().

7.418.2.7 FlashGetUserProgBaseAddress()

```
blt_addr FlashGetUserProgBaseAddress (
    void )
```

Obtains the base address of the flash memory available to the user program. This is basically the first address in the flashLayout table.

Returns

Base address.

7.418.2.8 FlashInit()

```
void FlashInit (
    void )
```

Initializes the flash driver.

Returns

none.

7.418.2.9 FlashInitBlock()

```
static blt_bool FlashInitBlock (
    tFlashBlockInfo * block,
    blt_addr address ) [static]
```

Copies data currently in flash to the block->data and sets the base address.

Parameters

<i>block</i>	Pointer to flash block info structure to operate on.
<i>address</i>	Base address of the block data.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

Referenced by FlashAddToBlock(), and FlashSwitchBlock().

7.418.2.10 FlashReinit()

```
void FlashReinit (
    void )
```

Reinitializes the flash driver.

Returns

none.

7.418.2.11 FlashSwitchBlock()

```
static tFlashBlockInfo * FlashSwitchBlock (
    tFlashBlockInfo * block,
    blt_addr base_addr ) [static]
```

Switches blocks by programming the current one and initializing the next.

Parameters

<i>block</i>	Pointer to flash block info structure to operate on.
<i>base_addr</i>	Base address of the next block.

Returns

The pointer of the block info struct that is no being used, or a NULL pointer in case of error.

Referenced by FlashAddToBlock().

7.418.2.12 FlashTranslateToNonCachedAddress()

```
static blt_addr FlashTranslateToNonCachedAddress (
    blt_addr address ) [static]
```

The XMC4xxx has its PFLASH accessible in the memory map in two regions. One is the non-cached region starting at FLASH_UNCACHED_BASE_ADDR and the other is the cached region starting at FLASH_CACHED_BASE_ADDR. Flash erase and programming operations need to operate on addresses in the non-cached region. It is possible that the caller of this driver's API functions, specifies memory addresses in the cached region. This function automatically translates the memory address from cached to non-cached.

Parameters

<i>address</i>	Address to translate.
----------------	-----------------------

Returns

Translated address.

Referenced by FlashErase(), and FlashWrite().

7.418.2.13 FlashVerifyChecksum()

```
blt_bool FlashVerifyChecksum (
    void )
```

Verifies the checksum, which indicates that a valid user program is present and can be started.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.418.2.14 FlashWrite()

```
blt_bool FlashWrite (
    blt_addr addr,
    blt_int32u len,
    blt_int8u * data )
```

Writes the data to flash through a flash block manager. Note that this function also checks that no data is programmed outside the flash memory region, so the bootloader can never be overwritten.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.
<i>data</i>	Pointer to the data buffer.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

Referenced by FlashWriteChecksum().

7.418.2.15 FlashWriteBlock()

```
static blt_bool FlashWriteBlock (  
    tFlashBlockInfo * block ) [static]
```

Programs FLASH_WRITE_BLOCK_SIZE bytes to flash from the block->data array.

Parameters

<i>block</i>	Pointer to flash block info structure to operate on.
--------------	--

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

Referenced by FlashDone(), and FlashSwitchBlock().

7.418.2.16 FlashWriteChecksum()

```
blt_bool FlashWriteChecksum (  
    void )
```

Writes a checksum of the user program to non-volatile memory. This is performed once the entire user program has been programmed. Through the checksum, the bootloader can check if the programming session was completed, which indicates that a valid user programming is present and can be started.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.418.3 Variable Documentation

7.418.3.1 blockInfo

```
tFlashBlockInfo blockInfo [static]
```

Local variable with information about the flash block that is currently being operated on.

The smallest amount of flash that can be programmed is FLASH_WRITE_BLOCK_SIZE. A flash block manager is implemented in this driver and stores info in this variable. Whenever new data should be flashed, it is first added to a RAM buffer, which is part of this variable. Whenever the RAM buffer, which has the size of a flash block, is full or data needs to be written to a different block, the contents of the RAM buffer are programmed to flash. The flash block manager requires some software overhead, yet results in faster flash programming because data is first harvested, ideally until there is enough to program an entire flash block, before the flash device is actually operated on.

Referenced by FlashSwitchBlock().

7.418.3.2 bootBlockInfo

```
tFlashBlockInfo bootBlockInfo [static]
```

Local variable with information about the flash boot block.

The first block of the user program holds the vector table, which on the STM32 is also the where the checksum is written to. Is it likely that the vector table is first flashed and then, at the end of the programming sequence, the checksum. This means that this flash block need to be written to twice. Normally this is not a problem with flash memory, as long as you write the same values to those bytes that are not supposed to be changed and the locations where you do write to are still in the erased 0xFF state. Unfortunately, writing twice to flash this way, does not work reliably on all micros. This is why we need to have an extra block, the bootblock, placed under the management of the block manager. This way is it possible to implement functionality so that the bootblock is only written to once at the end of the programming sequence.

Referenced by FlashSwitchBlock().

7.418.3.3 flashLayout

```
const tFlashSector flashLayout[] [static]
```

Initial value:

```
=
{
    { 0x0c00c000, 0x04000, 3},
    { 0x0c010000, 0x04000, 4},
    { 0x0c014000, 0x04000, 5},
    { 0x0c018000, 0x04000, 6},
    { 0x0c01c000, 0x04000, 7},
    { 0x0c020000, 0x20000, 8},

    { 0x0c040000, 0x40000, 9},

    { 0x0c080000, 0x40000, 10},
```



```
{ 0x0c0C0000, 0x40000, 11},  
  
}
```

If desired, it is possible to set `BOOT_FLASH_CUSTOM_LAYOUT_ENABLE` to `> 0` in `blt_conf.h` and then implement your own version of the `flashLayout[]` table in a source-file with the name `flash_layout.c`. This way you customize the flash memory size reserved for the bootloader, without having to modify the `flashLayout[]` table in this file directly. This file will then include `flash_layout.c` so there is no need to compile it additionally with your project.

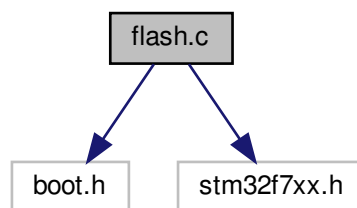
Array with the layout of the flash memory.

Also controls what part of the flash memory is reserved for the bootloader. If the bootloader size changes, the reserved sectors for the bootloader might need adjustment to make sure the bootloader doesn't get overwritten. Note that the table contains uncached addresses, because flash program/ erase operations need to be performed on uncached addresses. This flash driver automatically translated cached to uncached addresses, so there is no need for the user to adjust this when calling this driver's API.

7.419 flash.c File Reference

Bootloader flash driver source file.

```
#include "boot.h"  
#include "stm32f7xx.h"  
Include dependency graph for ARMC7_STM32F7/flash.c:
```



Data Structures

- struct [tFlashSector](#)
Flash sector descriptor type.
- struct [tFlashBlockInfo](#)
Structure type for grouping flash block information.

Macros

- `#define FLASH_INVALID_SECTOR (0xff)`
Value for an invalid flash sector.
- `#define FLASH_INVALID_ADDRESS (0xffffffff)`
Value for an invalid flash address.
- `#define FLASH_WRITE_BLOCK_SIZE (512)`
Standard size of a flash block for writing.
- `#define FLASH_TOTAL_SECTORS (sizeof(flashLayout)/sizeof(flashLayout[0]))`
Total numbers of sectors in array flashLayout[].
- `#define FLASH_END_ADDRESS`
End address of the bootloader programmable flash.
- `#define BOOT_FLASH_VECTOR_TABLE_CS_OFFSET (0x1C8)`
Offset into the user program's vector table where the checksum is located. For this target it is set to the end of the vector table. Note that the value can be overridden in blt_conf.h, because the size of the vector table could vary. When changing this value, don't forget to update the location of the checksum in the user program accordingly. Otherwise the checksum verification will always fail.

Functions

- `static blt_bool FlashInitBlock (tFlashBlockInfo *block, blt_addr address)`
Copies data currently in flash to the block->data and sets the base address.
- `static tFlashBlockInfo * FlashSwitchBlock (tFlashBlockInfo *block, blt_addr base_addr)`
Switches blocks by programming the current one and initializing the next.
- `static blt_bool FlashAddToBlock (tFlashBlockInfo *block, blt_addr address, blt_int8u *data, blt_int32u len)`
Programming is done per block. This function adds data to the block that is currently collecting data to be written to flash. If the address is outside of the current block, the current block is written to flash and a new block is initialized.
- `static blt_bool FlashWriteBlock (tFlashBlockInfo *block)`
Programs FLASH_WRITE_BLOCK_SIZE bytes to flash from the block->data array.
- `static blt_bool FlashEraseSectors (blt_int8u first_sector, blt_int8u last_sector)`
Erases the flash sectors from first_sector up until last_sector.
- `static blt_int8u FlashGetSector (blt_addr address)`
Determines the flash sector the address is in.
- `static blt_bool FlashIsSingleBankMode (void)`
Determines the flash is configured in single bank mode, which is required by this flash driver.
- `void FlashInit (void)`
Initializes the flash driver.
- `void FlashReinit (void)`
Reinitializes the flash driver.
- `blt_bool FlashWrite (blt_addr addr, blt_int32u len, blt_int8u *data)`
Writes the data to flash through a flash block manager. Note that this function also checks that no data is programmed outside the flash memory region, so the bootloader can never be overwritten.
- `blt_bool FlashErase (blt_addr addr, blt_int32u len)`
Erases the flash memory. Note that this function also checks that no data is erased outside the flash memory region, so the bootloader can never be erased.
- `blt_bool FlashWriteChecksum (void)`
Writes a checksum of the user program to non-volatile memory. This is performed once the entire user program has been programmed. Through the checksum, the bootloader can check if the programming session was completed, which indicates that a valid user programming is present and can be started.
- `blt_bool FlashVerifyChecksum (void)`
Verifies the checksum, which indicates that a valid user program is present and can be started.
- `blt_bool FlashDone (void)`
Finalizes the flash driver operations. There could still be data in the currently active block that needs to be flashed.
- `blt_addr FlashGetUserProgBaseAddress (void)`
Obtains the base address of the flash memory available to the user program. This is basically the first address in the flashLayout table.

Variables

- static const [tFlashSector flashLayout \[\]](#)

If desired, it is possible to set `BOOT_FLASH_CUSTOM_LAYOUT_ENABLE` to `> 0` in `blt_conf.h` and then implement your own version of the `flashLayout[]` table in a source-file with the name `flash_layout.c`. This way you customize the flash memory size reserved for the bootloader, without having to modify the `flashLayout[]` table in this file directly. This file will then include `flash_layout.c` so there is no need to compile it additionally with your project.

- static [tFlashBlockInfo blockInfo](#)

Local variable with information about the flash block that is currently being operated on.

- static [tFlashBlockInfo bootBlockInfo](#)

Local variable with information about the flash boot block.

7.419.1 Detailed Description

Bootloader flash driver source file.

7.419.2 Function Documentation

7.419.2.1 FlashAddToBlock()

```
static blt\_bool FlashAddToBlock (  
    tFlashBlockInfo * block,  
    blt\_addr address,  
    blt\_int8u * data,  
    blt\_int32u len ) [static]
```

Programming is done per block. This function adds data to the block that is currently collecting data to be written to flash. If the address is outside of the current block, the current block is written to flash and a new block is initialized.

Parameters

<i>block</i>	Pointer to flash block info structure to operate on.
<i>address</i>	Flash destination address.
<i>data</i>	Pointer to the byte array with data.
<i>len</i>	Number of bytes to add to the block.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

Referenced by `FlashWrite()`.

7.419.2.2 FlashDone()

```
blt_bool FlashDone (
    void )
```

Finalizes the flash driver operations. There could still be data in the currently active block that needs to be flashed.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.419.2.3 FlashErase()

```
blt_bool FlashErase (
    blt_addr addr,
    blt_int32u len )
```

Erases the flash memory. Note that this function also checks that no data is erased outside the flash memory region, so the bootloader can never be erased.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.419.2.4 FlashEraseSectors()

```
static blt_bool FlashEraseSectors (
    blt_int8u first_sector,
    blt_int8u last_sector ) [static]
```

Erases the flash sectors from *first_sector* up until *last_sector*.

Parameters

<i>first_sector</i>	First flash sector number.
<i>last_sector</i>	Last flash sector number.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

Referenced by FlashErase().

7.419.2.5 FlashGetSector()

```
static blt_int8u FlashGetSector (
    blt_addr address ) [static]
```

Determines the flash sector the address is in.

Parameters

<i>address</i>	Address in the flash sector.
----------------	------------------------------

Returns

Flash sector number or FLASH_INVALID_SECTOR.

Referenced by FlashErase(), and FlashWrite().

7.419.2.6 FlashGetUserProgBaseAddress()

```
blt_addr FlashGetUserProgBaseAddress (
    void )
```

Obtains the base address of the flash memory available to the user program. This is basically the first address in the flashLayout table.

Returns

Base address.

7.419.2.7 FlashInit()

```
void FlashInit (
    void )
```

Initializes the flash driver.

Returns

none.

7.419.2.8 FlashInitBlock()

```
static blt_bool FlashInitBlock (
    tFlashBlockInfo * block,
    blt_addr address ) [static]
```

Copies data currently in flash to the block->data and sets the base address.

Parameters

<i>block</i>	Pointer to flash block info structure to operate on.
<i>address</i>	Base address of the block data.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

Referenced by FlashAddToBlock(), and FlashSwitchBlock().

7.419.2.9 FlashIsSingleBankMode()

```
static blt_bool FlashIsSingleBankMode (  
    void ) [static]
```

Determines the flash is configured in single bank mode, which is required by this flash driver.

Returns

BLT_TRUE if the flash is in single bank mode, BLT_FALSE otherwise.

Referenced by FlashEraseSectors(), and FlashWriteBlock().

7.419.2.10 FlashReinit()

```
void FlashReinit (  
    void )
```

Reinitializes the flash driver.

Returns

none.

7.419.2.11 FlashSwitchBlock()

```
static tFlashBlockInfo * FlashSwitchBlock (  
    tFlashBlockInfo * block,  
    blt_addr base_addr ) [static]
```

Switches blocks by programming the current one and initializing the next.

Parameters

<i>block</i>	Pointer to flash block info structure to operate on.
<i>base_addr</i>	Base address of the next block.

Returns

The pointer of the block info struct that is no being used, or a NULL pointer in case of error.

Referenced by FlashAddToBlock().

7.419.2.12 FlashVerifyChecksum()

```
blt_bool FlashVerifyChecksum (
    void )
```

Verifies the checksum, which indicates that a valid user program is present and can be started.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.419.2.13 FlashWrite()

```
blt_bool FlashWrite (
    blt_addr addr,
    blt_int32u len,
    blt_int8u * data )
```

Writes the data to flash through a flash block manager. Note that this function also checks that no data is programmed outside the flash memory region, so the bootloader can never be overwritten.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.
<i>data</i>	Pointer to the data buffer.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

Referenced by FlashWriteChecksum().

7.419.2.14 FlashWriteBlock()

```
static blt_bool FlashWriteBlock (
    tFlashBlockInfo * block ) [static]
```

Programs FLASH_WRITE_BLOCK_SIZE bytes to flash from the block->data array.

Parameters

<i>block</i>	Pointer to flash block info structure to operate on.
--------------	--

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

Referenced by FlashDone(), and FlashSwitchBlock().

7.419.2.15 FlashWriteChecksum()

```
blt_bool FlashWriteChecksum (
    void )
```

Writes a checksum of the user program to non-volatile memory. This is performed once the entire user program has been programmed. Through the checksum, the bootloader can check if the programming session was completed, which indicates that a valid user programming is present and can be started.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.419.3 Variable Documentation

7.419.3.1 blockInfo

```
tFlashBlockInfo blockInfo [static]
```

Local variable with information about the flash block that is currently being operated on.

The smallest amount of flash that can be programmed is FLASH_WRITE_BLOCK_SIZE. A flash block manager is implemented in this driver and stores info in this variable. Whenever new data should be flashed, it is first added to a RAM buffer, which is part of this variable. Whenever the RAM buffer, which has the size of a flash block, is full or data needs to be written to a different block, the contents of the RAM buffer are programmed to flash. The flash block manager requires some software overhead, yet results in faster flash programming because data is first harvested, ideally until there is enough to program an entire flash block, before the flash device is actually operated on.

Referenced by FlashSwitchBlock().

7.419.3.2 bootBlockInfo

```
tFlashBlockInfo bootBlockInfo [static]
```

Local variable with information about the flash boot block.

The first block of the user program holds the vector table, which on the STM32 is also the where the checksum is written to. Is it likely that the vector table is first flashed and then, at the end of the programming sequence, the checksum. This means that this flash block need to be written to twice. Normally this is not a problem with flash memory, as long as you write the same values to those bytes that are not supposed to be changed and the locations where you do write to are still in the erased 0xFF state. Unfortunately, writing twice to flash this way, does not work reliably on all micros. This is why we need to have an extra block, the bootblock, placed under the management of the block manager. This way is it possible to implement functionality so that the bootblock is only written to once at the end of the programming sequence.

Referenced by FlashSwitchBlock().

7.419.3.3 flashLayout

```
const tFlashSector flashLayout[] [static]
```

Initial value:

```
=
{
    { 0x08008000, 0x08000, 1},
    { 0x08010000, 0x08000, 2},
    { 0x08018000, 0x08000, 3},

    { 0x08020000, 0x20000, 4},

    { 0x08040000, 0x40000, 5},

    { 0x08080000, 0x40000, 6},
    { 0x080C0000, 0x40000, 7},

}
```

If desired, it is possible to set `BOOT_FLASH_CUSTOM_LAYOUT_ENABLE` to `> 0` in `blt_conf.h` and then implement your own version of the `flashLayout[]` table in a source-file with the name `flash_layout.c`. This way you customize the flash memory size reserved for the bootloader, without having to modify the `flashLayout[]` table in this file directly. This file will then include `flash_layout.c` so there is no need to compile it additionally with your project.

Array wit the layout of the flash memory.

Also controls what part of the flash memory is reserved for the bootloader. If the bootloader size changes, the reserved sectors for the bootloader might need adjustment to make sure the bootloader doesn't get overwritten.

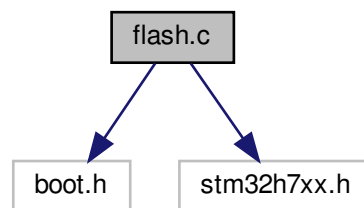
7.420 flash.c File Reference

Bootloader flash driver source file.

```
#include "boot.h"
```

```
#include "stm32h7xx.h"
```

Include dependency graph for ARMCM7_STM32H7/flash.c:



Data Structures

- struct [tFlashSector](#)
Flash sector descriptor type.
- struct [tFlashBlockInfo](#)
Structure type for grouping flash block information.

Macros

- #define [FLASH_INVALID_SECTOR_IDX](#) (0xff)
Value for an invalid sector entry index into flashLayout[].
- #define [FLASH_INVALID_ADDRESS](#) (0xffffffff)
Value for an invalid flash address.
- #define [FLASH_WRITE_BLOCK_SIZE](#) (1024)
Standard size of a flash block for writing.
- #define [FLASH_TOTAL_SECTORS](#) (sizeof(flashLayout)/sizeof(flashLayout[0]))
Total numbers of sectors in array flashLayout[].
- #define [FLASH_END_ADDRESS](#)
End address of the bootloader programmable flash.
- #define [BOOT_FLASH_VECTOR_TABLE_CS_OFFSET](#) (0x298)
Offset into the user program's vector table where the checksum is located. For this target it is set to the end of the vector table. Note that the value can be overridden in blt_conf.h, because the size of the vector table could vary. When changing this value, don't forget to update the location of the checksum in the user program accordingly. Otherwise the checksum verification will always fail.

Functions

- static `blt_bool` `FlashInitBlock` (`tFlashBlockInfo` *block, `blt_addr` address)
Copies data currently in flash to the block->data and sets the base address.
- static `tFlashBlockInfo` * `FlashSwitchBlock` (`tFlashBlockInfo` *block, `blt_addr` base_addr)
Switches blocks by programming the current one and initializing the next.
- static `blt_bool` `FlashAddToBlock` (`tFlashBlockInfo` *block, `blt_addr` address, `blt_int8u` *data, `blt_int32u` len)
Programming is done per block. This function adds data to the block that is currently collecting data to be written to flash. If the address is outside of the current block, the current block is written to flash and a new block is initialized.
- static `blt_bool` `FlashWriteBlock` (`tFlashBlockInfo` *block)
Programs FLASH_WRITE_BLOCK_SIZE bytes to flash from the block->data array.
- static `blt_bool` `FlashEraseSectors` (`blt_int8u` first_sector_idx, `blt_int8u` last_sector_idx)
Erases the flash sectors from indices first_sector_idx up until last_sector_idx into the flashLayout[] array.
- static `blt_int8u` `FlashGetSectorIdx` (`blt_addr` address)
Determines the index into the flashLayout[] array of the flash sector that the specified address is in.
- void `FlashInit` (void)
Initializes the flash driver.
- void `FlashReinit` (void)
Reinitializes the flash driver.
- `blt_bool` `FlashWrite` (`blt_addr` addr, `blt_int32u` len, `blt_int8u` *data)
Writes the data to flash through a flash block manager. Note that this function also checks that no data is programmed outside the flash memory region, so the bootloader can never be overwritten.
- `blt_bool` `FlashErase` (`blt_addr` addr, `blt_int32u` len)
Erases the flash memory. Note that this function also checks that no data is erased outside the flash memory region, so the bootloader can never be erased.
- `blt_bool` `FlashWriteChecksum` (void)
Writes a checksum of the user program to non-volatile memory. This is performed once the entire user program has been programmed. Through the checksum, the bootloader can check if the programming session was completed, which indicates that a valid user programming is present and can be started.
- `blt_bool` `FlashVerifyChecksum` (void)
Verifies the checksum, which indicates that a valid user program is present and can be started.
- `blt_bool` `FlashDone` (void)
Finalizes the flash driver operations. There could still be data in the currently active block that needs to be flashed.
- `blt_addr` `FlashGetUserProgBaseAddress` (void)
Obtains the base address of the flash memory available to the user program. This is basically the first address in the flashLayout table.

Variables

- static const `tFlashSector` `flashLayout` []
If desired, it is possible to set BOOT_FLASH_CUSTOM_LAYOUT_ENABLE to > 0 in blt_conf.h and then implement your own version of the flashLayout[] table in a source-file with the name flash_layout.c. This way you customize the flash memory size reserved for the bootloader, without having to modify the flashLayout[] table in this file directly. This file will then include flash_layout.c so there is no need to compile it additionally with your project.
- static `tFlashBlockInfo` `blockInfo`
Local variable with information about the flash block that is currently being operated on.
- static `tFlashBlockInfo` `bootBlockInfo`
Local variable with information about the flash boot block.

7.420.1 Detailed Description

Bootloader flash driver source file.

7.420.2 Function Documentation

7.420.2.1 FlashAddToBlock()

```
static blt_bool FlashAddToBlock (
    tFlashBlockInfo * block,
    blt_addr address,
    blt_int8u * data,
    blt_int32u len ) [static]
```

Programming is done per block. This function adds data to the block that is currently collecting data to be written to flash. If the address is outside of the current block, the current block is written to flash and a new block is initialized.

Parameters

<i>block</i>	Pointer to flash block info structure to operate on.
<i>address</i>	Flash destination address.
<i>data</i>	Pointer to the byte array with data.
<i>len</i>	Number of bytes to add to the block.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

Referenced by FlashWrite().

7.420.2.2 FlashDone()

```
blt_bool FlashDone (
    void )
```

Finalizes the flash driver operations. There could still be data in the currently active block that needs to be flashed.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.420.2.3 FlashErase()

```
blt_bool FlashErase (
    blt_addr addr,
    blt_int32u len )
```

Erases the flash memory. Note that this function also checks that no data is erased outside the flash memory region, so the bootloader can never be erased.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.420.2.4 FlashEraseSectors()

```
static blt_bool FlashEraseSectors (  
    blt_int8u first_sector_idx,  
    blt_int8u last_sector_idx ) [static]
```

Erases the flash sectors from indices first_sector_idx up until last_sector_idx into the flashLayout[] array.

Parameters

<i>first_sector_idx</i>	First flash sector number index into flashLayout[].
<i>last_sector_idx</i>	Last flash sector number index into flashLayout[].

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

Referenced by FlashErase().

7.420.2.5 FlashGetSectorIdx()

```
static blt_int8u FlashGetSectorIdx (  
    blt_addr address ) [static]
```

Determines the index into the flashLayout[] array of the flash sector that the specified address is in.

Parameters

<i>address</i>	Address in the flash sector.
----------------	------------------------------

Returns

Flash sector index in flashLayout[] or FLASH_INVALID_SECTOR_IDX.

Referenced by FlashErase(), FlashWrite(), and FlashWriteBlock().

7.420.2.6 FlashGetUserProgBaseAddress()

```
blt_addr FlashGetUserProgBaseAddress (
    void )
```

Obtains the base address of the flash memory available to the user program. This is basically the first address in the flashLayout table.

Returns

Base address.

7.420.2.7 FlashInit()

```
void FlashInit (
    void )
```

Initializes the flash driver.

Returns

none.

7.420.2.8 FlashInitBlock()

```
static blt_bool FlashInitBlock (
    tFlashBlockInfo * block,
    blt_addr address ) [static]
```

Copies data currently in flash to the block->data and sets the base address.

Parameters

<i>block</i>	Pointer to flash block info structure to operate on.
<i>address</i>	Base address of the block data.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

Referenced by FlashAddToBlock(), and FlashSwitchBlock().

7.420.2.9 FlashReinit()

```
void FlashReinit (
    void )
```

Reinitializes the flash driver.

Returns

none.

7.420.2.10 FlashSwitchBlock()

```
static tFlashBlockInfo * FlashSwitchBlock (
    tFlashBlockInfo * block,
    blt_addr base_addr ) [static]
```

Switches blocks by programming the current one and initializing the next.

Parameters

<i>block</i>	Pointer to flash block info structure to operate on.
<i>base_addr</i>	Base address of the next block.

Returns

The pointer of the block info struct that is now being used, or a NULL pointer in case of error.

Referenced by FlashAddToBlock().

7.420.2.11 FlashVerifyChecksum()

```
blt_bool FlashVerifyChecksum (
    void )
```

Verifies the checksum, which indicates that a valid user program is present and can be started.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.420.2.12 FlashWrite()

```
blt_bool FlashWrite (
    blt_addr addr,
    blt_int32u len,
    blt_int8u * data )
```

Writes the data to flash through a flash block manager. Note that this function also checks that no data is programmed outside the flash memory region, so the bootloader can never be overwritten.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.
<i>data</i>	Pointer to the data buffer.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

Referenced by FlashWriteChecksum().

7.420.2.13 FlashWriteBlock()

```
static blt_bool FlashWriteBlock (  
    tFlashBlockInfo * block ) [static]
```

Programs FLASH_WRITE_BLOCK_SIZE bytes to flash from the block->data array.

Parameters

<i>block</i>	Pointer to flash block info structure to operate on.
--------------	--

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

Referenced by FlashDone(), and FlashSwitchBlock().

7.420.2.14 FlashWriteChecksum()

```
blt_bool FlashWriteChecksum (  
    void )
```

Writes a checksum of the user program to non-volatile memory. This is performed once the entire user program has been programmed. Through the checksum, the bootloader can check if the programming session was completed, which indicates that a valid user programming is present and can be started.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.420.3 Variable Documentation

7.420.3.1 blockInfo

```
tFlashBlockInfo blockInfo [static]
```

Local variable with information about the flash block that is currently being operated on.

The smallest amount of flash that can be programmed is FLASH_WRITE_BLOCK_SIZE. A flash block manager is implemented in this driver and stores info in this variable. Whenever new data should be flashed, it is first added to a RAM buffer, which is part of this variable. Whenever the RAM buffer, which has the size of a flash block, is full or data needs to be written to a different block, the contents of the RAM buffer are programmed to flash. The flash block manager requires some software overhead, yet results in faster flash programming because data is first harvested, ideally until there is enough to program an entire flash block, before the flash device is actually operated on.

Referenced by FlashSwitchBlock().

7.420.3.2 bootBlockInfo

```
tFlashBlockInfo bootBlockInfo [static]
```

Local variable with information about the flash boot block.

The first block of the user program holds the vector table, which on the STM32 is also the where the checksum is written to. Is it likely that the vector table is first flashed and then, at the end of the programming sequence, the checksum. This means that this flash block need to be written to twice. Normally this is not a problem with flash memory, as long as you write the same values to those bytes that are not supposed to be changed and the locations where you do write to are still in the erased 0xFF state. Unfortunately, writing twice to flash this way, does not work reliably on all micros. This is why we need to have an extra block, the bootblock, placed under the management of the block manager. This way is it possible to implement functionality so that the bootblock is only written to once at the end of the programming sequence.

Referenced by FlashSwitchBlock().

7.420.3.3 flashLayout

```
const tFlashSector flashLayout[] [static]
```

If desired, it is possible to set BOOT_FLASH_CUSTOM_LAYOUT_ENABLE to > 0 in blt_conf.h and then implement your own version of the flashLayout[] table in a source-file with the name flash_layout.c. This way you customize the flash memory size reserved for the bootloader, without having to modify the flashLayout[] table in this file directly. This file will then include flash_layout.c so there is no need to compile it additionally with your project.

Array with the layout of the flash memory.

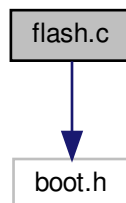
Also controls what part of the flash memory is reserved for the bootloader. If the bootloader size changes, the reserved sectors for the bootloader might need adjustment to make sure the bootloader doesn't get overwritten.

7.421 flash.c File Reference

Bootloader flash driver source file.

```
#include "boot.h"
```

Include dependency graph for HCS12/flash.c:



Data Structures

- struct [tFlashSector](#)
Flash sector descriptor type.
- struct [tFlashBlockInfo](#)
Structure type for grouping flash block information.
- struct [tFlashRegs](#)
Structure type for the flash control registers.

Macros

- #define [FLASH_INVALID_SECTOR_IDX](#) (0xff)
Value for an invalid flash sector.
- #define [FLASH_INVALID_ADDRESS](#) (0xffffffff)
Value for an invalid flash address.
- #define [FLASH_WRITE_BLOCK_SIZE](#) (512)
Standard size of a flash block for writing.
- #define [FLASH_TOTAL_SECTORS](#) (sizeof(flashLayout)/sizeof(flashLayout[0]))
Total numbers of sectors in array flashLayout[].
- #define [BOOT_FLASH_VECTOR_TABLE_CS_OFFSET](#) (0x82)
Offset into the user program's vector table where the checksum is located. Note that the value can be overridden in blt_conf.h, because the size of the vector table could vary. When changing this value, don't forget to update the location of the checksum in the user program accordingly. Otherwise the checksum verification will always fail.
- #define [FLASH_VECTOR_TABLE_SIZE](#) (0x80)
Total size of the vector table, excluding the bootloader specific checksum.
- #define [FLASH_START_ADDRESS](#) (flashLayout[0].sector_start)
Start address of the bootloader programmable flash.
- #define [FLASH_END_ADDRESS](#)
End address of the bootloader programmable flash.
- #define [FLASH_PAGE_SIZE](#) (0x4000) /* flash page size in bytes */

- Size of a flash page on the HCS12.*

 - #define [FLASH_PAGE_OFFSET](#) (0x8000) /* physical start addr. of pages */

Physical start address of the HCS12 page window.
- #define [FLASH_PPAGE_REG](#) (*(volatile [blt_int8u](#) *) (0x0030))

PPAGE register to select a specific flash page.
- #define [FLASH_REGS_BASE_ADDRESS](#) (0x0100)

Base address of the flash related control registers.
- #define [FLASH](#) ((volatile [tFlashRegs](#) *) [FLASH_REGS_BASE_ADDRESS](#))

Macro for accessing the flash related control registers.
- #define [FLASH_PROGRAM_WORD_CMD](#) (0x20)

Program word flash command.
- #define [FLASH_ERASE_SECTOR_CMD](#) (0x40)

Erase sector flash command.
- #define [FLASH_PAGES_PER_BLOCK](#) (8)

Number of flash pages in a block.
- #define [FLASH_BLOCK_SEL_MASK](#) (0x03)

Bitmask for selecting a block with flash pages.
- #define [PRDIV8_BIT](#) (0x40)

FCLKDIV - enable prescaler by 8 bit.
- #define [ACCERR_BIT](#) (0x10)

FSTAT - flash access error bit.
- #define [PVIOL_BIT](#) (0x20)

FSTAT - protection violation bit.
- #define [CBEIF_BIT](#) (0x80)

FSTAT - command buffer empty flag bit.
- #define [CBEIE_BIT](#) (0x80)

FCNFG - command buf. empty irq enable bit.
- #define [CCIE_BIT](#) (0x40)

FCNFG - command complete irq enable bit.
- #define [KEYACC_BIT](#) (0x20)

FCNFG - enable security key writing bit.

Typedefs

- typedef void(* [pFlashExeCmdFct](#)) (void)

Pointer type to flash command execution function.

Functions

- static [blt_bool](#) [FlashInitBlock](#) ([tFlashBlockInfo](#) *block, [blt_addr](#) address)

Copies data currently in flash to the block->data and sets the base address.
- static [tFlashBlockInfo](#) * [FlashSwitchBlock](#) ([tFlashBlockInfo](#) *block, [blt_addr](#) base_addr)

Switches blocks by programming the current one and initializing the next.
- static [blt_bool](#) [FlashAddToBlock](#) ([tFlashBlockInfo](#) *block, [blt_addr](#) address, [blt_int8u](#) *data, [blt_int32u](#) len)

Programming is done per block. This function adds data to the block that is currently collecting data to be written to flash. If the address is outside of the current block, the current block is written to flash and a new block is initialized.
- static [blt_bool](#) [FlashWriteBlock](#) ([tFlashBlockInfo](#) *block)

Programs FLASH_WRITE_BLOCK_SIZE bytes to flash from the block->data array.
- static [blt_int8u](#) [FlashGetLinearAddrByte](#) ([blt_addr](#) addr)

- Reads the byte value from the linear address.*

 - static `blt_int8u FlashGetPhysPage (blt_addr addr)`

Extracts the physical flash page number from a linear address.
 - static `blt_int16u FlashGetPhysAddr (blt_addr addr)`

Extracts the physical address on the flash page number from a linear address.
 - static void `FlashExecuteCommand (void)`

Executes the command. The actual code for the command execution is stored as location independant machine code in array `flashExecCmd[]`. The contents of this array are temporarily copied to RAM. This way the function can be executed from RAM avoiding problem when try to perform a flash operation on the same flash block that this driver is located.
 - static `blt_bool FlashOperate (blt_int8u cmd, blt_addr addr, blt_int16u data)`

Prepares the flash command and executes it.
 - void `FlashInit (void)`

Initializes the flash driver.
 - void `FlashReinit (void)`

Reinitializes the flash driver.
 - `blt_bool FlashWrite (blt_addr addr, blt_int32u len, blt_int8u *data)`

Writes the data to flash through a flash block manager. Note that this function also checks that no data is programmed outside the flash memory region, so the bootloader can never be overwritten.
 - `blt_bool FlashErase (blt_addr addr, blt_int32u len)`

Erases the flash memory. Note that this function also checks that no data is erased outside the flash memory region, so the bootloader can never be erased.
 - `blt_bool FlashWriteChecksum (void)`

Writes a checksum of the user program to non-volatile memory. This is performed once the entire user program has been programmed. Through the checksum, the bootloader can check if the programming session was completed, which indicates that a valid user programming is present and can be started.
 - `blt_bool FlashVerifyChecksum (void)`

Verifies the checksum, which indicates that a valid user program is present and can be started.
 - `blt_bool FlashDone (void)`

Finalizes the flash driver operations. There could still be data in the currently active block that needs to be flashed.
 - `blt_addr FlashGetUserProgBaseAddress (void)`

Obtains the base address of the flash memory available to the user program. This is basically the last address in the `flashLayout` table converted to the physical address on the last page (0x3f), because this is where the address will be in.

Variables

- static const `tFlashSector flashLayout []`

If desired, it is possible to set `BOOT_FLASH_CUSTOM_LAYOUT_ENABLE` to `> 0` in `blt_conf.h` and then implement your own version of the `flashLayout[]` table in a source-file with the name `flash_layout.c`. This way you customize the flash memory size reserved for the bootloader, without having to modify the `flashLayout[]` table in this file directly. This file will then include `flash_layout.c` so there is no need to compile it additionally with your project.
- static const `blt_int8u flashExecCmd []`

Array with executable code for performing flash operations.
- static `tFlashBlockInfo blockInfo`

Local variable with information about the flash block that is currently being operated on.
- static `tFlashBlockInfo bootBlockInfo`

Local variable with information about the flash boot block.
- static `blt_int8u flashExecCmdRam [(sizeof(flashExecCmd)/sizeof(flashExecCmd[0]))]`

RAM buffer where the executable flash operation code is copied to.
- static `blt_int8u flashMaxNrBlocks`

Maximum number of supported blocks, which is determined dynamically to have code that is independent of the used HCS12 derivative.

7.421.1 Detailed Description

Bootloader flash driver source file.

7.421.2 Function Documentation

7.421.2.1 FlashAddToBlock()

```
static blt_bool FlashAddToBlock (
    tFlashBlockInfo * block,
    blt_addr address,
    blt_int8u * data,
    blt_int32u len ) [static]
```

Programming is done per block. This function adds data to the block that is currently collecting data to be written to flash. If the address is outside of the current block, the current block is written to flash and a new block is initialized.

Parameters

<i>block</i>	Pointer to flash block info structure to operate on.
<i>address</i>	Flash destination address.
<i>data</i>	Pointer to the byte array with data.
<i>len</i>	Number of bytes to add to the block.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.421.2.2 FlashDone()

```
blt_bool FlashDone (
    void )
```

Finalizes the flash driver operations. There could still be data in the currently active block that needs to be flashed.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.421.2.3 FlashErase()

```
blt_bool FlashErase (
    blt_addr addr,
    blt_int32u len )
```

Erases the flash memory. Note that this function also checks that no data is erased outside the flash memory region, so the bootloader can never be erased.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.421.2.4 FlashExecuteCommand()

```
static void FlashExecuteCommand (  
    void ) [static]
```

Executes the command. The actual code for the command execution is stored as location independant machine code in array flashExecCmd[]. The contents of this array are temporarily copied to RAM. This way the function can be executed from RAM avoiding problem when try to perform a flash operation on the same flash block that this driver is located.

Returns

none.

Referenced by FlashOperate().

7.421.2.5 FlashGetLinearAddrByte()

```
static blt_int8u FlashGetLinearAddrByte (  
    blt_addr addr ) [static]
```

Reads the byte value from the linear address.

Parameters

<i>addr</i>	Linear address.
-------------	-----------------

Returns

The byte value located at the linear address.

Referenced by FlashWriteBlock().

7.421.2.6 FlashGetPhysAddr()

```
static blt_int16u FlashGetPhysAddr (
    blt_addr addr ) [static]
```

Extracts the physical address on the flash page number from a linear address.

Parameters

<i>addr</i>	Linear address.
-------------	-----------------

Returns

The physical address.

Referenced by FlashGetLinearAddrByte(), FlashGetUserProgBaseAddress(), FlashInitBlock(), and FlashOperate().

7.421.2.7 FlashGetPhysPage()

```
static blt_int8u FlashGetPhysPage (
    blt_addr addr ) [static]
```

Extracts the physical flash page number from a linear address.

Parameters

<i>addr</i>	Linear address.
-------------	-----------------

Returns

The page number.

Referenced by FlashGetLinearAddrByte(), FlashInitBlock(), and FlashOperate().

7.421.2.8 FlashGetUserProgBaseAddress()

```
blt_addr FlashGetUserProgBaseAddress (
    void )
```

Obtains the base address of the flash memory available to the user program. This is basically the last address in the flashLayout table converted to the physical address on the last page (0x3f), because this is where the address will be in.

Obtains the base address of the flash memory available to the user program. This is basically the first address in the flashLayout table.

Returns

Base address.

7.421.2.9 FlashInit()

```
void FlashInit (
    void )
```

Initializes the flash driver.

Returns

none.

7.421.2.10 FlashInitBlock()

```
static blt_bool FlashInitBlock (
    tFlashBlockInfo * block,
    blt_addr address ) [static]
```

Copies data currently in flash to the block->data and sets the base address.

Parameters

<i>block</i>	Pointer to flash block info structure to operate on.
<i>address</i>	Base address of the block data.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

Referenced by FlashAddToBlock().

7.421.2.11 FlashOperate()

```
static blt_bool FlashOperate (
    blt_int8u cmd,
    blt_addr addr,
    blt_int16u data ) [static]
```

Prepares the flash command and executes it.

Parameters

<i>cmd</i>	Command to be launched.
<i>addr</i>	Physical address for operation.
<i>data</i>	Data to write to addr for operation.

Returns

BLT_TRUE if operation was successful, otherwise BLT_FALSE.

Referenced by FlashWriteBlock().

7.421.2.12 FlashReinit()

```
void FlashReinit (
    void )
```

Reinitializes the flash driver.

Returns

none.

7.421.2.13 FlashSwitchBlock()

```
static tFlashBlockInfo * FlashSwitchBlock (
    tFlashBlockInfo * block,
    blt_addr base_addr ) [static]
```

Switches blocks by programming the current one and initializing the next.

Parameters

<i>block</i>	Pointer to flash block info structure to operate on.
<i>base_addr</i>	Base address of the next block.

Returns

The pointer of the block info struct that is no being used, or a NULL pointer in case of error.

Referenced by FlashAddToBlock().

7.421.2.14 FlashVerifyChecksum()

```
blt_bool FlashVerifyChecksum (
    void )
```

Verifies the checksum, which indicates that a valid user program is present and can be started.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.421.2.15 FlashWrite()

```
blt_bool FlashWrite (
    blt_addr addr,
    blt_int32u len,
    blt_int8u * data )
```

Writes the data to flash through a flash block manager. Note that this function also checks that no data is programmed outside the flash memory region, so the bootloader can never be overwritten.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.
<i>data</i>	Pointer to the data buffer.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.421.2.16 FlashWriteBlock()

```
static blt_bool FlashWriteBlock (
    tFlashBlockInfo * block ) [static]
```

Programs FLASH_WRITE_BLOCK_SIZE bytes to flash from the block->data array.

Parameters

<i>block</i>	Pointer to flash block info structure to operate on.
--------------	--

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

Referenced by FlashDone().

7.421.2.17 FlashWriteChecksum()

```
blt_bool FlashWriteChecksum (
    void )
```

Writes a checksum of the user program to non-volatile memory. This is performed once the entire user program has been programmed. Through the checksum, the bootloader can check if the programming session was completed, which indicates that a valid user programming is present and can be started.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.421.3 Variable Documentation

7.421.3.1 blockInfo

```
tFlashBlockInfo blockInfo [static]
```

Local variable with information about the flash block that is currently being operated on.

The smallest amount of flash that can be programmed is FLASH_WRITE_BLOCK_SIZE. A flash block manager is implemented in this driver and stores info in this variable. Whenever new data should be flashed, it is first added to a RAM buffer, which is part of this variable. Whenever the RAM buffer, which has the size of a flash block, is full or data needs to be written to a different block, the contents of the RAM buffer are programmed to flash. The flash block manager requires some software overhead, yet results in faster flash programming because data is first harvested, ideally until there is enough to program an entire flash block, before the flash device is actually operated on.

7.421.3.2 bootBlockInfo

```
tFlashBlockInfo bootBlockInfo [static]
```

Local variable with information about the flash boot block.

The first block of the user program holds the vector table, which on the STM32 is also the where the checksum is written to. Is it likely that the vector table is first flashed and then, at the end of the programming sequence, the checksum. This means that this flash block need to be written to twice. Normally this is not a problem with flash memory, as long as you write the same values to those bytes that are not supposed to be changed and the locations where you do write to are still in the erased 0xFF state. Unfortunately, writing twice to flash this way, does not work reliably on all micros. This is why we need to have an extra block, the bootblock, placed under the management of the block manager. This way is it possible to implement functionality so that the bootblock is only written to once at the end of the programming sequence.

7.421.3.3 flashExecCmd

```
const b1t_int8u flashExecCmd[] [static]
```

Initial value:

```
=
{
    0x36,
    0x34,
    0xce, 0x01, 0x00,
    0x1a, 0x05,
    0x86, 0x80,
    0x6a, 0x00,
    0xa7, 0xa7, 0xa7, 0xa7,
    0x0f, 0x00, 0x40, 0xfc,
    0x30,
    0x32,
    0x3d
}
```

Array with executable code for performing flash operations.

This array contains the machine code to perform the actual command on the flash device, such as program or erase. the code is compiler and location independent. This allows us to copy it to a ram buffer and execute the code from ram. This way the flash driver can be located in flash memory without running into problems when erasing/programming the same flash block that contains the flash driver. the source code for the machine code is as follows: `// launch the command FLASH->fstat = CBEIF_BIT; // wait at least 4 cycles (per AN2720) asm("nop"); asm("nop"); asm("nop"); // wait for command to complete while ((FLASH->fstat & CCIF_BIT) != CCI←F_BIT);`

Referenced by `FlashDone()`, and `FlashExecuteCommand()`.

7.421.3.4 flashLayout

```
const tFlashSector flashLayout[] [static]
```

If desired, it is possible to set `BOOT_FLASH_CUSTOM_LAYOUT_ENABLE` to `> 0` in `blt_conf.h` and then implement your own version of the `flashLayout[]` table in a source-file with the name `flash_layout.c`. This way you customize the flash memory size reserved for the bootloader, without having to modify the `flashLayout[]` table in this file directly. This file will then include `flash_layout.c` so there is no need to compile it additionally with your project.

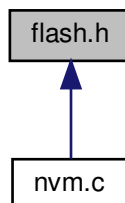
Array wit the layout of the flash memory.

Also controls what part of the flash memory is reserved for the bootloader. If the bootloader size changes, the reserved sectors for the bootloader might need adjustment to make sure the bootloader doesn't get overwritten. This layout uses linear addresses only. For example, the first address on page 0x3F is: `0x3F * 0x4000` (page size) = `0xFC000`. Note that page 0x3F is where the bootloader also resides and it has been entered as 8 chunks of 2kb. This allows flexibility for reserving more/less space for the bootloader in case its size changes in the future.

7.422 flash.h File Reference

Bootloader flash driver header file.

This graph shows which files directly or indirectly include this file:



Functions

- void [FlashInit](#) (void)
Initializes the flash driver.
- void [FlashReinit](#) (void)
Reinitializes the flash driver.
- [blt_bool](#) [FlashWrite](#) ([blt_addr](#) addr, [blt_int32u](#) len, [blt_int8u](#) *data)
Writes the data to flash through a flash block manager. Note that this function also checks that no data is programmed outside the flash memory region, so the bootloader can never be overwritten.
- [blt_bool](#) [FlashErase](#) ([blt_addr](#) addr, [blt_int32u](#) len)
Erases the flash memory. Note that this function also checks that no data is erased outside the flash memory region, so the bootloader can never be erased.
- [blt_bool](#) [FlashWriteChecksum](#) (void)
Writes a checksum of the user program to non-volatile memory. This is performed once the entire user program has been programmed. Through the checksum, the bootloader can check if the programming session was completed, which indicates that a valid user programming is present and can be started.
- [blt_bool](#) [FlashVerifyChecksum](#) (void)
Verifies the checksum, which indicates that a valid user program is present and can be started.
- [blt_bool](#) [FlashDone](#) (void)
Finalizes the flash driver operations. There could still be data in the currently active block that needs to be flashed.
- [blt_addr](#) [FlashGetUserProgBaseAddress](#) (void)
Obtains the base address of the flash memory available to the user program. This is basically the first address in the flashLayout table.

7.422.1 Detailed Description

Bootloader flash driver header file.

7.422.2 Function Documentation

7.422.2.1 [FlashDone\(\)](#)

```
blt\_bool FlashDone (
    void )
```

Finalizes the flash driver operations. There could still be data in the currently active block that needs to be flashed.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.422.2.2 [FlashErase\(\)](#)

```
blt\_bool FlashErase (
    blt\_addr addr,
    blt\_int32u len )
```

Erases the flash memory. Note that this function also checks that no data is erased outside the flash memory region, so the bootloader can never be erased.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

Erases the flash memory. Note that this function also checks that no data is erased outside the flash memory region, so the bootloader can never be erased.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.422.2.3 FlashGetUserProgBaseAddress()

```
b1t_addr FlashGetUserProgBaseAddress (
    void )
```

Obtains the base address of the flash memory available to the user program. This is basically the first address in the flashLayout table.

Returns

Base address.

Obtains the base address of the flash memory available to the user program. This is basically the first address in the flashLayout table.

Returns

Base address.

7.422.2.4 FlashInit()

```
void FlashInit (
    void )
```

Initializes the flash driver.

Returns

none.

7.422.2.5 FlashReinit()

```
void FlashReinit (
    void )
```

Reinitializes the flash driver.

Returns

none.

7.422.2.6 FlashVerifyChecksum()

```
blt_bool FlashVerifyChecksum (
    void )
```

Verifies the checksum, which indicates that a valid user program is present and can be started.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.422.2.7 FlashWrite()

```
blt_bool FlashWrite (
    blt_addr addr,
    blt_int32u len,
    blt_int8u * data )
```

Writes the data to flash through a flash block manager. Note that this function also checks that no data is programmed outside the flash memory region, so the bootloader can never be overwritten.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.
<i>data</i>	Pointer to the data buffer.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.422.2.8 FlashWriteChecksum()

```
blt_bool FlashWriteChecksum (
    void )
```

Writes a checksum of the user program to non-volatile memory. This is performed once the entire user program has been programmed. Through the checksum, the bootloader can check if the programming session was completed, which indicates that a valid user programming is present and can be started.

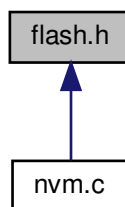
Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.423 flash.h File Reference

Bootloader flash driver header file.

This graph shows which files directly or indirectly include this file:



Functions

- void **FlashInit** (void)
Initializes the flash driver.
- void **FlashReinit** (void)
Reinitializes the flash driver.
- **blt_bool** **FlashWrite** (**blt_addr** addr, **blt_int32u** len, **blt_int8u** *data)
Writes the data to flash through a flash block manager. Note that this function also checks that no data is programmed outside the flash memory region, so the bootloader can never be overwritten.
- **blt_bool** **FlashErase** (**blt_addr** addr, **blt_int32u** len)
Erases the flash memory. Note that this function also checks that no data is erased outside the flash memory region, so the bootloader can never be erased.
- **blt_bool** **FlashWriteChecksum** (void)
Writes a checksum of the user program to non-volatile memory. This is performed once the entire user program has been programmed. Through the checksum, the bootloader can check if the programming session was completed, which indicates that a valid user programming is present and can be started.
- **blt_bool** **FlashVerifyChecksum** (void)
Verifies the checksum, which indicates that a valid user program is present and can be started.
- **blt_bool** **FlashDone** (void)
Finalizes the flash driver operations. There could still be data in the currently active block that needs to be flashed.
- **blt_addr** **FlashGetUserProgBaseAddress** (void)
Obtains the base address of the flash memory available to the user program. This is basically the first address in the flashLayout table.

7.423.1 Detailed Description

Bootloader flash driver header file.

7.423.2 Function Documentation

7.423.2.1 FlashDone()

```
blt_bool FlashDone (
    void )
```

Finalizes the flash driver operations. There could still be data in the currently active block that needs to be flashed.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.423.2.2 FlashErase()

```
blt_bool FlashErase (
    blt_addr addr,
    blt_int32u len )
```

Erases the flash memory. Note that this function also checks that no data is erased outside the flash memory region, so the bootloader can never be erased.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

Erases the flash memory. Note that this function also checks that no data is erased outside the flash memory region, so the bootloader can never be erased.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.423.2.3 FlashGetUserProgBaseAddress()

```
blt_addr FlashGetUserProgBaseAddress (
    void )
```

Obtains the base address of the flash memory available to the user program. This is basically the first address in the flashLayout table.

Returns

Base address.

Obtains the base address of the flash memory available to the user program. This is basically the first address in the flashLayout table.

Returns

Base address.

7.423.2.4 FlashInit()

```
void FlashInit (
    void )
```

Initializes the flash driver.

Returns

none.

7.423.2.5 FlashReinit()

```
void FlashReinit (
    void )
```

Reinitializes the flash driver.

Returns

none.

7.423.2.6 FlashVerifyChecksum()

```
blt_bool FlashVerifyChecksum (
    void )
```

Verifies the checksum, which indicates that a valid user program is present and can be started.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.423.2.7 FlashWrite()

```
blt_bool FlashWrite (
    blt_addr addr,
    blt_int32u len,
    blt_int8u * data )
```

Writes the data to flash through a flash block manager. Note that this function also checks that no data is programmed outside the flash memory region, so the bootloader can never be overwritten.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.
<i>data</i>	Pointer to the data buffer.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.423.2.8 FlashWriteChecksum()

```
blt_bool FlashWriteChecksum (
    void )
```

Writes a checksum of the user program to non-volatile memory. This is performed once the entire user program has been programmed. Through the checksum, the bootloader can check if the programming session was completed, which indicates that a valid user programming is present and can be started.

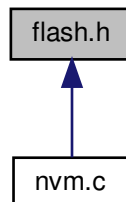
Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.424 flash.h File Reference

Bootloader flash driver header file.

This graph shows which files directly or indirectly include this file:



Functions

- void [FlashInit](#) (void)
Initializes the flash driver.
- void [FlashReinit](#) (void)
Reinitializes the flash driver.
- [blt_bool](#) [FlashWrite](#) ([blt_addr](#) addr, [blt_int32u](#) len, [blt_int8u](#) *data)
Writes the data to flash through a flash block manager. Note that this function also checks that no data is programmed outside the flash memory region, so the bootloader can never be overwritten.
- [blt_bool](#) [FlashErase](#) ([blt_addr](#) addr, [blt_int32u](#) len)
Erases the flash memory. Note that this function also checks that no data is erased outside the flash memory region, so the bootloader can never be erased.
- [blt_bool](#) [FlashWriteChecksum](#) (void)
Writes a checksum of the user program to non-volatile memory. This is performed once the entire user program has been programmed. Through the checksum, the bootloader can check if the programming session was completed, which indicates that a valid user programming is present and can be started.
- [blt_bool](#) [FlashVerifyChecksum](#) (void)
Verifies the checksum, which indicates that a valid user program is present and can be started.
- [blt_bool](#) [FlashDone](#) (void)
Finalizes the flash driver operations. There could still be data in the currently active block that needs to be flashed.
- [blt_addr](#) [FlashGetUserProgBaseAddress](#) (void)
Obtains the base address of the flash memory available to the user program. This is basically the first address in the flashLayout table.

7.424.1 Detailed Description

Bootloader flash driver header file.

7.424.2 Function Documentation

7.424.2.1 FlashDone()

```
blt_bool FlashDone (
    void )
```

Finalizes the flash driver operations. There could still be data in the currently active block that needs to be flashed.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.424.2.2 FlashErase()

```
blt_bool FlashErase (
    blt_addr addr,
    blt_int32u len )
```

Erases the flash memory. Note that this function also checks that no data is erased outside the flash memory region, so the bootloader can never be erased.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

Erases the flash memory. Note that this function also checks that no data is erased outside the flash memory region, so the bootloader can never be erased.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.424.2.3 FlashGetUserProgBaseAddress()

```
blt_addr FlashGetUserProgBaseAddress (
    void )
```

Obtains the base address of the flash memory available to the user program. This is basically the first address in the flashLayout table.

Returns

Base address.

Obtains the base address of the flash memory available to the user program. This is basically the first address in the flashLayout table.

Returns

Base address.

7.424.2.4 FlashInit()

```
void FlashInit (
    void )
```

Initializes the flash driver.

Returns

none.

7.424.2.5 FlashReinit()

```
void FlashReinit (
    void )
```

Reinitializes the flash driver.

Returns

none.

7.424.2.6 FlashVerifyChecksum()

```
blt_bool FlashVerifyChecksum (
    void )
```

Verifies the checksum, which indicates that a valid user program is present and can be started.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.424.2.7 FlashWrite()

```
blt_bool FlashWrite (
    blt_addr addr,
    blt_int32u len,
    blt_int8u * data )
```

Writes the data to flash through a flash block manager. Note that this function also checks that no data is programmed outside the flash memory region, so the bootloader can never be overwritten.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.
<i>data</i>	Pointer to the data buffer.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.424.2.8 FlashWriteChecksum()

```
blt_bool FlashWriteChecksum (  
    void )
```

Writes a checksum of the user program to non-volatile memory. This is performed once the entire user program has been programmed. Through the checksum, the bootloader can check if the programming session was completed, which indicates that a valid user programming is present and can be started.

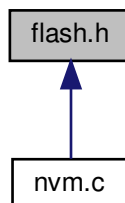
Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.425 flash.h File Reference

Bootloader flash driver header file.

This graph shows which files directly or indirectly include this file:



Functions

- void [FlashInit](#) (void)
Initializes the flash driver.
- void [FlashReinit](#) (void)
Reinitializes the flash driver.
- [blt_bool](#) [FlashWrite](#) ([blt_addr](#) addr, [blt_int32u](#) len, [blt_int8u](#) *data)
Writes the data to flash through a flash block manager. Note that this function also checks that no data is programmed outside the flash memory region, so the bootloader can never be overwritten.
- [blt_bool](#) [FlashErase](#) ([blt_addr](#) addr, [blt_int32u](#) len)
Erases the flash memory. Note that this function also checks that no data is erased outside the flash memory region, so the bootloader can never be erased.
- [blt_bool](#) [FlashWriteChecksum](#) (void)
Writes a checksum of the user program to non-volatile memory. This is performed once the entire user program has been programmed. Through the checksum, the bootloader can check if the programming session was completed, which indicates that a valid user programming is present and can be started.
- [blt_bool](#) [FlashVerifyChecksum](#) (void)
Verifies the checksum, which indicates that a valid user program is present and can be started.
- [blt_bool](#) [FlashDone](#) (void)
Finalizes the flash driver operations. There could still be data in the currently active block that needs to be flashed.
- [blt_addr](#) [FlashGetUserProgBaseAddress](#) (void)
Obtains the base address of the flash memory available to the user program. This is basically the first address in the flashLayout table.

7.425.1 Detailed Description

Bootloader flash driver header file.

7.425.2 Function Documentation

7.425.2.1 [FlashDone\(\)](#)

```
blt\_bool FlashDone (
    void )
```

Finalizes the flash driver operations. There could still be data in the currently active block that needs to be flashed.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.425.2.2 [FlashErase\(\)](#)

```
blt\_bool FlashErase (
    blt\_addr addr,
    blt\_int32u len )
```

Erases the flash memory. Note that this function also checks that no data is erased outside the flash memory region, so the bootloader can never be erased.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

Erases the flash memory. Note that this function also checks that no data is erased outside the flash memory region, so the bootloader can never be erased.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.425.2.3 FlashGetUserProgBaseAddress()

```
blt_addr FlashGetUserProgBaseAddress (
    void )
```

Obtains the base address of the flash memory available to the user program. This is basically the first address in the flashLayout table.

Returns

Base address.

Obtains the base address of the flash memory available to the user program. This is basically the first address in the flashLayout table.

Returns

Base address.

7.425.2.4 FlashInit()

```
void FlashInit (
    void )
```

Initializes the flash driver.

Returns

none.

7.425.2.5 FlashReinit()

```
void FlashReinit (
    void )
```

Reinitializes the flash driver.

Returns

none.

7.425.2.6 FlashVerifyChecksum()

```
blt_bool FlashVerifyChecksum (
    void )
```

Verifies the checksum, which indicates that a valid user program is present and can be started.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.425.2.7 FlashWrite()

```
blt_bool FlashWrite (
    blt_addr addr,
    blt_int32u len,
    blt_int8u * data )
```

Writes the data to flash through a flash block manager. Note that this function also checks that no data is programmed outside the flash memory region, so the bootloader can never be overwritten.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.
<i>data</i>	Pointer to the data buffer.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.425.2.8 FlashWriteChecksum()

```
blt_bool FlashWriteChecksum (
    void )
```

Writes a checksum of the user program to non-volatile memory. This is performed once the entire user program has been programmed. Through the checksum, the bootloader can check if the programming session was completed, which indicates that a valid user programming is present and can be started.

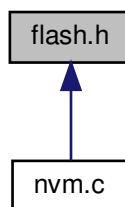
Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.426 flash.h File Reference

Bootloader flash driver header file.

This graph shows which files directly or indirectly include this file:



Functions

- void **FlashInit** (void)
Initializes the flash driver.
- void **FlashReinit** (void)
Reinitializes the flash driver.
- **blt_bool FlashWrite** (**blt_addr** addr, **blt_int32u** len, **blt_int8u** *data)
Writes the data to flash through a flash block manager. Note that this function also checks that no data is programmed outside the flash memory region, so the bootloader can never be overwritten.
- **blt_bool FlashErase** (**blt_addr** addr, **blt_int32u** len)
Erases the flash memory. Note that this function also checks that no data is erased outside the flash memory region, so the bootloader can never be erased.
- **blt_bool FlashWriteChecksum** (void)
Writes a checksum of the user program to non-volatile memory. This is performed once the entire user program has been programmed. Through the checksum, the bootloader can check if the programming session was completed, which indicates that a valid user programming is present and can be started.
- **blt_bool FlashVerifyChecksum** (void)
Verifies the checksum, which indicates that a valid user program is present and can be started.
- **blt_bool FlashDone** (void)
Finalizes the flash driver operations. There could still be data in the currently active block that needs to be flashed.
- **blt_addr FlashGetUserProgBaseAddress** (void)
Obtains the base address of the flash memory available to the user program. This is basically the first address in the flashLayout table.

7.426.1 Detailed Description

Bootloader flash driver header file.

7.426.2 Function Documentation

7.426.2.1 FlashDone()

```
blt_bool FlashDone (  
    void )
```

Finalizes the flash driver operations. There could still be data in the currently active block that needs to be flashed.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.426.2.2 FlashErase()

```
blt_bool FlashErase (  
    blt_addr addr,  
    blt_int32u len )
```

Erases the flash memory. Note that this function also checks that no data is erased outside the flash memory region, so the bootloader can never be erased.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

Erases the flash memory. Note that this function also checks that no data is erased outside the flash memory region, so the bootloader can never be erased.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.426.2.3 FlashGetUserProgBaseAddress()

```
blt_addr FlashGetUserProgBaseAddress (
    void )
```

Obtains the base address of the flash memory available to the user program. This is basically the first address in the flashLayout table.

Returns

Base address.

Obtains the base address of the flash memory available to the user program. This is basically the first address in the flashLayout table.

Returns

Base address.

7.426.2.4 FlashInit()

```
void FlashInit (
    void )
```

Initializes the flash driver.

Returns

none.

7.426.2.5 FlashReinit()

```
void FlashReinit (
    void )
```

Reinitializes the flash driver.

Returns

none.

7.426.2.6 FlashVerifyChecksum()

```
blt_bool FlashVerifyChecksum (
    void )
```

Verifies the checksum, which indicates that a valid user program is present and can be started.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.426.2.7 FlashWrite()

```
blt_bool FlashWrite (
    blt_addr addr,
    blt_int32u len,
    blt_int8u * data )
```

Writes the data to flash through a flash block manager. Note that this function also checks that no data is programmed outside the flash memory region, so the bootloader can never be overwritten.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.
<i>data</i>	Pointer to the data buffer.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.426.2.8 FlashWriteChecksum()

```
blt_bool FlashWriteChecksum (
    void )
```

Writes a checksum of the user program to non-volatile memory. This is performed once the entire user program has been programmed. Through the checksum, the bootloader can check if the programming session was completed, which indicates that a valid user programming is present and can be started.

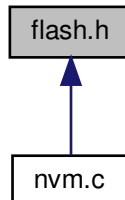
Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.427 flash.h File Reference

Bootloader flash driver header file.

This graph shows which files directly or indirectly include this file:



Functions

- void [FlashInit](#) (void)
Initializes the flash driver.
- void [FlashReinit](#) (void)
Reinitializes the flash driver.
- [blt_bool](#) [FlashWrite](#) ([blt_addr](#) addr, [blt_int32u](#) len, [blt_int8u](#) *data)
Writes the data to flash through a flash block manager. Note that this function also checks that no data is programmed outside the flash memory region, so the bootloader can never be overwritten.
- [blt_bool](#) [FlashErase](#) ([blt_addr](#) addr, [blt_int32u](#) len)
Erases the flash memory. Note that this function also checks that no data is erased outside the flash memory region, so the bootloader can never be erased.
- [blt_bool](#) [FlashWriteChecksum](#) (void)
Writes a checksum of the user program to non-volatile memory. This is performed once the entire user program has been programmed. Through the checksum, the bootloader can check if the programming session was completed, which indicates that a valid user programming is present and can be started.
- [blt_bool](#) [FlashVerifyChecksum](#) (void)
Verifies the checksum, which indicates that a valid user program is present and can be started.
- [blt_bool](#) [FlashDone](#) (void)
Finalizes the flash driver operations. There could still be data in the currently active block that needs to be flashed.
- [blt_addr](#) [FlashGetUserProgBaseAddress](#) (void)
Obtains the base address of the flash memory available to the user program. This is basically the first address in the flashLayout table.

7.427.1 Detailed Description

Bootloader flash driver header file.

7.427.2 Function Documentation

7.427.2.1 FlashDone()

```
blt_bool FlashDone (
    void )
```

Finalizes the flash driver operations. There could still be data in the currently active block that needs to be flashed.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.427.2.2 FlashErase()

```
blt_bool FlashErase (
    blt_addr addr,
    blt_int32u len )
```

Erases the flash memory. Note that this function also checks that no data is erased outside the flash memory region, so the bootloader can never be erased.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

Erases the flash memory. Note that this function also checks that no data is erased outside the flash memory region, so the bootloader can never be erased.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.427.2.3 FlashGetUserProgBaseAddress()

```
blt_addr FlashGetUserProgBaseAddress (
    void )
```

Obtains the base address of the flash memory available to the user program. This is basically the first address in the flashLayout table.

Returns

Base address.

Obtains the base address of the flash memory available to the user program. This is basically the first address in the flashLayout table.

Returns

Base address.

7.427.2.4 FlashInit()

```
void FlashInit (
    void )
```

Initializes the flash driver.

Returns

none.

7.427.2.5 FlashReinit()

```
void FlashReinit (
    void )
```

Reinitializes the flash driver.

Returns

none.

7.427.2.6 FlashVerifyChecksum()

```
blt_bool FlashVerifyChecksum (
    void )
```

Verifies the checksum, which indicates that a valid user program is present and can be started.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.427.2.7 FlashWrite()

```
blt_bool FlashWrite (
    blt_addr addr,
    blt_int32u len,
    blt_int8u * data )
```

Writes the data to flash through a flash block manager. Note that this function also checks that no data is programmed outside the flash memory region, so the bootloader can never be overwritten.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.
<i>data</i>	Pointer to the data buffer.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.427.2.8 FlashWriteChecksum()

```
blt_bool FlashWriteChecksum (  
    void )
```

Writes a checksum of the user program to non-volatile memory. This is performed once the entire user program has been programmed. Through the checksum, the bootloader can check if the programming session was completed, which indicates that a valid user programming is present and can be started.

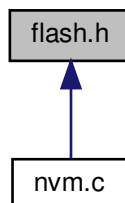
Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.428 flash.h File Reference

Bootloader flash driver header file.

This graph shows which files directly or indirectly include this file:



Functions

- void [FlashInit](#) (void)
Initializes the flash driver.
- void [FlashReinit](#) (void)
Reinitializes the flash driver.
- [blt_bool](#) [FlashWrite](#) ([blt_addr](#) addr, [blt_int32u](#) len, [blt_int8u](#) *data)
Writes the data to flash through a flash block manager. Note that this function also checks that no data is programmed outside the flash memory region, so the bootloader can never be overwritten.
- [blt_bool](#) [FlashErase](#) ([blt_addr](#) addr, [blt_int32u](#) len)
Erases the flash memory. Note that this function also checks that no data is erased outside the flash memory region, so the bootloader can never be erased.
- [blt_bool](#) [FlashWriteChecksum](#) (void)
Writes a checksum of the user program to non-volatile memory. This is performed once the entire user program has been programmed. Through the checksum, the bootloader can check if the programming session was completed, which indicates that a valid user programming is present and can be started.
- [blt_bool](#) [FlashVerifyChecksum](#) (void)
Verifies the checksum, which indicates that a valid user program is present and can be started.
- [blt_bool](#) [FlashDone](#) (void)
Finalizes the flash driver operations. There could still be data in the currently active block that needs to be flashed.
- [blt_addr](#) [FlashGetUserProgBaseAddress](#) (void)
Obtains the base address of the flash memory available to the user program. This is basically the first address in the flashLayout table.

7.428.1 Detailed Description

Bootloader flash driver header file.

7.428.2 Function Documentation

7.428.2.1 [FlashDone\(\)](#)

```
blt\_bool FlashDone (
    void )
```

Finalizes the flash driver operations. There could still be data in the currently active block that needs to be flashed.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.428.2.2 [FlashErase\(\)](#)

```
blt\_bool FlashErase (
    blt\_addr addr,
    blt\_int32u len )
```

Erases the flash memory. Note that this function also checks that no data is erased outside the flash memory region, so the bootloader can never be erased.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

Erases the flash memory. Note that this function also checks that no data is erased outside the flash memory region, so the bootloader can never be erased.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.428.2.3 FlashGetUserProgBaseAddress()

```
blt_addr FlashGetUserProgBaseAddress (
    void )
```

Obtains the base address of the flash memory available to the user program. This is basically the first address in the flashLayout table.

Returns

Base address.

Obtains the base address of the flash memory available to the user program. This is basically the first address in the flashLayout table.

Returns

Base address.

7.428.2.4 FlashInit()

```
void FlashInit (
    void )
```

Initializes the flash driver.

Returns

none.

7.428.2.5 FlashReinit()

```
void FlashReinit (
    void )
```

Reinitializes the flash driver.

Returns

none.

7.428.2.6 FlashVerifyChecksum()

```
blt_bool FlashVerifyChecksum (
    void )
```

Verifies the checksum, which indicates that a valid user program is present and can be started.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.428.2.7 FlashWrite()

```
blt_bool FlashWrite (
    blt_addr addr,
    blt_int32u len,
    blt_int8u * data )
```

Writes the data to flash through a flash block manager. Note that this function also checks that no data is programmed outside the flash memory region, so the bootloader can never be overwritten.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.
<i>data</i>	Pointer to the data buffer.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.428.2.8 FlashWriteChecksum()

```
blt_bool FlashWriteChecksum (
    void )
```

Writes a checksum of the user program to non-volatile memory. This is performed once the entire user program has been programmed. Through the checksum, the bootloader can check if the programming session was completed, which indicates that a valid user programming is present and can be started.

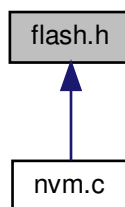
Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.429 flash.h File Reference

Bootloader flash driver header file.

This graph shows which files directly or indirectly include this file:



Functions

- void **FlashInit** (void)
Initializes the flash driver.
- void **FlashReinit** (void)
Reinitializes the flash driver.
- blt_bool **FlashWrite** (blt_addr addr, blt_int32u len, blt_int8u *data)
Writes the data to flash through a flash block manager. Note that this function also checks that no data is programmed outside the flash memory region, so the bootloader can never be overwritten.
- blt_bool **FlashErase** (blt_addr addr, blt_int32u len)
Erases the flash memory. Note that this function also checks that no data is erased outside the flash memory region, so the bootloader can never be erased.
- blt_bool **FlashWriteChecksum** (void)
Writes a checksum of the user program to non-volatile memory. This is performed once the entire user program has been programmed. Through the checksum, the bootloader can check if the programming session was completed, which indicates that a valid user programming is present and can be started.
- blt_bool **FlashVerifyChecksum** (void)
Verifies the checksum, which indicates that a valid user program is present and can be started.
- blt_bool **FlashDone** (void)
Finalizes the flash driver operations. There could still be data in the currently active block that needs to be flashed.
- blt_addr **FlashGetUserProgBaseAddress** (void)
Obtains the base address of the flash memory available to the user program. This is basically the first address in the flashLayout table.

7.429.1 Detailed Description

Bootloader flash driver header file.

7.429.2 Function Documentation

7.429.2.1 FlashDone()

```
blt_bool FlashDone (  
    void )
```

Finalizes the flash driver operations. There could still be data in the currently active block that needs to be flashed.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.429.2.2 FlashErase()

```
blt_bool FlashErase (  
    blt_addr addr,  
    blt_int32u len )
```

Erases the flash memory. Note that this function also checks that no data is erased outside the flash memory region, so the bootloader can never be erased.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

Erases the flash memory. Note that this function also checks that no data is erased outside the flash memory region, so the bootloader can never be erased.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.429.2.3 FlashGetUserProgBaseAddress()

```
blt_addr FlashGetUserProgBaseAddress (
    void )
```

Obtains the base address of the flash memory available to the user program. This is basically the first address in the flashLayout table.

Returns

Base address.

Obtains the base address of the flash memory available to the user program. This is basically the first address in the flashLayout table.

Returns

Base address.

7.429.2.4 FlashInit()

```
void FlashInit (
    void )
```

Initializes the flash driver.

Returns

none.

7.429.2.5 FlashReinit()

```
void FlashReinit (
    void )
```

Reinitializes the flash driver.

Returns

none.

7.429.2.6 FlashVerifyChecksum()

```
blt_bool FlashVerifyChecksum (
    void )
```

Verifies the checksum, which indicates that a valid user program is present and can be started.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.429.2.7 FlashWrite()

```
blt_bool FlashWrite (
    blt_addr addr,
    blt_int32u len,
    blt_int8u * data )
```

Writes the data to flash through a flash block manager. Note that this function also checks that no data is programmed outside the flash memory region, so the bootloader can never be overwritten.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.
<i>data</i>	Pointer to the data buffer.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.429.2.8 FlashWriteChecksum()

```
blt_bool FlashWriteChecksum (
    void )
```

Writes a checksum of the user program to non-volatile memory. This is performed once the entire user program has been programmed. Through the checksum, the bootloader can check if the programming session was completed, which indicates that a valid user programming is present and can be started.

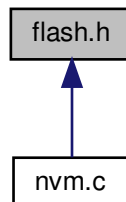
Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.430 flash.h File Reference

Bootloader flash driver header file.

This graph shows which files directly or indirectly include this file:



Functions

- void [FlashInit](#) (void)
Initializes the flash driver.
- void [FlashReinit](#) (void)
Reinitializes the flash driver.
- [blt_bool](#) [FlashWrite](#) ([blt_addr](#) addr, [blt_int32u](#) len, [blt_int8u](#) *data)
Writes the data to flash through a flash block manager. Note that this function also checks that no data is programmed outside the flash memory region, so the bootloader can never be overwritten.
- [blt_bool](#) [FlashErase](#) ([blt_addr](#) addr, [blt_int32u](#) len)
Erases the flash memory. Note that this function also checks that no data is erased outside the flash memory region, so the bootloader can never be erased.
- [blt_bool](#) [FlashWriteChecksum](#) (void)
Writes a checksum of the user program to non-volatile memory. This is performed once the entire user program has been programmed. Through the checksum, the bootloader can check if the programming session was completed, which indicates that a valid user programming is present and can be started.
- [blt_bool](#) [FlashVerifyChecksum](#) (void)
Verifies the checksum, which indicates that a valid user program is present and can be started.
- [blt_bool](#) [FlashDone](#) (void)
Finalizes the flash driver operations. There could still be data in the currently active block that needs to be flashed.
- [blt_addr](#) [FlashGetUserProgBaseAddress](#) (void)
Obtains the base address of the flash memory available to the user program. This is basically the first address in the flashLayout table.

7.430.1 Detailed Description

Bootloader flash driver header file.

7.430.2 Function Documentation

7.430.2.1 FlashDone()

```
blt_bool FlashDone (
    void )
```

Finalizes the flash driver operations. There could still be data in the currently active block that needs to be flashed.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.430.2.2 FlashErase()

```
blt_bool FlashErase (
    blt_addr addr,
    blt_int32u len )
```

Erases the flash memory. Note that this function also checks that no data is erased outside the flash memory region, so the bootloader can never be erased.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

Erases the flash memory. Note that this function also checks that no data is erased outside the flash memory region, so the bootloader can never be erased.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.430.2.3 FlashGetUserProgBaseAddress()

```
blt_addr FlashGetUserProgBaseAddress (
    void )
```

Obtains the base address of the flash memory available to the user program. This is basically the first address in the flashLayout table.

Returns

Base address.

Obtains the base address of the flash memory available to the user program. This is basically the first address in the flashLayout table.

Returns

Base address.

7.430.2.4 FlashInit()

```
void FlashInit (
    void )
```

Initializes the flash driver.

Returns

none.

7.430.2.5 FlashReinit()

```
void FlashReinit (
    void )
```

Reinitializes the flash driver.

Returns

none.

7.430.2.6 FlashVerifyChecksum()

```
blt_bool FlashVerifyChecksum (
    void )
```

Verifies the checksum, which indicates that a valid user program is present and can be started.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.430.2.7 FlashWrite()

```
blt_bool FlashWrite (
    blt_addr addr,
    blt_int32u len,
    blt_int8u * data )
```

Writes the data to flash through a flash block manager. Note that this function also checks that no data is programmed outside the flash memory region, so the bootloader can never be overwritten.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.
<i>data</i>	Pointer to the data buffer.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.430.2.8 FlashWriteChecksum()

```
blt_bool FlashWriteChecksum (  
    void )
```

Writes a checksum of the user program to non-volatile memory. This is performed once the entire user program has been programmed. Through the checksum, the bootloader can check if the programming session was completed, which indicates that a valid user programming is present and can be started.

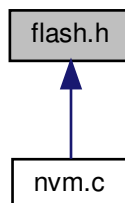
Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.431 flash.h File Reference

Bootloader flash driver header file.

This graph shows which files directly or indirectly include this file:



Functions

- void [FlashInit](#) (void)
Initializes the flash driver.
- void [FlashReinit](#) (void)
Reinitializes the flash driver.
- [blt_bool](#) [FlashWrite](#) ([blt_addr](#) addr, [blt_int32u](#) len, [blt_int8u](#) *data)
Writes the data to flash through a flash block manager. Note that this function also checks that no data is programmed outside the flash memory region, so the bootloader can never be overwritten.
- [blt_bool](#) [FlashErase](#) ([blt_addr](#) addr, [blt_int32u](#) len)
Erases the flash memory. Note that this function also checks that no data is erased outside the flash memory region, so the bootloader can never be erased.
- [blt_bool](#) [FlashWriteChecksum](#) (void)
Writes a checksum of the user program to non-volatile memory. This is performed once the entire user program has been programmed. Through the checksum, the bootloader can check if the programming session was completed, which indicates that a valid user programming is present and can be started.
- [blt_bool](#) [FlashVerifyChecksum](#) (void)
Verifies the checksum, which indicates that a valid user program is present and can be started.
- [blt_bool](#) [FlashDone](#) (void)
Finalizes the flash driver operations. There could still be data in the currently active block that needs to be flashed.
- [blt_addr](#) [FlashGetUserProgBaseAddress](#) (void)
Obtains the base address of the flash memory available to the user program. This is basically the first address in the `flashLayout` table.

7.431.1 Detailed Description

Bootloader flash driver header file.

7.431.2 Function Documentation

7.431.2.1 [FlashDone\(\)](#)

```
blt\_bool FlashDone (
    void )
```

Finalizes the flash driver operations. There could still be data in the currently active block that needs to be flashed.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.431.2.2 [FlashErase\(\)](#)

```
blt\_bool FlashErase (
    blt\_addr addr,
    blt\_int32u len )
```

Erases the flash memory. Note that this function also checks that no data is erased outside the flash memory region, so the bootloader can never be erased.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

Erases the flash memory. Note that this function also checks that no data is erased outside the flash memory region, so the bootloader can never be erased.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.431.2.3 FlashGetUserProgBaseAddress()

```
blt_addr FlashGetUserProgBaseAddress (
    void )
```

Obtains the base address of the flash memory available to the user program. This is basically the first address in the flashLayout table.

Returns

Base address.

Obtains the base address of the flash memory available to the user program. This is basically the first address in the flashLayout table.

Returns

Base address.

7.431.2.4 FlashInit()

```
void FlashInit (
    void )
```

Initializes the flash driver.

Returns

none.

7.431.2.5 FlashReinit()

```
void FlashReinit (
    void )
```

Reinitializes the flash driver.

Returns

none.

7.431.2.6 FlashVerifyChecksum()

```
blt_bool FlashVerifyChecksum (
    void )
```

Verifies the checksum, which indicates that a valid user program is present and can be started.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.431.2.7 FlashWrite()

```
blt_bool FlashWrite (
    blt_addr addr,
    blt_int32u len,
    blt_int8u * data )
```

Writes the data to flash through a flash block manager. Note that this function also checks that no data is programmed outside the flash memory region, so the bootloader can never be overwritten.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.
<i>data</i>	Pointer to the data buffer.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.431.2.8 FlashWriteChecksum()

```
blt_bool FlashWriteChecksum (
    void )
```

Writes a checksum of the user program to non-volatile memory. This is performed once the entire user program has been programmed. Through the checksum, the bootloader can check if the programming session was completed, which indicates that a valid user programming is present and can be started.

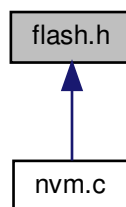
Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.432 flash.h File Reference

Bootloader flash driver header file.

This graph shows which files directly or indirectly include this file:



Functions

- void **FlashInit** (void)
Initializes the flash driver.
- void **FlashReinit** (void)
Reinitializes the flash driver.
- **blt_bool** **FlashWrite** (**blt_addr** addr, **blt_int32u** len, **blt_int8u** *data)
Writes the data to flash through a flash block manager. Note that this function also checks that no data is programmed outside the flash memory region, so the bootloader can never be overwritten.
- **blt_bool** **FlashErase** (**blt_addr** addr, **blt_int32u** len)
Erases the flash memory. Note that this function also checks that no data is erased outside the flash memory region, so the bootloader can never be erased.
- **blt_bool** **FlashWriteChecksum** (void)
Writes a checksum of the user program to non-volatile memory. This is performed once the entire user program has been programmed. Through the checksum, the bootloader can check if the programming session was completed, which indicates that a valid user programming is present and can be started.
- **blt_bool** **FlashVerifyChecksum** (void)
Verifies the checksum, which indicates that a valid user program is present and can be started.
- **blt_bool** **FlashDone** (void)
Finalizes the flash driver operations. There could still be data in the currently active block that needs to be flashed.
- **blt_addr** **FlashGetUserProgBaseAddress** (void)
Obtains the base address of the flash memory available to the user program. This is basically the first address in the flashLayout table.

7.432.1 Detailed Description

Bootloader flash driver header file.

7.432.2 Function Documentation

7.432.2.1 FlashDone()

```
blt_bool FlashDone (  
    void )
```

Finalizes the flash driver operations. There could still be data in the currently active block that needs to be flashed.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.432.2.2 FlashErase()

```
blt_bool FlashErase (  
    blt_addr addr,  
    blt_int32u len )
```

Erases the flash memory. Note that this function also checks that no data is erased outside the flash memory region, so the bootloader can never be erased.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

Erases the flash memory. Note that this function also checks that no data is erased outside the flash memory region, so the bootloader can never be erased.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.432.2.3 FlashGetUserProgBaseAddress()

```
blt_addr FlashGetUserProgBaseAddress (
    void )
```

Obtains the base address of the flash memory available to the user program. This is basically the first address in the flashLayout table.

Returns

Base address.

Obtains the base address of the flash memory available to the user program. This is basically the first address in the flashLayout table.

Returns

Base address.

7.432.2.4 FlashInit()

```
void FlashInit (
    void )
```

Initializes the flash driver.

Returns

none.

7.432.2.5 FlashReinit()

```
void FlashReinit (
    void )
```

Reinitializes the flash driver.

Returns

none.

7.432.2.6 FlashVerifyChecksum()

```
blt_bool FlashVerifyChecksum (
    void )
```

Verifies the checksum, which indicates that a valid user program is present and can be started.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.432.2.7 FlashWrite()

```
blt_bool FlashWrite (
    blt_addr addr,
    blt_int32u len,
    blt_int8u * data )
```

Writes the data to flash through a flash block manager. Note that this function also checks that no data is programmed outside the flash memory region, so the bootloader can never be overwritten.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.
<i>data</i>	Pointer to the data buffer.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.432.2.8 FlashWriteChecksum()

```
blt_bool FlashWriteChecksum (
    void )
```

Writes a checksum of the user program to non-volatile memory. This is performed once the entire user program has been programmed. Through the checksum, the bootloader can check if the programming session was completed, which indicates that a valid user programming is present and can be started.

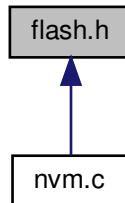
Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.433 flash.h File Reference

Bootloader flash driver header file.

This graph shows which files directly or indirectly include this file:



Functions

- void [FlashInit](#) (void)
Initializes the flash driver.
- void [FlashReinit](#) (void)
Reinitializes the flash driver.
- [blt_bool](#) [FlashWrite](#) ([blt_addr](#) addr, [blt_int32u](#) len, [blt_int8u](#) *data)
Writes the data to flash through a flash block manager. Note that this function also checks that no data is programmed outside the flash memory region, so the bootloader can never be overwritten.
- [blt_bool](#) [FlashErase](#) ([blt_addr](#) addr, [blt_int32u](#) len)
Erases the flash memory. Note that this function also checks that no data is erased outside the flash memory region, so the bootloader can never be erased.
- [blt_bool](#) [FlashWriteChecksum](#) (void)
Writes a checksum of the user program to non-volatile memory. This is performed once the entire user program has been programmed. Through the checksum, the bootloader can check if the programming session was completed, which indicates that a valid user programming is present and can be started.
- [blt_bool](#) [FlashVerifyChecksum](#) (void)
Verifies the checksum, which indicates that a valid user program is present and can be started.
- [blt_bool](#) [FlashDone](#) (void)
Finalizes the flash driver operations. There could still be data in the currently active block that needs to be flashed.
- [blt_addr](#) [FlashGetUserProgBaseAddress](#) (void)
Obtains the base address of the flash memory available to the user program. This is basically the first address in the flashLayout table.

7.433.1 Detailed Description

Bootloader flash driver header file.

7.433.2 Function Documentation

7.433.2.1 FlashDone()

```
blt_bool FlashDone (
    void )
```

Finalizes the flash driver operations. There could still be data in the currently active block that needs to be flashed.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.433.2.2 FlashErase()

```
blt_bool FlashErase (
    blt_addr addr,
    blt_int32u len )
```

Erases the flash memory. Note that this function also checks that no data is erased outside the flash memory region, so the bootloader can never be erased.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

Erases the flash memory. Note that this function also checks that no data is erased outside the flash memory region, so the bootloader can never be erased.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.433.2.3 FlashGetUserProgBaseAddress()

```
blt_addr FlashGetUserProgBaseAddress (
    void )
```

Obtains the base address of the flash memory available to the user program. This is basically the first address in the flashLayout table.

Returns

Base address.

Obtains the base address of the flash memory available to the user program. This is basically the first address in the flashLayout table.

Returns

Base address.

7.433.2.4 FlashInit()

```
void FlashInit (
    void )
```

Initializes the flash driver.

Returns

none.

7.433.2.5 FlashReinit()

```
void FlashReinit (
    void )
```

Reinitializes the flash driver.

Returns

none.

7.433.2.6 FlashVerifyChecksum()

```
blt_bool FlashVerifyChecksum (
    void )
```

Verifies the checksum, which indicates that a valid user program is present and can be started.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.433.2.7 FlashWrite()

```
blt_bool FlashWrite (
    blt_addr addr,
    blt_int32u len,
    blt_int8u * data )
```

Writes the data to flash through a flash block manager. Note that this function also checks that no data is programmed outside the flash memory region, so the bootloader can never be overwritten.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.
<i>data</i>	Pointer to the data buffer.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.433.2.8 FlashWriteChecksum()

```
blt_bool FlashWriteChecksum (  
    void )
```

Writes a checksum of the user program to non-volatile memory. This is performed once the entire user program has been programmed. Through the checksum, the bootloader can check if the programming session was completed, which indicates that a valid user programming is present and can be started.

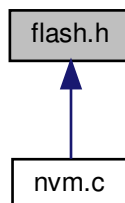
Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.434 flash.h File Reference

Bootloader flash driver header file.

This graph shows which files directly or indirectly include this file:



Functions

- void [FlashInit](#) (void)
Initializes the flash driver.
- void [FlashReinit](#) (void)
Reinitializes the flash driver.
- [blt_bool](#) [FlashWrite](#) ([blt_addr](#) addr, [blt_int32u](#) len, [blt_int8u](#) *data)
Writes the data to flash through a flash block manager. Note that this function also checks that no data is programmed outside the flash memory region, so the bootloader can never be overwritten.
- [blt_bool](#) [FlashErase](#) ([blt_addr](#) addr, [blt_int32u](#) len)
Erases the flash memory. Note that this function also checks that no data is erased outside the flash memory region, so the bootloader can never be erased.
- [blt_bool](#) [FlashWriteChecksum](#) (void)
Writes a checksum of the user program to non-volatile memory. This is performed once the entire user program has been programmed. Through the checksum, the bootloader can check if the programming session was completed, which indicates that a valid user programming is present and can be started.
- [blt_bool](#) [FlashVerifyChecksum](#) (void)
Verifies the checksum, which indicates that a valid user program is present and can be started.
- [blt_bool](#) [FlashDone](#) (void)
Finalizes the flash driver operations. There could still be data in the currently active block that needs to be flashed.
- [blt_addr](#) [FlashGetUserProgBaseAddress](#) (void)
Obtains the base address of the flash memory available to the user program. This is basically the first address in the flashLayout table.

7.434.1 Detailed Description

Bootloader flash driver header file.

7.434.2 Function Documentation

7.434.2.1 [FlashDone\(\)](#)

```
blt\_bool FlashDone (
    void )
```

Finalizes the flash driver operations. There could still be data in the currently active block that needs to be flashed.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.434.2.2 [FlashErase\(\)](#)

```
blt\_bool FlashErase (
    blt\_addr addr,
    blt\_int32u len )
```

Erases the flash memory. Note that this function also checks that no data is erased outside the flash memory region, so the bootloader can never be erased.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

Erases the flash memory. Note that this function also checks that no data is erased outside the flash memory region, so the bootloader can never be erased.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.434.2.3 FlashGetUserProgBaseAddress()

```
blt_addr FlashGetUserProgBaseAddress (
    void )
```

Obtains the base address of the flash memory available to the user program. This is basically the first address in the flashLayout table.

Returns

Base address.

Obtains the base address of the flash memory available to the user program. This is basically the first address in the flashLayout table.

Returns

Base address.

7.434.2.4 FlashInit()

```
void FlashInit (
    void )
```

Initializes the flash driver.

Returns

none.

7.434.2.5 FlashReinit()

```
void FlashReinit (
    void )
```

Reinitializes the flash driver.

Returns

none.

7.434.2.6 FlashVerifyChecksum()

```
blt_bool FlashVerifyChecksum (
    void )
```

Verifies the checksum, which indicates that a valid user program is present and can be started.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.434.2.7 FlashWrite()

```
blt_bool FlashWrite (
    blt_addr addr,
    blt_int32u len,
    blt_int8u * data )
```

Writes the data to flash through a flash block manager. Note that this function also checks that no data is programmed outside the flash memory region, so the bootloader can never be overwritten.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.
<i>data</i>	Pointer to the data buffer.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.434.2.8 FlashWriteChecksum()

```
blt_bool FlashWriteChecksum (
    void )
```

Writes a checksum of the user program to non-volatile memory. This is performed once the entire user program has been programmed. Through the checksum, the bootloader can check if the programming session was completed, which indicates that a valid user programming is present and can be started.

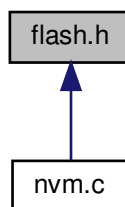
Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.435 flash.h File Reference

Bootloader flash driver header file.

This graph shows which files directly or indirectly include this file:



Functions

- void **FlashInit** (void)
Initializes the flash driver.
- void **FlashReinit** (void)
Reinitializes the flash driver.
- **blt_bool** **FlashWrite** (**blt_addr** addr, **blt_int32u** len, **blt_int8u** *data)
Writes the data to flash through a flash block manager. Note that this function also checks that no data is programmed outside the flash memory region, so the bootloader can never be overwritten.
- **blt_bool** **FlashErase** (**blt_addr** addr, **blt_int32u** len)
Erases the flash memory. Note that this function also checks that no data is erased outside the flash memory region, so the bootloader can never be erased.
- **blt_bool** **FlashWriteChecksum** (void)
Writes a checksum of the user program to non-volatile memory. This is performed once the entire user program has been programmed. Through the checksum, the bootloader can check if the programming session was completed, which indicates that a valid user programming is present and can be started.
- **blt_bool** **FlashVerifyChecksum** (void)
Verifies the checksum, which indicates that a valid user program is present and can be started.
- **blt_bool** **FlashDone** (void)
Finalizes the flash driver operations. There could still be data in the currently active block that needs to be flashed.
- **blt_addr** **FlashGetUserProgBaseAddress** (void)
Obtains the base address of the flash memory available to the user program. This is basically the first address in the flashLayout table.

7.435.1 Detailed Description

Bootloader flash driver header file.

7.435.2 Function Documentation

7.435.2.1 FlashDone()

```
blt_bool FlashDone (  
    void )
```

Finalizes the flash driver operations. There could still be data in the currently active block that needs to be flashed.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.435.2.2 FlashErase()

```
blt_bool FlashErase (  
    blt_addr addr,  
    blt_int32u len )
```

Erases the flash memory. Note that this function also checks that no data is erased outside the flash memory region, so the bootloader can never be erased.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

Erases the flash memory. Note that this function also checks that no data is erased outside the flash memory region, so the bootloader can never be erased.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.435.2.3 FlashGetUserProgBaseAddress()

```
blt_addr FlashGetUserProgBaseAddress (
    void )
```

Obtains the base address of the flash memory available to the user program. This is basically the first address in the flashLayout table.

Returns

Base address.

Obtains the base address of the flash memory available to the user program. This is basically the first address in the flashLayout table.

Returns

Base address.

7.435.2.4 FlashInit()

```
void FlashInit (
    void )
```

Initializes the flash driver.

Returns

none.

7.435.2.5 FlashReinit()

```
void FlashReinit (
    void )
```

Reinitializes the flash driver.

Returns

none.

7.435.2.6 FlashVerifyChecksum()

```
blt_bool FlashVerifyChecksum (
    void )
```

Verifies the checksum, which indicates that a valid user program is present and can be started.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.435.2.7 FlashWrite()

```
blt_bool FlashWrite (
    blt_addr addr,
    blt_int32u len,
    blt_int8u * data )
```

Writes the data to flash through a flash block manager. Note that this function also checks that no data is programmed outside the flash memory region, so the bootloader can never be overwritten.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.
<i>data</i>	Pointer to the data buffer.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.435.2.8 FlashWriteChecksum()

```
blt_bool FlashWriteChecksum (
    void )
```

Writes a checksum of the user program to non-volatile memory. This is performed once the entire user program has been programmed. Through the checksum, the bootloader can check if the programming session was completed, which indicates that a valid user programming is present and can be started.

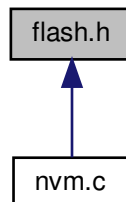
Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.436 flash.h File Reference

Bootloader flash driver header file.

This graph shows which files directly or indirectly include this file:



Functions

- void [FlashInit](#) (void)
Initializes the flash driver.
- void [FlashReinit](#) (void)
Reinitializes the flash driver.
- [blt_bool](#) [FlashWrite](#) ([blt_addr](#) addr, [blt_int32u](#) len, [blt_int8u](#) *data)
Writes the data to flash through a flash block manager. Note that this function also checks that no data is programmed outside the flash memory region, so the bootloader can never be overwritten.
- [blt_bool](#) [FlashErase](#) ([blt_addr](#) addr, [blt_int32u](#) len)
Erases the flash memory. Note that this function also checks that no data is erased outside the flash memory region, so the bootloader can never be erased.
- [blt_bool](#) [FlashWriteChecksum](#) (void)
Writes a checksum of the user program to non-volatile memory. This is performed once the entire user program has been programmed. Through the checksum, the bootloader can check if the programming session was completed, which indicates that a valid user programming is present and can be started.
- [blt_bool](#) [FlashVerifyChecksum](#) (void)
Verifies the checksum, which indicates that a valid user program is present and can be started.
- [blt_bool](#) [FlashDone](#) (void)
Finalizes the flash driver operations. There could still be data in the currently active block that needs to be flashed.
- [blt_addr](#) [FlashGetUserProgBaseAddress](#) (void)
Obtains the base address of the flash memory available to the user program. This is basically the first address in the flashLayout table.

7.436.1 Detailed Description

Bootloader flash driver header file.

7.436.2 Function Documentation

7.436.2.1 FlashDone()

```
blt_bool FlashDone (
    void )
```

Finalizes the flash driver operations. There could still be data in the currently active block that needs to be flashed.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.436.2.2 FlashErase()

```
blt_bool FlashErase (
    blt_addr addr,
    blt_int32u len )
```

Erases the flash memory. Note that this function also checks that no data is erased outside the flash memory region, so the bootloader can never be erased.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

Erases the flash memory. Note that this function also checks that no data is erased outside the flash memory region, so the bootloader can never be erased.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.436.2.3 FlashGetUserProgBaseAddress()

```
blt_addr FlashGetUserProgBaseAddress (
    void )
```

Obtains the base address of the flash memory available to the user program. This is basically the first address in the flashLayout table.

Returns

Base address.

Obtains the base address of the flash memory available to the user program. This is basically the first address in the flashLayout table.

Returns

Base address.

7.436.2.4 FlashInit()

```
void FlashInit (
    void )
```

Initializes the flash driver.

Returns

none.

7.436.2.5 FlashReinit()

```
void FlashReinit (
    void )
```

Reinitializes the flash driver.

Returns

none.

7.436.2.6 FlashVerifyChecksum()

```
blt_bool FlashVerifyChecksum (
    void )
```

Verifies the checksum, which indicates that a valid user program is present and can be started.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.436.2.7 FlashWrite()

```
blt_bool FlashWrite (
    blt_addr addr,
    blt_int32u len,
    blt_int8u * data )
```

Writes the data to flash through a flash block manager. Note that this function also checks that no data is programmed outside the flash memory region, so the bootloader can never be overwritten.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.
<i>data</i>	Pointer to the data buffer.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.436.2.8 FlashWriteChecksum()

```
blt_bool FlashWriteChecksum (  
    void )
```

Writes a checksum of the user program to non-volatile memory. This is performed once the entire user program has been programmed. Through the checksum, the bootloader can check if the programming session was completed, which indicates that a valid user programming is present and can be started.

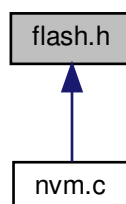
Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.437 flash.h File Reference

Bootloader flash driver header file.

This graph shows which files directly or indirectly include this file:



Functions

- void `FlashInit` (void)
Initializes the flash driver.
- void `FlashReinit` (void)
Reinitializes the flash driver.
- `blt_bool` `FlashWrite` (`blt_addr` addr, `blt_int32u` len, `blt_int8u` *data)
Writes the data to flash through a flash block manager. Note that this function also checks that no data is programmed outside the flash memory region, so the bootloader can never be overwritten.
- `blt_bool` `FlashErase` (`blt_addr` addr, `blt_int32u` len)
Erases the flash memory. Note that this function also checks that no data is erased outside the flash memory region, so the bootloader can never be erased.
- `blt_bool` `FlashWriteChecksum` (void)
Writes a checksum of the user program to non-volatile memory. This is performed once the entire user program has been programmed. Through the checksum, the bootloader can check if the programming session was completed, which indicates that a valid user programming is present and can be started.
- `blt_bool` `FlashVerifyChecksum` (void)
Verifies the checksum, which indicates that a valid user program is present and can be started.
- `blt_bool` `FlashDone` (void)
Finalizes the flash driver operations. There could still be data in the currently active block that needs to be flashed.
- `blt_addr` `FlashGetUserProgBaseAddress` (void)
Obtains the base address of the flash memory available to the user program. This is basically the first address in the flashLayout table.

7.437.1 Detailed Description

Bootloader flash driver header file.

7.437.2 Function Documentation

7.437.2.1 `FlashDone()`

```
blt_bool FlashDone (
    void )
```

Finalizes the flash driver operations. There could still be data in the currently active block that needs to be flashed.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.437.2.2 `FlashErase()`

```
blt_bool FlashErase (
    blt_addr addr,
    blt_int32u len )
```

Erases the flash memory. Note that this function also checks that no data is erased outside the flash memory region, so the bootloader can never be erased.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

Erases the flash memory. Note that this function also checks that no data is erased outside the flash memory region, so the bootloader can never be erased.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.437.2.3 FlashGetUserProgBaseAddress()

```
blt_addr FlashGetUserProgBaseAddress (
    void )
```

Obtains the base address of the flash memory available to the user program. This is basically the first address in the flashLayout table.

Returns

Base address.

Obtains the base address of the flash memory available to the user program. This is basically the first address in the flashLayout table.

Returns

Base address.

7.437.2.4 FlashInit()

```
void FlashInit (
    void )
```

Initializes the flash driver.

Returns

none.

7.437.2.5 FlashReinit()

```
void FlashReinit (
    void )
```

Reinitializes the flash driver.

Returns

none.

7.437.2.6 FlashVerifyChecksum()

```
blt_bool FlashVerifyChecksum (
    void )
```

Verifies the checksum, which indicates that a valid user program is present and can be started.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.437.2.7 FlashWrite()

```
blt_bool FlashWrite (
    blt_addr addr,
    blt_int32u len,
    blt_int8u * data )
```

Writes the data to flash through a flash block manager. Note that this function also checks that no data is programmed outside the flash memory region, so the bootloader can never be overwritten.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.
<i>data</i>	Pointer to the data buffer.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.437.2.8 FlashWriteChecksum()

```
blt_bool FlashWriteChecksum (
    void )
```

Writes a checksum of the user program to non-volatile memory. This is performed once the entire user program has been programmed. Through the checksum, the bootloader can check if the programming session was completed, which indicates that a valid user programming is present and can be started.

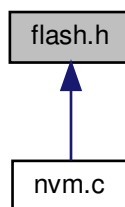
Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.438 flash.h File Reference

Bootloader flash driver header file.

This graph shows which files directly or indirectly include this file:



Functions

- void **FlashInit** (void)
Initializes the flash driver.
- void **FlashReinit** (void)
Reinitializes the flash driver.
- **blt_bool** **FlashWrite** (**blt_addr** addr, **blt_int32u** len, **blt_int8u** *data)
Writes the data to flash through a flash block manager. Note that this function also checks that no data is programmed outside the flash memory region, so the bootloader can never be overwritten.
- **blt_bool** **FlashErase** (**blt_addr** addr, **blt_int32u** len)
Erases the flash memory. Note that this function also checks that no data is erased outside the flash memory region, so the bootloader can never be erased.
- **blt_bool** **FlashWriteChecksum** (void)
Writes a checksum of the user program to non-volatile memory. This is performed once the entire user program has been programmed. Through the checksum, the bootloader can check if the programming session was completed, which indicates that a valid user programming is present and can be started.
- **blt_bool** **FlashVerifyChecksum** (void)
Verifies the checksum, which indicates that a valid user program is present and can be started.
- **blt_bool** **FlashDone** (void)
Finalizes the flash driver operations. There could still be data in the currently active block that needs to be flashed.
- **blt_addr** **FlashGetUserProgBaseAddress** (void)
Obtains the base address of the flash memory available to the user program. This is basically the first address in the flashLayout table.

7.438.1 Detailed Description

Bootloader flash driver header file.

7.438.2 Function Documentation

7.438.2.1 FlashDone()

```
blt_bool FlashDone (  
    void )
```

Finalizes the flash driver operations. There could still be data in the currently active block that needs to be flashed.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.438.2.2 FlashErase()

```
blt_bool FlashErase (  
    blt_addr addr,  
    blt_int32u len )
```

Erases the flash memory. Note that this function also checks that no data is erased outside the flash memory region, so the bootloader can never be erased.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

Erases the flash memory. Note that this function also checks that no data is erased outside the flash memory region, so the bootloader can never be erased.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.438.2.3 FlashGetUserProgBaseAddress()

```
blt_addr FlashGetUserProgBaseAddress (
    void )
```

Obtains the base address of the flash memory available to the user program. This is basically the first address in the flashLayout table.

Returns

Base address.

Obtains the base address of the flash memory available to the user program. This is basically the first address in the flashLayout table.

Returns

Base address.

7.438.2.4 FlashInit()

```
void FlashInit (
    void )
```

Initializes the flash driver.

Returns

none.

7.438.2.5 FlashReinit()

```
void FlashReinit (
    void )
```

Reinitializes the flash driver.

Returns

none.

7.438.2.6 FlashVerifyChecksum()

```
blt_bool FlashVerifyChecksum (
    void )
```

Verifies the checksum, which indicates that a valid user program is present and can be started.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.438.2.7 FlashWrite()

```
blt_bool FlashWrite (
    blt_addr addr,
    blt_int32u len,
    blt_int8u * data )
```

Writes the data to flash through a flash block manager. Note that this function also checks that no data is programmed outside the flash memory region, so the bootloader can never be overwritten.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.
<i>data</i>	Pointer to the data buffer.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.438.2.8 FlashWriteChecksum()

```
blt_bool FlashWriteChecksum (
    void )
```

Writes a checksum of the user program to non-volatile memory. This is performed once the entire user program has been programmed. Through the checksum, the bootloader can check if the programming session was completed, which indicates that a valid user programming is present and can be started.

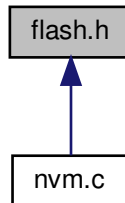
Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.439 flash.h File Reference

Bootloader flash driver header file.

This graph shows which files directly or indirectly include this file:



Functions

- void [FlashInit](#) (void)
Initializes the flash driver.
- void [FlashReinit](#) (void)
Reinitializes the flash driver.
- [blt_bool](#) [FlashWrite](#) ([blt_addr](#) addr, [blt_int32u](#) len, [blt_int8u](#) *data)
Writes the data to flash through a flash block manager. Note that this function also checks that no data is programmed outside the flash memory region, so the bootloader can never be overwritten.
- [blt_bool](#) [FlashErase](#) ([blt_addr](#) addr, [blt_int32u](#) len)
Erases the flash memory. Note that this function also checks that no data is erased outside the flash memory region, so the bootloader can never be erased.
- [blt_bool](#) [FlashWriteChecksum](#) (void)
Writes a checksum of the user program to non-volatile memory. This is performed once the entire user program has been programmed. Through the checksum, the bootloader can check if the programming session was completed, which indicates that a valid user programming is present and can be started.
- [blt_bool](#) [FlashVerifyChecksum](#) (void)
Verifies the checksum, which indicates that a valid user program is present and can be started.
- [blt_bool](#) [FlashDone](#) (void)
Finalizes the flash driver operations. There could still be data in the currently active block that needs to be flashed.
- [blt_addr](#) [FlashGetUserProgBaseAddress](#) (void)
Obtains the base address of the flash memory available to the user program. This is basically the first address in the flashLayout table.

7.439.1 Detailed Description

Bootloader flash driver header file.

7.439.2 Function Documentation

7.439.2.1 FlashDone()

```
blt_bool FlashDone (
    void )
```

Finalizes the flash driver operations. There could still be data in the currently active block that needs to be flashed.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.439.2.2 FlashErase()

```
blt_bool FlashErase (
    blt_addr addr,
    blt_int32u len )
```

Erases the flash memory. Note that this function also checks that no data is erased outside the flash memory region, so the bootloader can never be erased.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

Erases the flash memory. Note that this function also checks that no data is erased outside the flash memory region, so the bootloader can never be erased.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.439.2.3 FlashGetUserProgBaseAddress()

```
blt_addr FlashGetUserProgBaseAddress (
    void )
```

Obtains the base address of the flash memory available to the user program. This is basically the first address in the flashLayout table.

Returns

Base address.

Obtains the base address of the flash memory available to the user program. This is basically the first address in the flashLayout table.

Returns

Base address.

7.439.2.4 FlashInit()

```
void FlashInit (
    void )
```

Initializes the flash driver.

Returns

none.

7.439.2.5 FlashReinit()

```
void FlashReinit (
    void )
```

Reinitializes the flash driver.

Returns

none.

7.439.2.6 FlashVerifyChecksum()

```
blt_bool FlashVerifyChecksum (
    void )
```

Verifies the checksum, which indicates that a valid user program is present and can be started.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.439.2.7 FlashWrite()

```
blt_bool FlashWrite (
    blt_addr addr,
    blt_int32u len,
    blt_int8u * data )
```

Writes the data to flash through a flash block manager. Note that this function also checks that no data is programmed outside the flash memory region, so the bootloader can never be overwritten.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.
<i>data</i>	Pointer to the data buffer.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.439.2.8 FlashWriteChecksum()

```
blt_bool FlashWriteChecksum (  
    void )
```

Writes a checksum of the user program to non-volatile memory. This is performed once the entire user program has been programmed. Through the checksum, the bootloader can check if the programming session was completed, which indicates that a valid user programming is present and can be started.

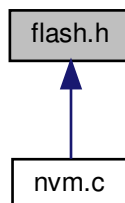
Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.440 flash.h File Reference

Bootloader flash driver header file.

This graph shows which files directly or indirectly include this file:



Functions

- void [FlashInit](#) (void)
Initializes the flash driver.
- void [FlashReinit](#) (void)
Reinitializes the flash driver.
- [blt_bool](#) [FlashWrite](#) ([blt_addr](#) addr, [blt_int32u](#) len, [blt_int8u](#) *data)
Writes the data to flash through a flash block manager. Note that this function also checks that no data is programmed outside the flash memory region, so the bootloader can never be overwritten.
- [blt_bool](#) [FlashErase](#) ([blt_addr](#) addr, [blt_int32u](#) len)
Erases the flash memory. Note that this function also checks that no data is erased outside the flash memory region, so the bootloader can never be erased.
- [blt_bool](#) [FlashWriteChecksum](#) (void)
Writes a checksum of the user program to non-volatile memory. This is performed once the entire user program has been programmed. Through the checksum, the bootloader can check if the programming session was completed, which indicates that a valid user programming is present and can be started.
- [blt_bool](#) [FlashVerifyChecksum](#) (void)
Verifies the checksum, which indicates that a valid user program is present and can be started.
- [blt_bool](#) [FlashDone](#) (void)
Finalizes the flash driver operations. There could still be data in the currently active block that needs to be flashed.
- [blt_addr](#) [FlashGetUserProgBaseAddress](#) (void)
Obtains the base address of the flash memory available to the user program. This is basically the first address in the flashLayout table.

7.440.1 Detailed Description

Bootloader flash driver header file.

7.440.2 Function Documentation

7.440.2.1 [FlashDone\(\)](#)

```
blt\_bool FlashDone (  
    void )
```

Finalizes the flash driver operations. There could still be data in the currently active block that needs to be flashed.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

Referenced by [NvmDone\(\)](#).

7.440.2.2 [FlashErase\(\)](#)

```
blt\_bool FlashErase (  
    blt\_addr addr,  
    blt\_int32u len )
```

Erases the flash memory. Note that this function also checks that no data is erased outside the flash memory region, so the bootloader can never be erased.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

Erases the flash memory. Note that this function also checks that no data is erased outside the flash memory region, so the bootloader can never be erased.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

Referenced by NvmErase().

7.440.2.3 FlashGetUserProgBaseAddress()

```
blt_addr FlashGetUserProgBaseAddress (
    void )
```

Obtains the base address of the flash memory available to the user program. This is basically the first address in the flashLayout table.

Returns

Base address.

Obtains the base address of the flash memory available to the user program. This is basically the first address in the flashLayout table.

Returns

Base address.

Referenced by NvmGetUserProgBaseAddress().

7.440.2.4 FlashInit()

```
void FlashInit (
    void )
```

Initializes the flash driver.

Returns

none.

Referenced by NvmInit().

7.440.2.5 FlashReinit()

```
void FlashReinit (
    void )
```

Reinitializes the flash driver.

Returns

none.

Referenced by NvmReinit().

7.440.2.6 FlashVerifyChecksum()

```
blt_bool FlashVerifyChecksum (
    void )
```

Verifies the checksum, which indicates that a valid user program is present and can be started.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

Referenced by NvmVerifyChecksum().

7.440.2.7 FlashWrite()

```
blt_bool FlashWrite (
    blt_addr addr,
    blt_int32u len,
    blt_int8u * data )
```

Writes the data to flash through a flash block manager. Note that this function also checks that no data is programmed outside the flash memory region, so the bootloader can never be overwritten.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.
<i>data</i>	Pointer to the data buffer.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

Referenced by FlashWriteChecksum(), and NvmWrite().

7.440.2.8 FlashWriteChecksum()

```
blt_bool FlashWriteChecksum (  
    void )
```

Writes a checksum of the user program to non-volatile memory. This is performed once the entire user program has been programmed. Through the checksum, the bootloader can check if the programming session was completed, which indicates that a valid user programming is present and can be started.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

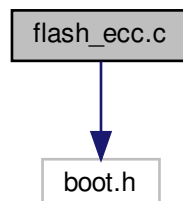
Referenced by NvmDone().

7.441 flash_ecc.c File Reference

Bootloader flash driver source file for HCS12 derivatives with error correction code in flash memory, such as the HCS12Pxx. This flash memory uses a different addressing scheme than other HCS12 derivatives.

```
#include "boot.h"
```

Include dependency graph for flash_ecc.c:



Data Structures

- struct [tFlashSector](#)
Flash sector descriptor type.
- struct [tFlashBlockInfo](#)
Structure type for grouping flash block information.
- struct [tFlashRegs](#)
Structure type for the flash control registers.
- struct [tFlashPrescalerSysclockMapping](#)
Mapping table for finding the corect flash clock divider prescaler.

Macros

- #define [FLASH_INVALID_SECTOR_IDX](#) (0xff)
Value for an invalid flash sector.
- #define [FLASH_INVALID_ADDRESS](#) (0xffffffff)
Value for an invalid flash address.
- #define [FLASH_WRITE_BLOCK_SIZE](#) (512)
Standard size of a flash block for writing.
- #define [FLASH_TOTAL_SECTORS](#) (sizeof(flashLayout)/sizeof(flashLayout[0]))
Total numbers of sectors in array flashLayout[].
- #define [BOOT_FLASH_VECTOR_TABLE_CS_OFFSET](#) (0x82)
Offset into the user program's vector table where the checksum is located. Note that the value can be overridden in blt_conf.h, because the size of the vector table could vary. When changing this value, don't forget to update the location of the checksum in the user program accordingly. Otherwise the checksum verification will always fail.
- #define [FLASH_VECTOR_TABLE_SIZE](#) (0x80)
Total size of the vector table, excluding the bootloader specific checksum.
- #define [FLASH_START_ADDRESS](#) (flashLayout[0].sector_start)
Start address of the bootloader programmable flash.
- #define [FLASH_END_ADDRESS](#)
End address of the bootloader programmable flash.
- #define [FLASH_PAGE_SIZE](#) (0x4000) /* flash page size in bytes */
Size of a flash page on the HCS12.
- #define [FLASH_PAGE_OFFSET](#) (0x8000) /* physical start addr. of pages */
Physical start address of the HCS12 page window.
- #define [FLASH_PPAGE_REG](#) (*(volatile blt_int8u *) (0x0015))
PPAGE register to select a specific flash page.
- #define [FLASH_REGS_BASE_ADDRESS](#) (0x0100)
Base address of the flash related control registers.
- #define [FLASH](#) ((volatile tFlashRegs *) FLASH_REGS_BASE_ADDRESS)
Macro for accessing the flash related control registers.
- #define [FLASH_FDIV_MASK](#) (0x3f)
Bitmask for flash clock divider bits.
- #define [FLASH_FDIV_INVALID](#) (0xff)
Invalid value for the flash clock divider bits.
- #define [FLASH_CMD_MAX_PARAMS](#) (4)
Maximum number of flash command parameters.
- #define [FLASH_PHRASE_SIZE](#) (8)
A phrase is an aligned group of 4 16-bit words, so 8 bytes.
- #define [FLASH_ERASE_SECTOR_CMD](#) (0x0A)
Erase sector flash command.

- #define **FLASH_PROGRAM_PHRASE_CMD** (0x06)
Program phrase flash command.
- #define **CCIF_BIT** (0x80)
FSTAT - command complete irg flag bit.
- #define **FDIVLD_BIT** (0x80)
FCLKDIV - clock divider loaded bit.
- #define **ACCERR_BIT** (0x20)
FSTAT - flash access error flag bit.
- #define **FPVIOL_BIT** (0x10)
FSTAT - flash protection violation flag bit.

Typedefs

- typedef void(* **pFlashExeCmdFct**) (void)
Pointer type to flash command execution function.

Functions

- static **blt_bool** **FlashInitBlock** (**tFlashBlockInfo** *block, **blt_addr** address)
Copies data currently in flash to the block->data and sets the base address.
- static **tFlashBlockInfo** * **FlashSwitchBlock** (**tFlashBlockInfo** *block, **blt_addr** base_addr)
Switches blocks by programming the current one and initializing the next.
- static **blt_bool** **FlashAddToBlock** (**tFlashBlockInfo** *block, **blt_addr** address, **blt_int8u** *data, **blt_int32u** len)
Programming is done per block. This function adds data to the block that is currently collecting data to be written to flash. If the address is outside of the current block, the current block is written to flash and a new block is initialized.
- static **blt_bool** **FlashWriteBlock** (**tFlashBlockInfo** *block)
Programs FLASH_WRITE_BLOCK_SIZE bytes to flash from the block->data array.
- static **blt_int8u** **FlashGetGlobalAddrByte** (**blt_addr** addr)
Reads the byte value from the linear address.
- static **blt_int8u** **FlashGetPhysPage** (**blt_addr** addr)
Extracts the physical flash page number from a linear address.
- static **blt_int16u** **FlashGetPhysAddr** (**blt_addr** addr)
Extracts the physical address on the flash page number from a linear address.
- static void **FlashExecuteCommand** (void)
Executes the command. The actual code for the command execution is stored as location independent machine code in array flashExecCmd[]. The contents of this array are temporarily copied to RAM. This way the function can be executed from RAM avoiding problem when try to perform a flash operation on the same flash block that this driver is located on.
- static **blt_bool** **FlashOperate** (**blt_int8u** cmd, **blt_addr** addr, **blt_int16u** params[], **blt_int8u** param_count)
Prepares the flash command and executes it.
- void **FlashInit** (void)
Initializes the flash driver.
- void **FlashReinit** (void)
Reinitializes the flash driver.
- **blt_bool** **FlashWrite** (**blt_addr** addr, **blt_int32u** len, **blt_int8u** *data)
Writes the data to flash through a flash block manager. Note that this function also checks that no data is programmed outside the flash memory region, so the bootloader can never be overwritten.
- **blt_bool** **FlashErase** (**blt_addr** addr, **blt_int32u** len)
Erases the flash memory. Note that this function also checks that no data is erased outside the flash memory region, so the bootloader can never be erased.

- [blt_bool FlashWriteChecksum](#) (void)
Writes a checksum of the user program to non-volatile memory. This is performed once the entire user program has been programmed. Through the checksum, the bootloader can check if the programming session was completed, which indicates that a valid user programming is present and can be started.
- [blt_bool FlashVerifyChecksum](#) (void)
Verifies the checksum, which indicates that a valid user program is present and can be started.
- [blt_bool FlashDone](#) (void)
Finalizes the flash driver operations. There could still be data in the currently active block that needs to be flashed.
- [blt_addr FlashGetUserProgBaseAddress](#) (void)
Obtains the base address of the flash memory available to the user program. This is basically the last address in the flashLayout table converted to the physical address on the last page, because this is where the address will be in.

Variables

- static const [tFlashSector flashLayout](#) []
If desired, it is possible to set `BOOT_FLASH_CUSTOM_LAYOUT_ENABLE` to `> 0` in `blt_conf.h` and then implement your own version of the `flashLayout[]` table in a source-file with the name `flash_layout.c`. This way you customize the flash memory size reserved for the bootloader, without having to modify the `flashLayout[]` table in this file directly. This file will then include `flash_layout.c` so there is no need to compile it additionally with your project.
- static const [tFlashPrescalerSysclockMapping flashFDIVlookup](#) []
Lookup table for determining the flash clock divider setting based on the system clock speed. The flash clock must be around 1MHz and is scaled down from the system clock using a prescaler value. Note that clock speeds in the table are in kHz.
- static const [blt_int8u flashExecCmd](#) []
Array with executable code for performing flash operations.
- static [tFlashBlockInfo blockInfo](#)
Local variable with information about the flash block that is currently being operated on.
- static [tFlashBlockInfo bootBlockInfo](#)
Local variable with information about the flash boot block.
- static [blt_int8u flashExecCmdRam](#) [(sizeof(flashExecCmd)/sizeof(flashExecCmd[0]))]
RAM buffer where the executable flash operation code is copied to.

7.441.1 Detailed Description

Bootloader flash driver source file for HCS12 derivatives with error correction code in flash memory, such as the HCS12Pxx. This flash memory uses a different addressing scheme than other HCS12 derivatives.

7.441.2 Function Documentation

7.441.2.1 FlashAddToBlock()

```
static blt_bool FlashAddToBlock (
    tFlashBlockInfo * block,
    blt_addr address,
    blt_int8u * data,
    blt_int32u len ) [static]
```

Programming is done per block. This function adds data to the block that is currently collecting data to be written to flash. If the address is outside of the current block, the current block is written to flash and a new block is initialized.

Parameters

<i>block</i>	Pointer to flash block info structure to operate on.
<i>address</i>	Flash destination address.
<i>data</i>	Pointer to the byte array with data.
<i>len</i>	Number of bytes to add to the block.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.441.2.2 FlashDone()

```
blt_bool FlashDone (  
    void )
```

Finalizes the flash driver operations. There could still be data in the currently active block that needs to be flashed.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.441.2.3 FlashErase()

```
blt_bool FlashErase (  
    blt_addr addr,  
    blt_int32u len )
```

Erases the flash memory. Note that this function also checks that no data is erased outside the flash memory region, so the bootloader can never be erased.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.441.2.4 FlashExecuteCommand()

```
static void FlashExecuteCommand (  
    void ) [static]
```

Executes the command. The actual code for the command execution is stored as location independant machine code in array flashExecCmd[]. The contents of this array are temporarily copied to RAM. This way the function can be executed from RAM avoiding problem when try to perform a flash operation on the same flash block that this driver is located on.

Returns

none.

Referenced by FlashOperate().

7.441.2.5 FlashGetGlobalAddrByte()

```
static blt_int8u FlashGetGlobalAddrByte (  
    blt_addr addr ) [static]
```

Reads the byte value from the linear address.

Parameters

<i>addr</i>	Linear address.
-------------	-----------------

Returns

The byte value located at the linear address.

Referenced by FlashWriteBlock().

7.441.2.6 FlashGetPhysAddr()

```
static blt_int16u FlashGetPhysAddr (  
    blt_addr addr ) [static]
```

Extracts the physical address on the flash page number from a linear address.

Parameters

<i>addr</i>	Linear address.
-------------	-----------------

Returns

The physical address.

Referenced by FlashGetGlobalAddrByte(), FlashGetUserProgBaseAddress(), and FlashInitBlock().

7.441.2.7 FlashGetPhysPage()

```
static blt_int8u FlashGetPhysPage (  
    blt_addr addr ) [static]
```

Extracts the physical flash page number from a linear address.

Parameters

<i>addr</i>	Linear address.
-------------	-----------------

Returns

The page number.

Referenced by FlashGetGlobalAddrByte(), and FlashInitBlock().

7.441.2.8 FlashGetUserProgBaseAddress()

```
blt_addr FlashGetUserProgBaseAddress (  
    void )
```

Obtains the base address of the flash memory available to the user program. This is basically the last address in the flashLayout table converted to the physical address on the last page, because this is where the address will be in.

Obtains the base address of the flash memory available to the user program. This is basically the first address in the flashLayout table.

Returns

Base address.

7.441.2.9 FlashInit()

```
void FlashInit (  
    void )
```

Initializes the flash driver.

Returns

none.

7.441.2.10 FlashInitBlock()

```
static blt_bool FlashInitBlock (
    tFlashBlockInfo * block,
    blt_addr address ) [static]
```

Copies data currently in flash to the block->data and sets the base address.

Parameters

<i>block</i>	Pointer to flash block info structure to operate on.
<i>address</i>	Base address of the block data.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

Referenced by FlashAddToBlock().

7.441.2.11 FlashOperate()

```
static blt_bool FlashOperate (
    blt_int8u cmd,
    blt_addr addr,
    blt_int16u params[ ],
    blt_int8u param_count ) [static]
```

Prepares the flash command and executes it.

Parameters

<i>cmd</i>	Command to be launched.
<i>addr</i>	Global address to operate on (18-bit).
<i>params</i>	Array with additional command parameters.
<i>param_count</i>	Number of parameters in the array.

Returns

BLT_TRUE if operation was successful, otherwise BLT_FALSE.

Referenced by FlashWriteBlock().

7.441.2.12 FlashReinit()

```
void FlashReinit (
    void )
```

Reinitializes the flash driver.

Returns

none.

7.441.2.13 FlashSwitchBlock()

```
static tFlashBlockInfo * FlashSwitchBlock (
    tFlashBlockInfo * block,
    blt_addr base_addr ) [static]
```

Switches blocks by programming the current one and initializing the next.

Parameters

<i>block</i>	Pointer to flash block info structure to operate on.
<i>base_addr</i>	Base address of the next block.

Returns

The pointer of the block info struct that is no being used, or a NULL pointer in case of error.

Referenced by FlashAddToBlock().

7.441.2.14 FlashVerifyChecksum()

```
blt_bool FlashVerifyChecksum (
    void )
```

Verifies the checksum, which indicates that a valid user program is present and can be started.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.441.2.15 FlashWrite()

```
blt_bool FlashWrite (
    blt_addr addr,
    blt_int32u len,
    blt_int8u * data )
```

Writes the data to flash through a flash block manager. Note that this function also checks that no data is programmed outside the flash memory region, so the bootloader can never be overwritten.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.
<i>data</i>	Pointer to the data buffer.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.441.2.16 FlashWriteBlock()

```
static blt_bool FlashWriteBlock (  
    tFlashBlockInfo * block ) [static]
```

Programs FLASH_WRITE_BLOCK_SIZE bytes to flash from the block->data array.

Parameters

<i>block</i>	Pointer to flash block info structure to operate on.
--------------	--

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

Referenced by FlashDone().

7.441.2.17 FlashWriteChecksum()

```
blt_bool FlashWriteChecksum (  
    void )
```

Writes a checksum of the user program to non-volatile memory. This is performed once the entire user program has been programmed. Through the checksum, the bootloader can check if the programming session was completed, which indicates that a valid user programming is present and can be started.

Returns

BLT_TRUE if successful, BLT_FALSE otherwise.

7.441.3 Variable Documentation

7.441.3.1 blockInfo

```
tFlashBlockInfo blockInfo [static]
```

Local variable with information about the flash block that is currently being operated on.

The smallest amount of flash that can be programmed is FLASH_WRITE_BLOCK_SIZE. A flash block manager is implemented in this driver and stores info in this variable. Whenever new data should be flashed, it is first added to a RAM buffer, which is part of this variable. Whenever the RAM buffer, which has the size of a flash block, is full or data needs to be written to a different block, the contents of the RAM buffer are programmed to flash. The flash block manager requires some software overhead, yet results in faster flash programming because data is first harvested, ideally until there is enough to program an entire flash block, before the flash device is actually operated on.

7.441.3.2 bootBlockInfo

```
tFlashBlockInfo bootBlockInfo [static]
```

Local variable with information about the flash boot block.

The first block of the user program holds the vector table, which on the STM32 is also the where the checksum is written to. Is it likely that the vector table is first flashed and then, at the end of the programming sequence, the checksum. This means that this flash block need to be written to twice. Normally this is not a problem with flash memory, as long as you write the same values to those bytes that are not supposed to be changed and the locations where you do write to are still in the erased 0xFF state. Unfortunately, writing twice to flash this way, does not work reliably on all micros. This is why we need to have an extra block, the bootblock, placed under the management of the block manager. This way is it possible to implement functionality so that the bootblock is only written to once at the end of the programming sequence.

7.441.3.3 flashExecCmd

```
const blt_int8u flashExecCmd[] [static]
```

Initial value:

```
=
{
    0x36,
    0x34,
    0xce, 0x01, 0x00,
    0x1a, 0x06,
    0x86, 0x80,
    0x6a, 0x00,
    0xa7, 0xa7, 0xa7, 0xa7,
    0x0f, 0x00, 0x80, 0xfc,
    0x30,
    0x32,
    0x3d
}
```

Array with executable code for performing flash operations.

This array contains the machine code to perform the actual command on the flash device, such as program or erase. the code is compiler and location independent. This allows us to copy it to a ram buffer and execute the code from ram. This way the flash driver can be located in flash memory without running into problems when erasing/programming the same flash block that contains the flash driver. the source code for the machine code is as follows: // launch the command FLASH->fstat = CCIF_BIT; // wait at least 4 cycles (per AN2720) asm("nop"); asm("nop"); asm("nop"); // wait for command to complete while ((FLASH->fstat & CCIF_BIT) != CCI↵F_BIT);

Referenced by FlashDone(), and FlashExecuteCommand().

7.441.3.4 flashLayout

```
const tFlashSector flashLayout[] [static]
```

If desired, it is possible to set BOOT_FLASH_CUSTOM_LAYOUT_ENABLE to > 0 in blt_conf.h and then implement your own version of the flashLayout[] table in a source-file with the name flash_layout.c. This way you customize the flash memory size reserved for the bootloader, without having to modify the flashLayout[] table in this file directly. This file will then include flash_layout.c so there is no need to compile it additionally with your project.

Array wit the layout of the flash memory.

Also controls what part of the flash memory is reserved for the bootloader. If the bootloader size changes, the reserved sectors for the bootloader might need adjustment to make sure the bootloader doesn't get overwritten. This layout uses global addresses only. Note that the last part is where the bootloader also resides and it has been entered as 8 chunks of 2kb. This allows flexibility for reserving more/less space for the bootloader in case its size changes in the future.

7.442 flash_layout.c File Reference

Custom flash layout table source file.

Variables

- static const tFlashSector flashLayout []
Array wit the layout of the flash memory.

7.442.1 Detailed Description

Custom flash layout table source file.

7.442.2 Variable Documentation

7.442.2.1 flashLayout

```
const tFlashSector flashLayout[] [static]
```

Initial value:

```
=
{

    { 0x08002000, 0x00800, 4},
    { 0x08002800, 0x00800, 5},
    { 0x08003000, 0x00800, 6},
    { 0x08003800, 0x00800, 7},
    { 0x08004000, 0x00800, 8},
    { 0x08004800, 0x00800, 9},
    { 0x08005000, 0x00800, 10},
    { 0x08005800, 0x00800, 11},
    { 0x08006000, 0x00800, 12},
    { 0x08006800, 0x00800, 13},
    { 0x08007000, 0x00800, 14},
    { 0x08007800, 0x00800, 15},
    { 0x08008000, 0x00800, 16},
}
```

Array wit the layout of the flash memory.

Also controls what part of the flash memory is reserved for the bootloader. If the bootloader size changes, the reserved sectors for the bootloader might need adjustment to make sure the bootloader doesn't get overwritten.

Referenced by FlashEraseSectors(), FlashGetSectorIdx(), FlashGetUserProgBaseAddress(), FlashSwitchBlock(), FlashVerifyChecksum(), FlashWrite(), and FlashWriteChecksum().

7.443 flash_layout.c File Reference

Custom flash layout table source file.

Variables

- static const tFlashSector flashLayout []
Array wit the layout of the flash memory.

7.443.1 Detailed Description

Custom flash layout table source file.

7.443.2 Variable Documentation

7.443.2.1 flashLayout

```
const tFlashSector flashLayout[] [static]
```

Initial value:

```
=  
{  
  
    { 0x08002000, 0x00800, 4},  
    { 0x08002800, 0x00800, 5},  
    { 0x08003000, 0x00800, 6},  
    { 0x08003800, 0x00800, 7},  
    { 0x08004000, 0x00800, 8},  
    { 0x08004800, 0x00800, 9},  
    { 0x08005000, 0x00800, 10},  
    { 0x08005800, 0x00800, 11},  
    { 0x08006000, 0x00800, 12},  
    { 0x08006800, 0x00800, 13},  
    { 0x08007000, 0x00800, 14},  
    { 0x08007800, 0x00800, 15},  
    { 0x08008000, 0x00800, 16},  
}
```

Array with the layout of the flash memory.

Also controls what part of the flash memory is reserved for the bootloader. If the bootloader size changes, the reserved sectors for the bootloader might need adjustment to make sure the bootloader doesn't get overwritten.

7.444 flash_layout.c File Reference

Custom flash layout table source file.

Variables

- static const tFlashSector flashLayout []
Array with the layout of the flash memory.

7.444.1 Detailed Description

Custom flash layout table source file.

7.444.2 Variable Documentation

7.444.2.1 flashLayout

```
const tFlashSector flashLayout[] [static]
```

Initial value:

```
=
{

    { 0x08002000, 0x00800, 4},
    { 0x08002800, 0x00800, 5},
    { 0x08003000, 0x00800, 6},
    { 0x08003800, 0x00800, 7},
    { 0x08004000, 0x00800, 8},
    { 0x08004800, 0x00800, 9},
    { 0x08005000, 0x00800, 10},
    { 0x08005800, 0x00800, 11},
    { 0x08006000, 0x00800, 12},
    { 0x08006800, 0x00800, 13},
    { 0x08007000, 0x00800, 14},
    { 0x08007800, 0x00800, 15},
    { 0x08008000, 0x08000, 16},
}
```

Array wit the layout of the flash memory.

Also controls what part of the flash memory is reserved for the bootloader. If the bootloader size changes, the reserved sectors for the bootloader might need adjustment to make sure the bootloader doesn't get overwritten.

7.445 flash_layout.c File Reference

Custom flash layout table source file.

Variables

- static const tFlashSector flashLayout []
Array wit the layout of the flash memory.

7.445.1 Detailed Description

Custom flash layout table source file.

7.445.2 Variable Documentation

7.445.2.1 flashLayout

```
const tFlashSector flashLayout[] [static]
```

Initial value:

```
=
{

    { 0x08002000, 0x00800, 4},
    { 0x08002800, 0x00800, 5},
    { 0x08003000, 0x00800, 6},
    { 0x08003800, 0x00800, 7},
    { 0x08004000, 0x00800, 8},
    { 0x08004800, 0x00800, 9},
    { 0x08005000, 0x00800, 10},
    { 0x08005800, 0x00800, 11},
    { 0x08006000, 0x00800, 12},
    { 0x08006800, 0x00800, 13},
    { 0x08007000, 0x00800, 14},
    { 0x08007800, 0x00800, 15},
    { 0x08008000, 0x08000, 16},
}
```

Array wit the layout of the flash memory.

Also controls what part of the flash memory is reserved for the bootloader. If the bootloader size changes, the reserved sectors for the bootloader might need adjustment to make sure the bootloader doesn't get overwritten.

7.446 flash_layout.c File Reference

Custom flash layout table source file.

Variables

- static const tFlashSector flashLayout []
Array wit the layout of the flash memory.

7.446.1 Detailed Description

Custom flash layout table source file.

7.446.2 Variable Documentation

7.446.2.1 flashLayout

```
const tFlashSector flashLayout[] [static]
```

Initial value:

```
=
{
    { 0x00004000, 0x02000, 2},
    { 0x00006000, 0x02000, 3},
    { 0x00008000, 0x02000, 4},
    { 0x0000A000, 0x02000, 5},
    { 0x0000C000, 0x02000, 6},
    { 0x0000E000, 0x02000, 7},
    { 0x00010000, 0x02000, 8},
    { 0x00012000, 0x02000, 9},
    { 0x00014000, 0x02000, 10},
    { 0x00016000, 0x02000, 11},
    { 0x00018000, 0x02000, 12},
    { 0x0001A000, 0x02000, 13},
    { 0x0001C000, 0x02000, 14},
    { 0x0001E000, 0x02000, 15},
    { 0x00020000, 0x08000, 16},
    { 0x00028000, 0x08000, 17},
    { 0x00030000, 0x08000, 18},
    { 0x00038000, 0x08000, 19},
}
```

Array with the layout of the flash memory.

Also controls what part of the flash memory is reserved for the bootloader. If the bootloader size changes, the reserved sectors for the bootloader might need adjustment to make sure the bootloader doesn't get overwritten.

7.447 flash_layout.c File Reference

Custom flash layout table source file.

Variables

- static const tFlashSector flashLayout []
Array with the layout of the flash memory.

7.447.1 Detailed Description

Custom flash layout table source file.

7.447.2 Variable Documentation

7.447.2.1 flashLayout

```
const tFlashSector flashLayout[] [static]
```

Initial value:

```
=
{
    { 0x00004000, 0x02000, 2},
    { 0x00006000, 0x02000, 3},
    { 0x00008000, 0x02000, 4},
    { 0x0000A000, 0x02000, 5},
    { 0x0000C000, 0x02000, 6},
    { 0x0000E000, 0x02000, 7},
    { 0x00010000, 0x02000, 8},
    { 0x00012000, 0x02000, 9},
    { 0x00014000, 0x02000, 10},
    { 0x00016000, 0x02000, 11},
    { 0x00018000, 0x02000, 12},
    { 0x0001A000, 0x02000, 13},
    { 0x0001C000, 0x02000, 14},
    { 0x0001E000, 0x02000, 15},
    { 0x00020000, 0x08000, 16},
    { 0x00028000, 0x08000, 17},
    { 0x00030000, 0x08000, 18},
    { 0x00038000, 0x08000, 19},
}
```

Array with the layout of the flash memory.

Also controls what part of the flash memory is reserved for the bootloader. If the bootloader size changes, the reserved sectors for the bootloader might need adjustment to make sure the bootloader doesn't get overwritten.

7.448 flash_layout.c File Reference

Custom flash layout table source file.

Variables

- static const tFlashSector flashLayout []
Array with the layout of the flash memory.

7.448.1 Detailed Description

Custom flash layout table source file.

7.448.2 Variable Documentation

7.448.2.1 flashLayout

```
const tFlashSector flashLayout[] [static]
```

Initial value:

```
=
{
    { 0x08002000, 0x02000 },
    { 0x08004000, 0x02000 },
    { 0x08006000, 0x02000 },
    { 0x08008000, 0x02000 },
    { 0x0800A000, 0x02000 },
    { 0x0800C000, 0x02000 },
    { 0x0800E000, 0x02000 },
    { 0x08010000, 0x02000 },
    { 0x08012000, 0x02000 },
    { 0x08014000, 0x02000 },
    { 0x08016000, 0x02000 },
    { 0x08018000, 0x02000 },
    { 0x0801A000, 0x02000 },
    { 0x0801C000, 0x02000 },
    { 0x0801E000, 0x02000 },
}
```

Array wit the layout of the flash memory.

Also controls what part of the flash memory is reserved for the bootloader. If the bootloader size changes, the reserved sectors for the bootloader might need adjustment to make sure the bootloader doesn't get overwritten.

7.449 flash_layout.c File Reference

Custom flash layout table source file.

Variables

- static const `tFlashSector flashLayout[]`
Array wit the layout of the flash memory.

7.449.1 Detailed Description

Custom flash layout table source file.

7.449.2 Variable Documentation

7.449.2.1 flashLayout

```
const tFlashSector flashLayout[] [static]
```

Initial value:

```
=
{
    { 0x08002000, 0x02000 },
    { 0x08004000, 0x02000 },
    { 0x08006000, 0x02000 },
    { 0x08008000, 0x02000 },
    { 0x0800A000, 0x02000 },
    { 0x0800C000, 0x02000 },
    { 0x0800E000, 0x02000 },
    { 0x08010000, 0x02000 },
    { 0x08012000, 0x02000 },
    { 0x08014000, 0x02000 },
    { 0x08016000, 0x02000 },
    { 0x08018000, 0x02000 },
    { 0x0801A000, 0x02000 },
    { 0x0801C000, 0x02000 },
    { 0x0801E000, 0x02000 },
}
```

Array with the layout of the flash memory.

Also controls what part of the flash memory is reserved for the bootloader. If the bootloader size changes, the reserved sectors for the bootloader might need adjustment to make sure the bootloader doesn't get overwritten.

7.450 flash_layout.c File Reference

Custom flash layout table source file.

Variables

- static const tFlashSector flashLayout []
Array with the layout of the flash memory.

7.450.1 Detailed Description

Custom flash layout table source file.

7.450.2 Variable Documentation

7.450.2.1 flashLayout

```
const tFlashSector flashLayout[] [static]
```

Initial value:

```
=
{
    { 0x08002000, 0x02000 },
    { 0x08004000, 0x02000 },
    { 0x08006000, 0x02000 },
    { 0x08008000, 0x02000 },
    { 0x0800A000, 0x02000 },
    { 0x0800C000, 0x02000 },
    { 0x0800E000, 0x02000 },
    { 0x08010000, 0x02000 },
    { 0x08012000, 0x02000 },
    { 0x08014000, 0x02000 },
    { 0x08016000, 0x02000 },
    { 0x08018000, 0x02000 },
    { 0x0801A000, 0x02000 },
    { 0x0801C000, 0x02000 },
    { 0x0801E000, 0x02000 },
}
```

Array with the layout of the flash memory.

Also controls what part of the flash memory is reserved for the bootloader. If the bootloader size changes, the reserved sectors for the bootloader might need adjustment to make sure the bootloader doesn't get overwritten.

7.451 flash_layout.c File Reference

Custom flash layout table source file.

Variables

- static const `tFlashSector flashLayout[]`
Array with the layout of the flash memory.

7.451.1 Detailed Description

Custom flash layout table source file.

7.451.2 Variable Documentation

7.451.2.1 flashLayout

```
const tFlashSector flashLayout[] [static]
```

Initial value:

```
=
{
    { 0x08002000, 0x02000 },
    { 0x08004000, 0x02000 },
    { 0x08006000, 0x02000 },
    { 0x08008000, 0x02000 },
    { 0x0800A000, 0x02000 },
    { 0x0800C000, 0x02000 },
    { 0x0800E000, 0x02000 },
    { 0x08010000, 0x02000 },
    { 0x08012000, 0x02000 },
    { 0x08014000, 0x02000 },
    { 0x08016000, 0x02000 },
    { 0x08018000, 0x02000 },
    { 0x0801A000, 0x02000 },
    { 0x0801C000, 0x02000 },
    { 0x0801E000, 0x02000 },
}
```

Array with the layout of the flash memory.

Also controls what part of the flash memory is reserved for the bootloader. If the bootloader size changes, the reserved sectors for the bootloader might need adjustment to make sure the bootloader doesn't get overwritten.

7.452 flash_layout.c File Reference

Custom flash layout table source file.

Variables

- static const tFlashSector flashLayout []
Array with the layout of the flash memory.

7.452.1 Detailed Description

Custom flash layout table source file.

7.452.2 Variable Documentation

7.452.2.1 flashLayout

```
const tFlashSector flashLayout[] [static]
```

Initial value:

```
=
{

    { 0x08004000, 0x02000 },
    { 0x08006000, 0x02000 },
    { 0x08008000, 0x02000 },
    { 0x0800A000, 0x02000 },
    { 0x0800C000, 0x02000 },
    { 0x0800E000, 0x02000 },
    { 0x08010000, 0x02000 },
    { 0x08012000, 0x02000 },
    { 0x08014000, 0x02000 },
    { 0x08016000, 0x02000 },
    { 0x08018000, 0x02000 },
    { 0x0801A000, 0x02000 },
    { 0x0801C000, 0x02000 },
    { 0x0801E000, 0x02000 },
}
```

Array wit the layout of the flash memory.

Also controls what part of the flash memory is reserved for the bootloader. If the bootloader size changes, the reserved sectors for the bootloader might need adjustment to make sure the bootloader doesn't get overwritten.

7.453 flash_layout.c File Reference

Custom flash layout table source file.

Variables

- static const tFlashSector flashLayout []
Array wit the layout of the flash memory.

7.453.1 Detailed Description

Custom flash layout table source file.

7.453.2 Variable Documentation

7.453.2.1 flashLayout

```
const tFlashSector flashLayout[] [static]
```

Initial value:

```
=
{

    { 0x08004000, 0x02000 },
    { 0x08006000, 0x02000 },
    { 0x08008000, 0x02000 },
    { 0x0800A000, 0x02000 },
    { 0x0800C000, 0x02000 },
    { 0x0800E000, 0x02000 },
    { 0x08010000, 0x02000 },
    { 0x08012000, 0x02000 },
    { 0x08014000, 0x02000 },
    { 0x08016000, 0x02000 },
    { 0x08018000, 0x02000 },
    { 0x0801A000, 0x02000 },
    { 0x0801C000, 0x02000 },
    { 0x0801E000, 0x02000 },
}
```

Array wit the layout of the flash memory.

Also controls what part of the flash memory is reserved for the bootloader. If the bootloader size changes, the reserved sectors for the bootloader might need adjustment to make sure the bootloader doesn't get overwritten.

7.454 flash_layout.c File Reference

Custom flash layout table source file.

Variables

- static const tFlashSector flashLayout []
Array wit the layout of the flash memory.

7.454.1 Detailed Description

Custom flash layout table source file.

7.454.2 Variable Documentation

7.454.2.1 flashLayout

```
const tFlashSector flashLayout[] [static]
```

Initial value:

```
=
{

    { 0x08004000, 0x02000 },
    { 0x08006000, 0x02000 },
    { 0x08008000, 0x02000 },
    { 0x0800A000, 0x02000 },
    { 0x0800C000, 0x02000 },
    { 0x0800E000, 0x02000 },
    { 0x08010000, 0x02000 },
    { 0x08012000, 0x02000 },
    { 0x08014000, 0x02000 },
    { 0x08016000, 0x02000 },
    { 0x08018000, 0x02000 },
    { 0x0801A000, 0x02000 },
    { 0x0801C000, 0x02000 },
    { 0x0801E000, 0x02000 },
}
```

Array with the layout of the flash memory.

Also controls what part of the flash memory is reserved for the bootloader. If the bootloader size changes, the reserved sectors for the bootloader might need adjustment to make sure the bootloader doesn't get overwritten.

7.455 flash_layout.c File Reference

Custom flash layout table source file.

Variables

- static const `tFlashSector flashLayout[]`
Array with the layout of the flash memory.

7.455.1 Detailed Description

Custom flash layout table source file.

7.455.2 Variable Documentation

7.455.2.1 flashLayout

```
const tFlashSector flashLayout[] [static]
```

Initial value:

```
=
{

    { 0x08004000, 0x02000 },
    { 0x08006000, 0x02000 },
    { 0x08008000, 0x02000 },
    { 0x0800A000, 0x02000 },
    { 0x0800C000, 0x02000 },
    { 0x0800E000, 0x02000 },
    { 0x08010000, 0x02000 },
    { 0x08012000, 0x02000 },
    { 0x08014000, 0x02000 },
    { 0x08016000, 0x02000 },
    { 0x08018000, 0x02000 },
    { 0x0801A000, 0x02000 },
    { 0x0801C000, 0x02000 },
    { 0x0801E000, 0x02000 },
}
```

Array wit the layout of the flash memory.

Also controls what part of the flash memory is reserved for the bootloader. If the bootloader size changes, the reserved sectors for the bootloader might need adjustment to make sure the bootloader doesn't get overwritten.

7.456 flash_layout.c File Reference

Custom flash layout table source file.

Variables

- static const `tFlashSector flashLayout []`
Array wit the layout of the flash memory.

7.456.1 Detailed Description

Custom flash layout table source file.

7.456.2 Variable Documentation

7.456.2.1 flashLayout

```
const tFlashSector flashLayout[] [static]
```

Initial value:

```
=
{

    { 0x08008000, 0x02000 },
    { 0x0800A000, 0x02000 },
    { 0x0800C000, 0x02000 },
    { 0x0800E000, 0x02000 },
    { 0x08010000, 0x02000 },
    { 0x08012000, 0x02000 },
    { 0x08014000, 0x02000 },
    { 0x08016000, 0x02000 },
    { 0x08018000, 0x02000 },
    { 0x0801A000, 0x02000 },
    { 0x0801C000, 0x02000 },
    { 0x0801E000, 0x02000 },
}
```

Array wit the layout of the flash memory.

Also controls what part of the flash memory is reserved for the bootloader. If the bootloader size changes, the reserved sectors for the bootloader might need adjustment to make sure the bootloader doesn't get overwritten.

7.457 flash_layout.c File Reference

Custom flash layout table source file.

Variables

- static const `tFlashSector flashLayout []`
Array wit the layout of the flash memory.

7.457.1 Detailed Description

Custom flash layout table source file.

7.457.2 Variable Documentation

7.457.2.1 flashLayout

```
const tFlashSector flashLayout[] [static]
```

Initial value:

```
=
{

    { 0x08008000, 0x02000 },
    { 0x0800A000, 0x02000 },
    { 0x0800C000, 0x02000 },
    { 0x0800E000, 0x02000 },
    { 0x08010000, 0x02000 },
    { 0x08012000, 0x02000 },
    { 0x08014000, 0x02000 },
    { 0x08016000, 0x02000 },
    { 0x08018000, 0x02000 },
    { 0x0801A000, 0x02000 },
    { 0x0801C000, 0x02000 },
    { 0x0801E000, 0x02000 },
}
```

Array wit the layout of the flash memory.

Also controls what part of the flash memory is reserved for the bootloader. If the bootloader size changes, the reserved sectors for the bootloader might need adjustment to make sure the bootloader doesn't get overwritten.

7.458 flash_layout.c File Reference

Custom flash layout table source file.

Variables

- static const tFlashSector flashLayout []
Array wit the layout of the flash memory.

7.458.1 Detailed Description

Custom flash layout table source file.

7.458.2 Variable Documentation

7.458.2.1 flashLayout

```
const tFlashSector flashLayout[] [static]
```

Initial value:

```
=
{

    { 0x08008000, 0x02000 },
    { 0x0800A000, 0x02000 },
    { 0x0800C000, 0x02000 },
    { 0x0800E000, 0x02000 },
    { 0x08010000, 0x02000 },
    { 0x08012000, 0x02000 },
    { 0x08014000, 0x02000 },
    { 0x08016000, 0x02000 },
    { 0x08018000, 0x02000 },
    { 0x0801A000, 0x02000 },
    { 0x0801C000, 0x02000 },
    { 0x0801E000, 0x02000 },
}
```

Array wit the layout of the flash memory.

Also controls what part of the flash memory is reserved for the bootloader. If the bootloader size changes, the reserved sectors for the bootloader might need adjustment to make sure the bootloader doesn't get overwritten.

7.459 flash_layout.c File Reference

Custom flash layout table source file.

Variables

- static const tFlashSector flashLayout []
Array wit the layout of the flash memory.

7.459.1 Detailed Description

Custom flash layout table source file.

7.459.2 Variable Documentation

7.459.2.1 flashLayout

```
const tFlashSector flashLayout[] [static]
```

Initial value:

```
=  
{
```

```
    { 0x08008000, 0x02000 },  
    { 0x0800A000, 0x02000 },  
    { 0x0800C000, 0x02000 },  
    { 0x0800E000, 0x02000 },  
    { 0x08010000, 0x02000 },  
    { 0x08012000, 0x02000 },  
    { 0x08014000, 0x02000 },  
    { 0x08016000, 0x02000 },  
    { 0x08018000, 0x02000 },  
    { 0x0801A000, 0x02000 },  
    { 0x0801C000, 0x02000 },  
    { 0x0801E000, 0x02000 },  
}
```

Array wit the layout of the flash memory.

Also controls what part of the flash memory is reserved for the bootloader. If the bootloader size changes, the reserved sectors for the bootloader might need adjustment to make sure the bootloader doesn't get overwritten.

7.460 flash_layout.c File Reference

Custom flash layout table source file.

Variables

- static const tFlashSector flashLayout []
Array wit the layout of the flash memory.

7.460.1 Detailed Description

Custom flash layout table source file.

7.460.2 Variable Documentation


```
const tFlashSector flashLayout[] [static]
```

$$= \{$$

```
{ 0x08004000, 0x008000 },
{ 0x080004800, 0x008000 },
{ 0x080005000, 0x008000 },
{ 0x080005800, 0x008000 },
{ 0x080006000, 0x008000 },
{ 0x080006800, 0x008000 },
{ 0x080007000, 0x008000 },
{ 0x080007800, 0x008000 },
{ 0x080008000, 0x080000 },
{ 0x08010000, 0x10000 },
{ 0x08020000, 0x20000 },
}
```

Also controls what part of the flash memory is reserved for the bootloader. If the bootloader size changes, the reserved sectors for the bootloader might need adjustment to make sure the bootloader doesn't get overwritten.

Custom flash layout table source file.

- static const `tFlashSector` `flashLayout` []
Array wit the layout of the flash memory.

Custom flash layout table source file.

Generated by Doxygen

7.463.2.1 flashLayout

```
const tFlashSector flashLayout[] [static]
```

Initial value:

```
=  
{
```

```
  
    { 0x08004000, 0x00800 },  
    { 0x08004800, 0x00800 },  
    { 0x08005000, 0x00800 },  
    { 0x08005800, 0x00800 },  
    { 0x08006000, 0x00800 },  
    { 0x08006800, 0x00800 },  
    { 0x08007000, 0x00800 },  
    { 0x08007800, 0x00800 },  
    { 0x08008000, 0x00800 },  
    { 0x08010000, 0x10000 },  
    { 0x08020000, 0x20000 },  
}
```

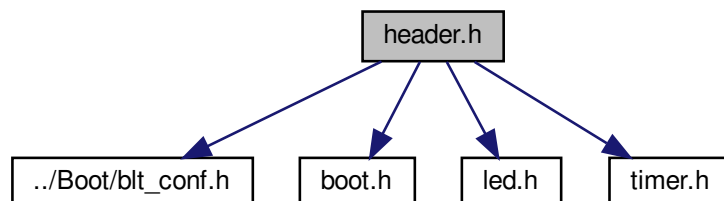
Array with the layout of the flash memory.

Also controls what part of the flash memory is reserved for the bootloader. If the bootloader size changes, the reserved sectors for the bootloader might need adjustment to make sure the bootloader doesn't get overwritten.

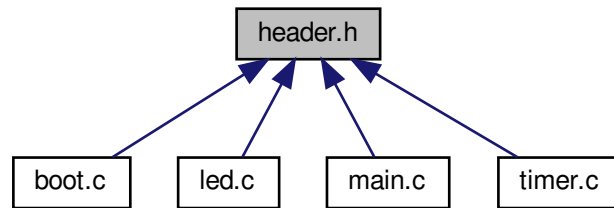
7.464 header.h File Reference

Generic header file.

```
#include "../Boot/blt_conf.h"  
#include "boot.h"  
#include "led.h"  
#include "timer.h"  
Include dependency graph for _template/Prog/header.h:
```



This graph shows which files directly or indirectly include this file:



7.464.1 Detailed Description

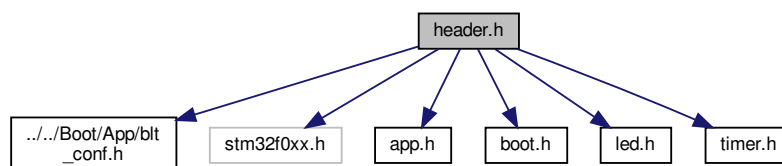
Generic header file.

7.465 header.h File Reference

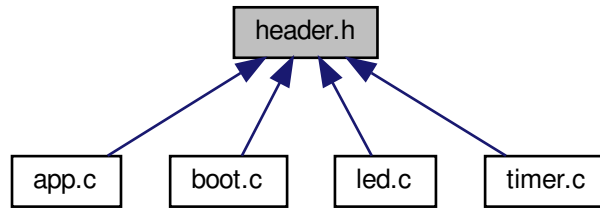
Generic header file.

```
#include "../..../Boot/App/blt_conf.h"
#include "stm32f0xx.h"
#include "app.h"
#include "boot.h"
#include "led.h"
#include "timer.h"
```

Include dependency graph for ARMCM0_STM32F0_Discovery_STM32F051_CubeIDE/Prog/App/header.h:



This graph shows which files directly or indirectly include this file:



7.465.1 Detailed Description

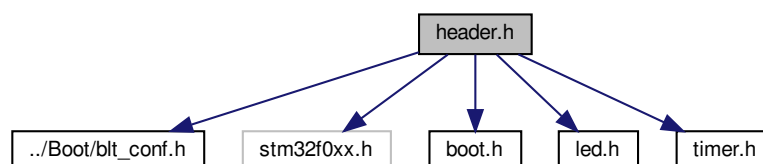
Generic header file.

7.466 header.h File Reference

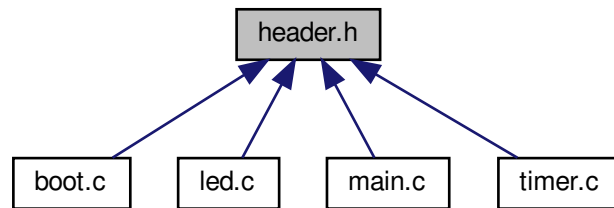
Generic header file.

```
#include "../Boot/blt_conf.h"
#include "stm32f0xx.h"
#include "boot.h"
#include "led.h"
#include "timer.h"
```

Include dependency graph for ARMC00_STM32F0_Discovery_STM32F051_GCC/Prog/header.h:



This graph shows which files directly or indirectly include this file:



7.466.1 Detailed Description

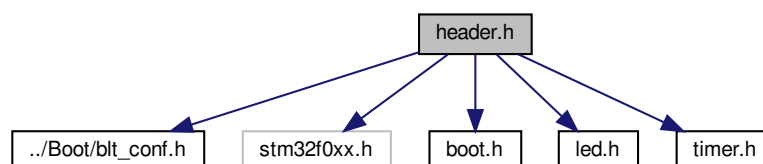
Generic header file.

7.467 header.h File Reference

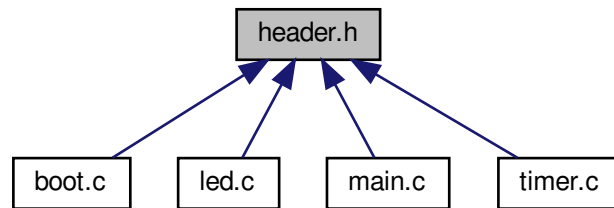
Generic header file.

```
#include "../Boot/blt_conf.h"
#include "stm32f0xx.h"
#include "boot.h"
#include "led.h"
#include "timer.h"
```

Include dependency graph for `ARMCM0_STM32F0_Discovery_STM32F051_IAR/Prog/header.h`:



This graph shows which files directly or indirectly include this file:



7.467.1 Detailed Description

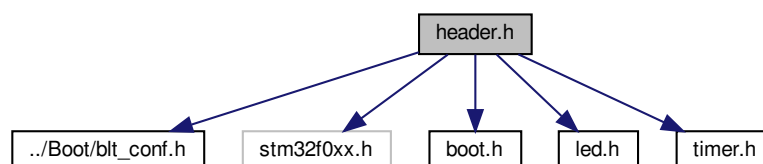
Generic header file.

7.468 header.h File Reference

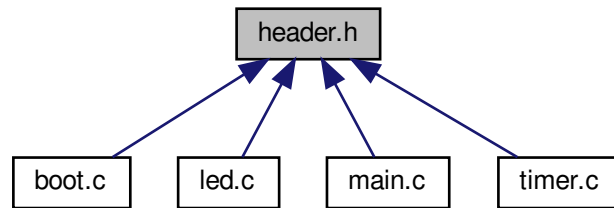
Generic header file.

```
#include "../Boot/blt_conf.h"
#include "stm32f0xx.h"
#include "boot.h"
#include "led.h"
#include "timer.h"
```

Include dependency graph for ARMC00_STM32F0_Discovery_STM32F051_Keil/Prog/header.h:



This graph shows which files directly or indirectly include this file:



7.468.1 Detailed Description

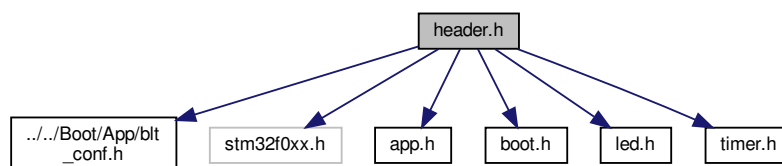
Generic header file.

7.469 header.h File Reference

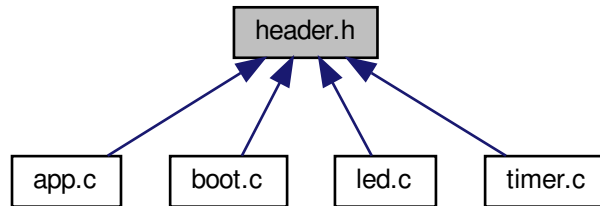
Generic header file.

```
#include "../..../Boot/App/blt_conf.h"
#include "stm32f0xx.h"
#include "app.h"
#include "boot.h"
#include "led.h"
#include "timer.h"
```

Include dependency graph for ARMCM0_STM32F0_Nucleo_F091RC_CubeIDE/Prog/App/header.h:



This graph shows which files directly or indirectly include this file:



7.469.1 Detailed Description

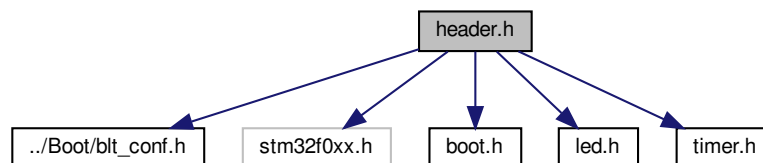
Generic header file.

7.470 header.h File Reference

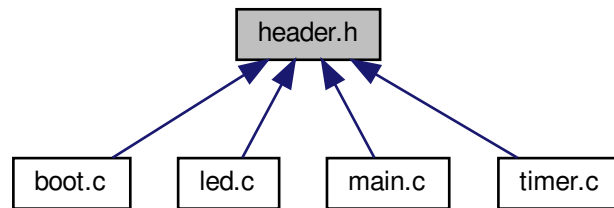
Generic header file.

```
#include "../Boot/blt_conf.h"
#include "stm32f0xx.h"
#include "boot.h"
#include "led.h"
#include "timer.h"
```

Include dependency graph for ARMC00_STM32F0_Nucleo_F091RC_GCC/Prog/header.h:



This graph shows which files directly or indirectly include this file:



7.470.1 Detailed Description

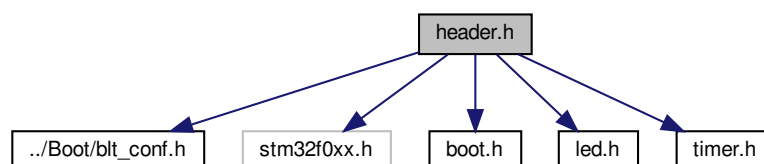
Generic header file.

7.471 header.h File Reference

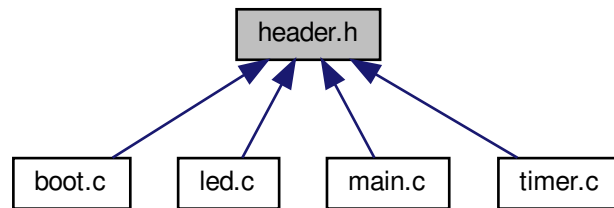
Generic header file.

```
#include "../Boot/blt_conf.h"
#include "stm32f0xx.h"
#include "boot.h"
#include "led.h"
#include "timer.h"
```

Include dependency graph for ARMC00_STM32F0_Nucleo_F091RC_IAR/Prog/header.h:



This graph shows which files directly or indirectly include this file:



7.471.1 Detailed Description

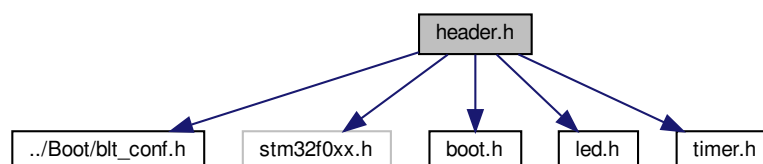
Generic header file.

7.472 header.h File Reference

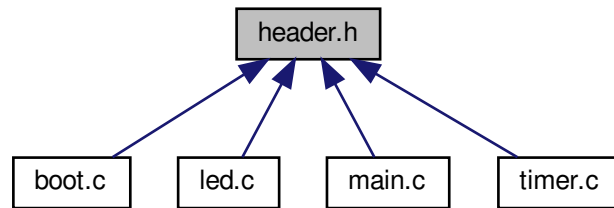
Generic header file.

```
#include "../Boot/blt_conf.h"
#include "stm32f0xx.h"
#include "boot.h"
#include "led.h"
#include "timer.h"
```

Include dependency graph for ARMC00_STM32F0_Nucleo_F091RC_Keil/Prog/header.h:



This graph shows which files directly or indirectly include this file:



7.472.1 Detailed Description

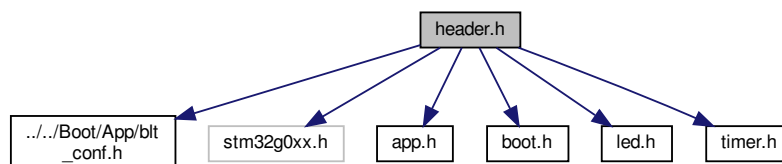
Generic header file.

7.473 header.h File Reference

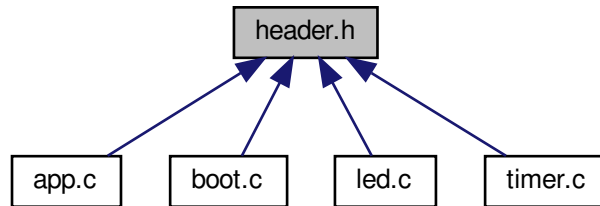
Generic header file.

```
#include "../../Boot/App/blt_conf.h"
#include "stm32g0xx.h"
#include "app.h"
#include "boot.h"
#include "led.h"
#include "timer.h"
```

Include dependency graph for ARMCM0_STM32G0_Nucleo_G071RB_CubeIDE/Prog/App/header.h:



This graph shows which files directly or indirectly include this file:



7.473.1 Detailed Description

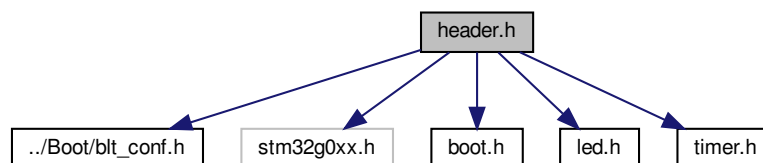
Generic header file.

7.474 header.h File Reference

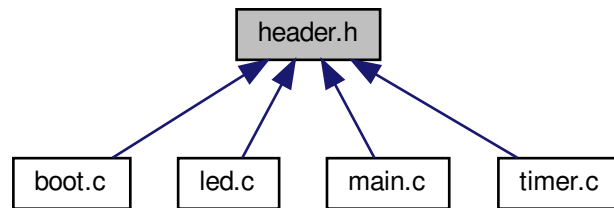
Generic header file.

```
#include "../Boot/blt_conf.h"
#include "stm32g0xx.h"
#include "boot.h"
#include "led.h"
#include "timer.h"
```

Include dependency graph for ARMC00_STM32G0_Nucleo_G071RB_GCC/Prog/header.h:



This graph shows which files directly or indirectly include this file:



7.474.1 Detailed Description

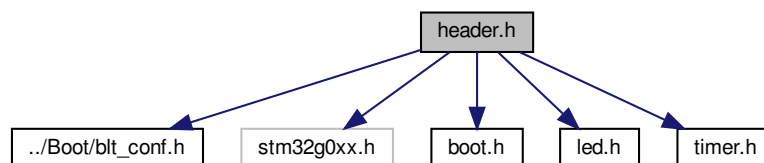
Generic header file.

7.475 header.h File Reference

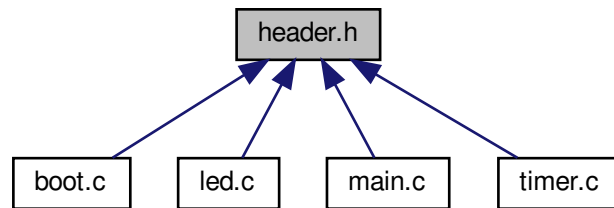
Generic header file.

```
#include "../Boot/blt_conf.h"
#include "stm32g0xx.h"
#include "boot.h"
#include "led.h"
#include "timer.h"
```

Include dependency graph for ARMC00_STM32G0_Nucleo_G071RB_IAR/Prog/header.h:



This graph shows which files directly or indirectly include this file:



7.475.1 Detailed Description

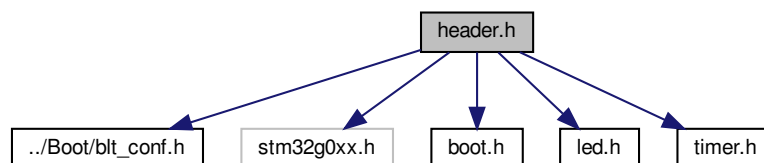
Generic header file.

7.476 header.h File Reference

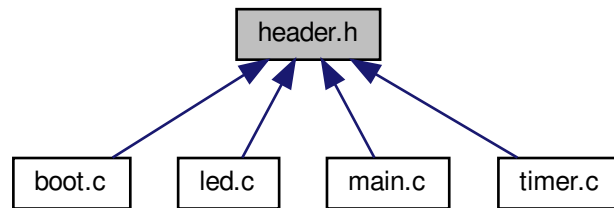
Generic header file.

```
#include "../Boot/blt_conf.h"
#include "stm32g0xx.h"
#include "boot.h"
#include "led.h"
#include "timer.h"
```

Include dependency graph for ARMC00_STM32G0_Nucleo_G071RB_Keil/Prog/header.h:



This graph shows which files directly or indirectly include this file:



7.476.1 Detailed Description

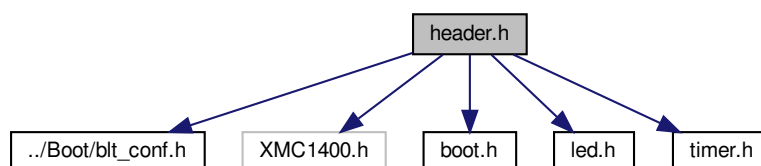
Generic header file.

7.477 header.h File Reference

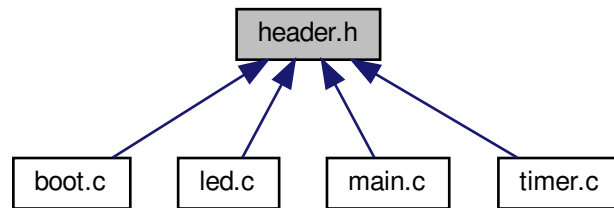
Generic header file.

```
#include "../Boot/blt_conf.h"
#include "XMC1400.h"
#include "boot.h"
#include "led.h"
#include "timer.h"
```

Include dependency graph for ARMCM0_XMC1_XMC1400_Boot_Kit_GCC/Prog/header.h:



This graph shows which files directly or indirectly include this file:



7.477.1 Detailed Description

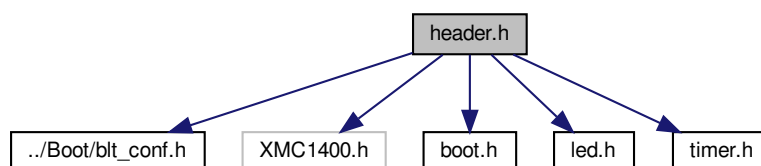
Generic header file.

7.478 header.h File Reference

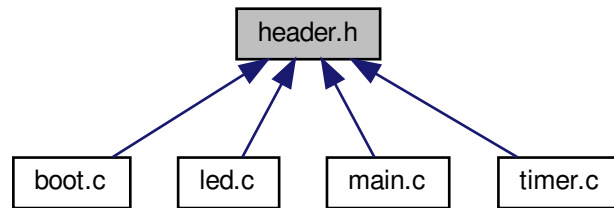
Generic header file.

```
#include "../Boot/blt_conf.h"
#include "XMC1400.h"
#include "boot.h"
#include "led.h"
#include "timer.h"
```

Include dependency graph for ARMCM0_XMC1_XMC1400_Boot_Kit_IAR/Prog/header.h:



This graph shows which files directly or indirectly include this file:



7.478.1 Detailed Description

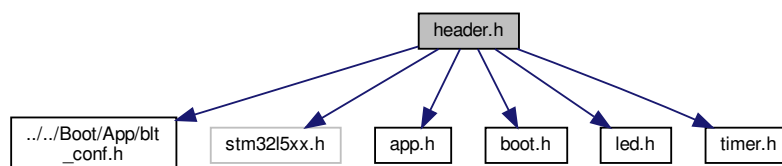
Generic header file.

7.479 header.h File Reference

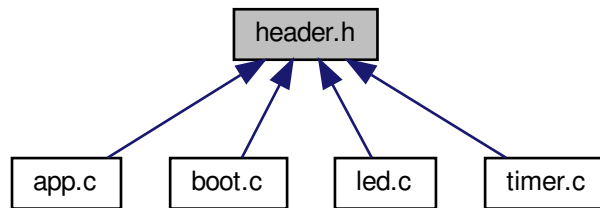
Generic header file.

```
#include "../../Boot/App/blt_conf.h"
#include "stm32l5xx.h"
#include "app.h"
#include "boot.h"
#include "led.h"
#include "timer.h"
```

Include dependency graph for ARMCM33_STM32L5_Nucleo_L552ZE_CubeIDE/Prog/App/header.h:



This graph shows which files directly or indirectly include this file:



7.479.1 Detailed Description

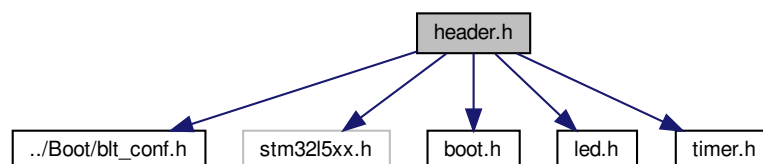
Generic header file.

7.480 header.h File Reference

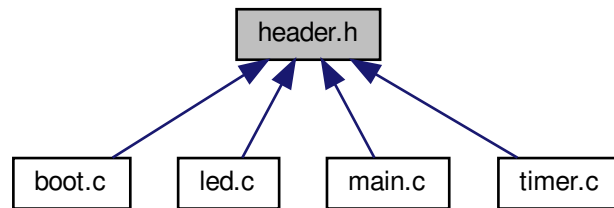
Generic header file.

```
#include "../Boot/blt_conf.h"
#include "stm32l5xx.h"
#include "boot.h"
#include "led.h"
#include "timer.h"
```

Include dependency graph for ARMC33_STM32L5_Nucleo_L552ZE_GCC/Prog/header.h:



This graph shows which files directly or indirectly include this file:



7.480.1 Detailed Description

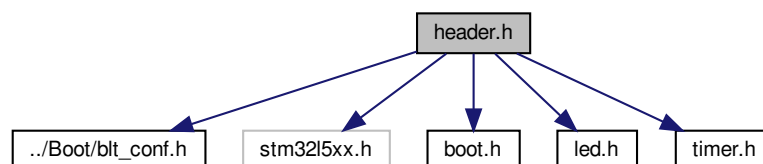
Generic header file.

7.481 header.h File Reference

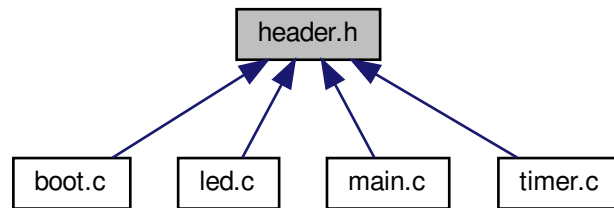
Generic header file.

```
#include "../Boot/blt_conf.h"
#include "stm32l5xx.h"
#include "boot.h"
#include "led.h"
#include "timer.h"
```

Include dependency graph for ARMCM33_STM32L5_Nucleo_L552ZE_IAR/Prog/header.h:



This graph shows which files directly or indirectly include this file:



7.481.1 Detailed Description

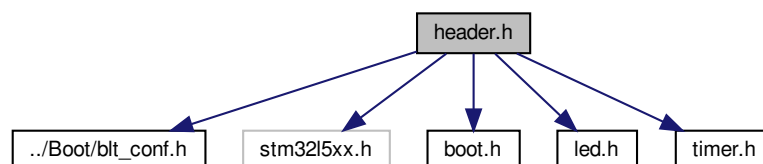
Generic header file.

7.482 header.h File Reference

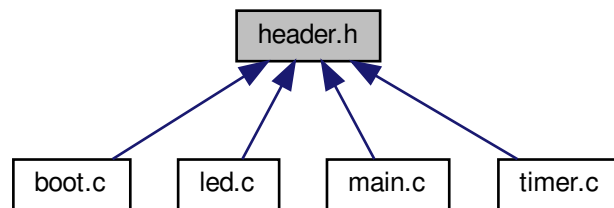
Generic header file.

```
#include "../Boot/blt_conf.h"
#include "stm32l5xx.h"
#include "boot.h"
#include "led.h"
#include "timer.h"
```

Include dependency graph for ARMC33_STM32L5_Nucleo_L552ZE_Keil/Prog/header.h:



This graph shows which files directly or indirectly include this file:



7.482.1 Detailed Description

Generic header file.

7.483 header.h File Reference

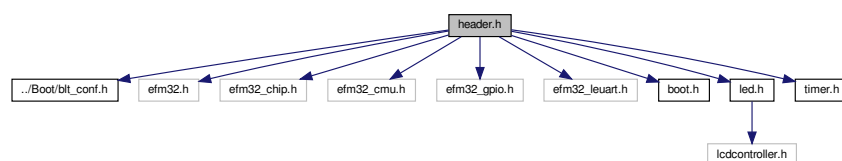
Generic header file.

```

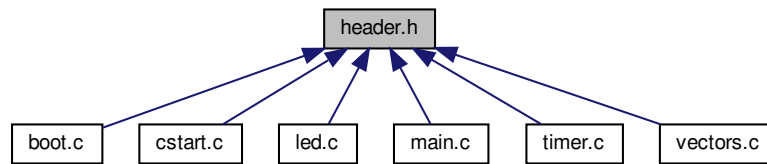
#include "../Boot/blt_conf.h"
#include "efm32.h"
#include "efm32_chip.h"
#include "efm32_cmu.h"
#include "efm32_gpio.h"
#include "efm32_leuart.h"
#include "boot.h"
#include "led.h"
#include "timer.h"

```

Include dependency graph for ARMC32_EFM32_Olimex_EM32G880F128STK_GCC/Prog/header.h:



This graph shows which files directly or indirectly include this file:



7.483.1 Detailed Description

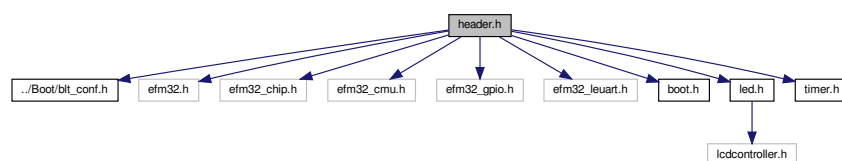
Generic header file.

7.484 header.h File Reference

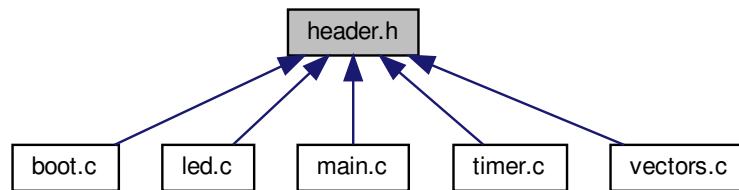
Generic header file.

```
#include "../Boot/blt_conf.h"
#include "efm32.h"
#include "efm32_chip.h"
#include "efm32_cmu.h"
#include "efm32_gpio.h"
#include "efm32_leuart.h"
#include "boot.h"
#include "led.h"
#include "timer.h"
```

Include dependency graph for ARMCM3_EFM32_Olimex_EM32G880F128STK_IAR/Prog/header.h:



This graph shows which files directly or indirectly include this file:



7.484.1 Detailed Description

Generic header file.

7.485 header.h File Reference

Generic header file.

```

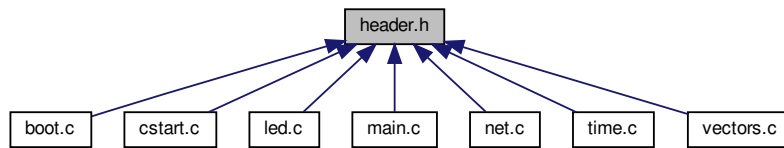
#include "../Boot/blt_conf.h"
#include "boot.h"
#include "led.h"
#include "time.h"
#include "net.h"
#include "shared_params.h"
#include "inc/hw_ints.h"
#include "inc/hw_memmap.h"
#include "inc/hw_nvic.h"
#include "inc/hw_sysctl.h"
#include "inc/hw_types.h"
#include "driverlib/sysctl.h"
#include "driverlib/gpio.h"
#include "driverlib/uart.h"
#include "driverlib/interrupt.h"
#include "driverlib/systick.h"

```

Include dependency graph for ARMCM3_LM3S_EK_LM3S6965_GCC/Prog/header.h:



This graph shows which files directly or indirectly include this file:



7.485.1 Detailed Description

Generic header file.

7.486 header.h File Reference

Generic header file.

```

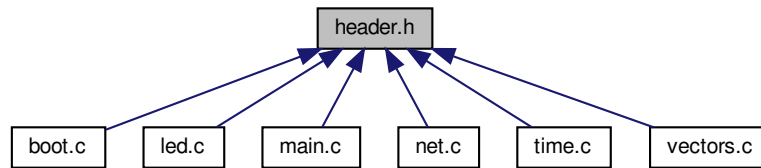
#include "../Boot/blt_conf.h"
#include "boot.h"
#include "led.h"
#include "time.h"
#include "net.h"
#include "shared_params.h"
#include "inc/hw_ints.h"
#include "inc/hw_memmap.h"
#include "inc/hw_nvic.h"
#include "inc/hw_sysctl.h"
#include "inc/hw_types.h"
#include "driverlib/sysctl.h"
#include "driverlib/gpio.h"
#include "driverlib/uart.h"
#include "driverlib/interrupt.h"
#include "driverlib/systick.h"

```

Include dependency graph for ARMCM3_LM3S_EK_LM3S6965_IAR/Prog/header.h:



This graph shows which files directly or indirectly include this file:



7.486.1 Detailed Description

Generic header file.

7.487 header.h File Reference

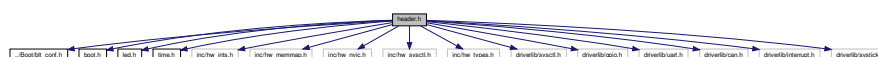
Generic header file.

```

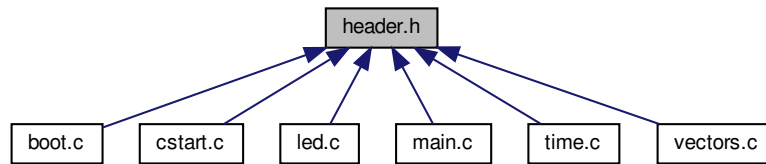
#include "../Boot/blt_conf.h"
#include "boot.h"
#include "led.h"
#include "time.h"
#include "inc/hw_ints.h"
#include "inc/hw_memmap.h"
#include "inc/hw_nvic.h"
#include "inc/hw_sysctl.h"
#include "inc/hw_types.h"
#include "driverlib/sysctl.h"
#include "driverlib/gpio.h"
#include "driverlib/uart.h"
#include "driverlib/can.h"
#include "driverlib/interrupt.h"
#include "driverlib/systick.h"

```

Include dependency graph for ARMCM3_LM3S_EK_LM3S8962_GCC/Prog/header.h:



This graph shows which files directly or indirectly include this file:



7.487.1 Detailed Description

Generic header file.

7.488 header.h File Reference

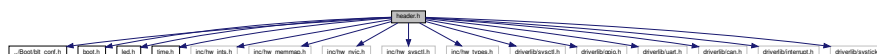
Generic header file.

```

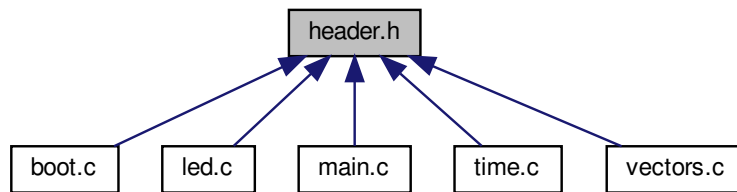
#include "../Boot/blt_conf.h"
#include "boot.h"
#include "led.h"
#include "time.h"
#include "inc/hw_ints.h"
#include "inc/hw_memmap.h"
#include "inc/hw_nvic.h"
#include "inc/hw_sysctl.h"
#include "inc/hw_types.h"
#include "driverlib/sysctl.h"
#include "driverlib/gpio.h"
#include "driverlib/uart.h"
#include "driverlib/can.h"
#include "driverlib/interrupt.h"
#include "driverlib/systick.h"

```

Include dependency graph for ARMCM3_LM3S_EK_LM3S8962_IAR/Prog/header.h:



This graph shows which files directly or indirectly include this file:



7.488.1 Detailed Description

Generic header file.

7.489 header.h File Reference

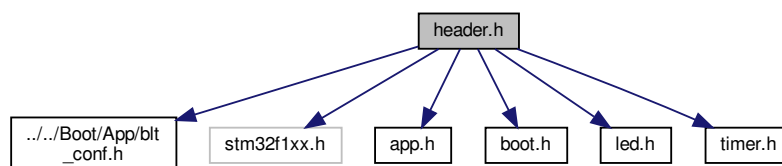
Generic header file.

```

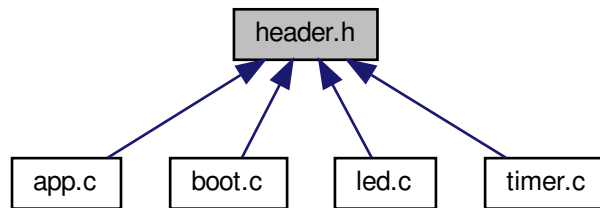
#include "../Boot/App/blt_conf.h"
#include "stm32f1xx.h"
#include "app.h"
#include "boot.h"
#include "led.h"
#include "timer.h"

```

Include dependency graph for ARMCM3_STM32F1_Nucleo_F103RB_CubeIDE/Prog/App/header.h:



This graph shows which files directly or indirectly include this file:



7.489.1 Detailed Description

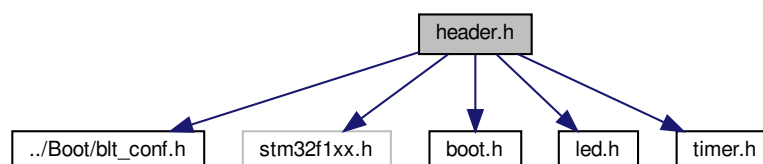
Generic header file.

7.490 header.h File Reference

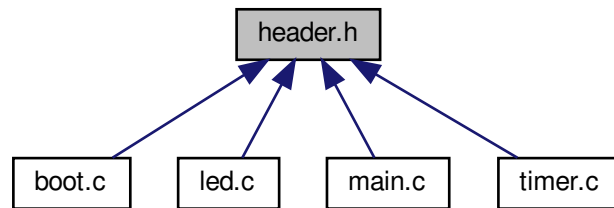
Generic header file.

```
#include "../Boot/blt_conf.h"
#include "stm32f1xx.h"
#include "boot.h"
#include "led.h"
#include "timer.h"
```

Include dependency graph for ARMC32F1_Nucleo_F103RB_GCC/Prog/header.h:



This graph shows which files directly or indirectly include this file:



7.490.1 Detailed Description

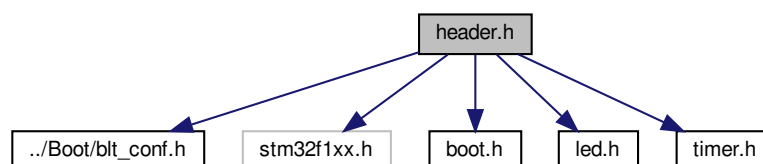
Generic header file.

7.491 header.h File Reference

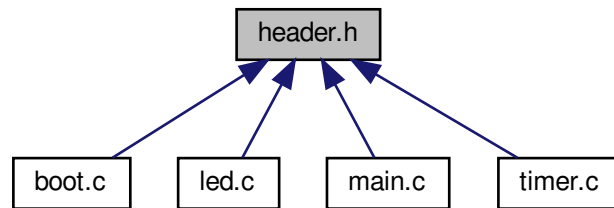
Generic header file.

```
#include "../Boot/blt_conf.h"
#include "stm32f1xx.h"
#include "boot.h"
#include "led.h"
#include "timer.h"
```

Include dependency graph for ARMC3M3_STM32F1_Nucleo_F103RB_IAR/Prog/header.h:



This graph shows which files directly or indirectly include this file:



7.491.1 Detailed Description

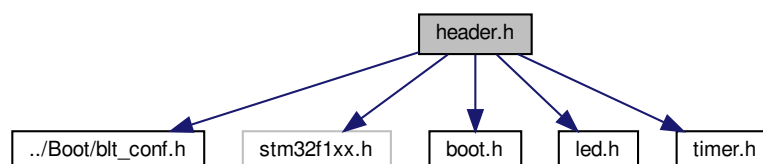
Generic header file.

7.492 header.h File Reference

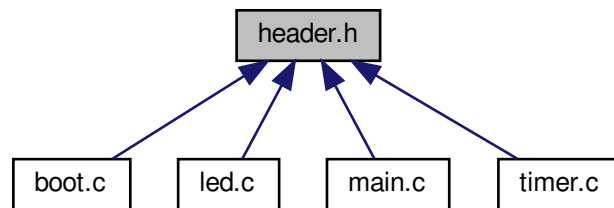
Generic header file.

```
#include "../Boot/blt_conf.h"
#include "stm32f1xx.h"
#include "boot.h"
#include "led.h"
#include "timer.h"
```

Include dependency graph for ARMC32F1_Nucleo_F103RB_Keil/Prog/header.h:



This graph shows which files directly or indirectly include this file:



7.492.1 Detailed Description

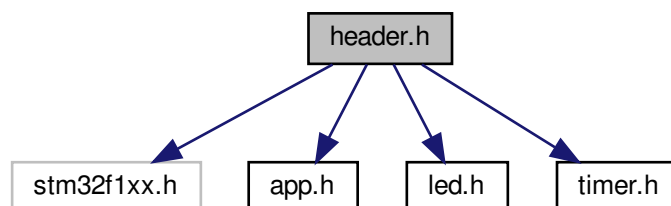
Generic header file.

7.493 header.h File Reference

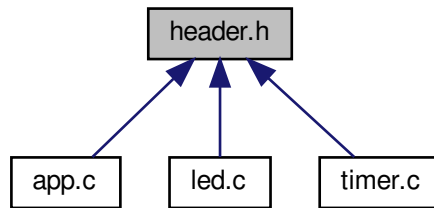
Generic header file.

```
#include "stm32f1xx.h"  
#include "app.h"  
#include "led.h"  
#include "timer.h"
```

Include dependency graph for ARMCM3_STM32F1_Olimex_STM32H103_CubeIDE/Prog/App/header.h:



This graph shows which files directly or indirectly include this file:



7.493.1 Detailed Description

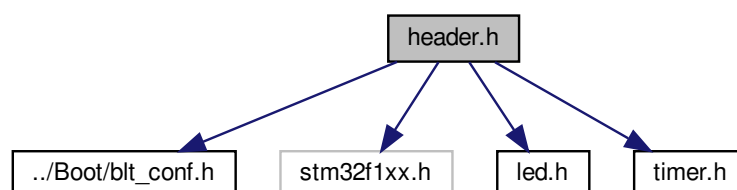
Generic header file.

7.494 header.h File Reference

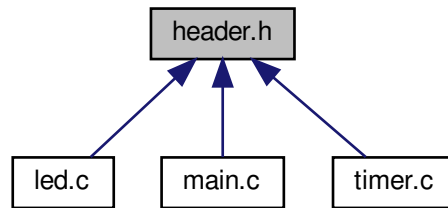
Generic header file.

```
#include "../Boot/blt_conf.h"
#include "stm32f1xx.h"
#include "led.h"
#include "timer.h"
```

Include dependency graph for ARMCM3_STM32F1_Olimex_STM32H103_GCC/Prog/header.h:



This graph shows which files directly or indirectly include this file:



7.494.1 Detailed Description

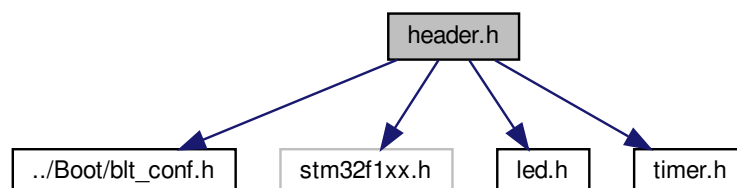
Generic header file.

7.495 header.h File Reference

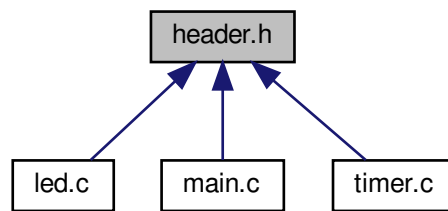
Generic header file.

```
#include "../Boot/blt_conf.h"  
#include "stm32f1xx.h"  
#include "led.h"  
#include "timer.h"
```

Include dependency graph for ARMCM3_STM32F1_Olimex_STM32H103_IAR/Prog/header.h:



This graph shows which files directly or indirectly include this file:



7.495.1 Detailed Description

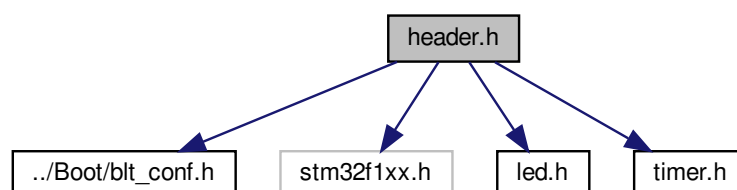
Generic header file.

7.496 header.h File Reference

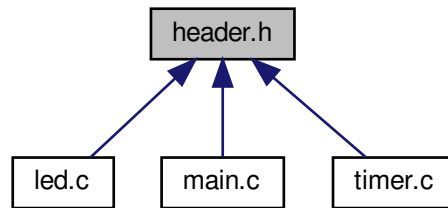
Generic header file.

```
#include "../Boot/blt_conf.h"  
#include "stm32f1xx.h"  
#include "led.h"  
#include "timer.h"
```

Include dependency graph for ARMCM3_STM32F1_Olimex_STM32H103_Keil/Prog/header.h:



This graph shows which files directly or indirectly include this file:



7.496.1 Detailed Description

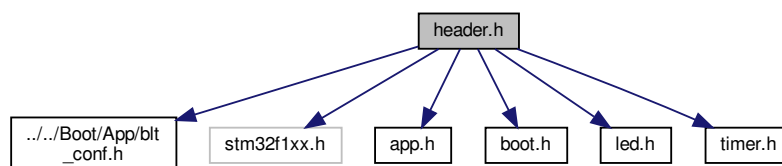
Generic header file.

7.497 header.h File Reference

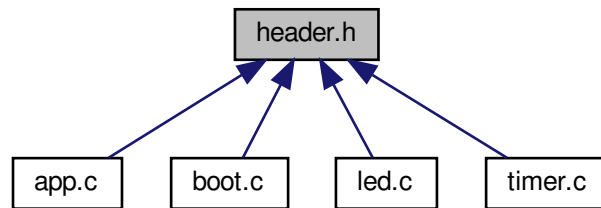
Generic header file.

```
#include "../../Boot/App/blt_conf.h"
#include "stm32f1xx.h"
#include "app.h"
#include "boot.h"
#include "led.h"
#include "timer.h"
```

Include dependency graph for ARMCM3_STM32F1_Olimex_STM32P103_CubeIDE/Prog/App/header.h:



This graph shows which files directly or indirectly include this file:



7.497.1 Detailed Description

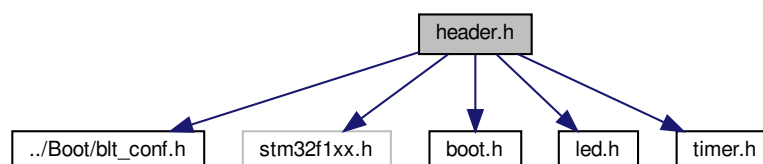
Generic header file.

7.498 header.h File Reference

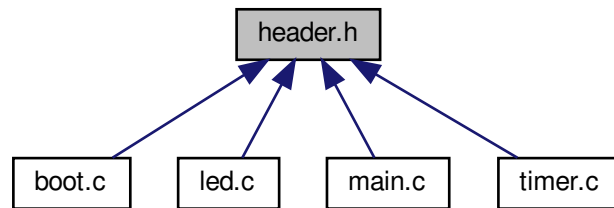
Generic header file.

```
#include "../Boot/blt_conf.h"
#include "stm32f1xx.h"
#include "boot.h"
#include "led.h"
#include "timer.h"
```

Include dependency graph for ARMC32F1_Olimex_STM32P103_GCC/Prog/header.h:



This graph shows which files directly or indirectly include this file:



7.498.1 Detailed Description

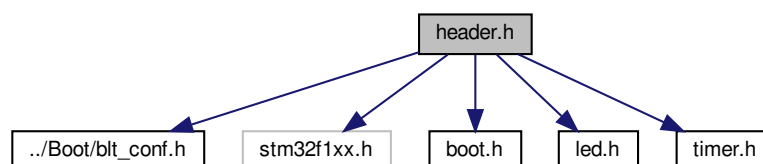
Generic header file.

7.499 header.h File Reference

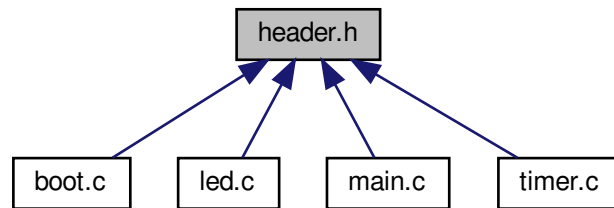
Generic header file.

```
#include "../Boot/blt_conf.h"
#include "stm32f1xx.h"
#include "boot.h"
#include "led.h"
#include "timer.h"
```

Include dependency graph for `ARMCM3_STM32F1_Olimex_STM32P103_IAR/Prog/header.h`:



This graph shows which files directly or indirectly include this file:



7.499.1 Detailed Description

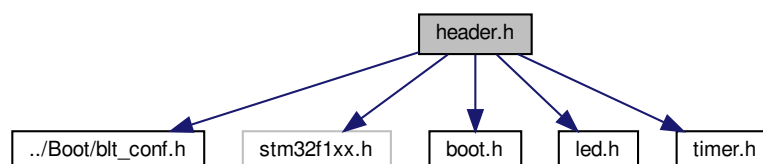
Generic header file.

7.500 header.h File Reference

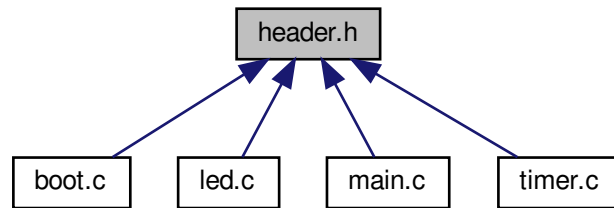
Generic header file.

```
#include "../Boot/blt_conf.h"
#include "stm32f1xx.h"
#include "boot.h"
#include "led.h"
#include "timer.h"
```

Include dependency graph for ARMC32F1_Olimex_STM32P103_Keil/Prog/header.h:



This graph shows which files directly or indirectly include this file:



7.500.1 Detailed Description

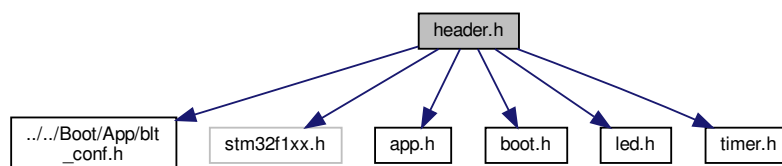
Generic header file.

7.501 header.h File Reference

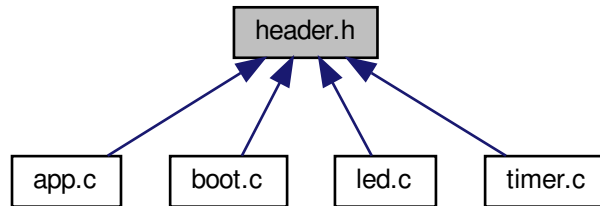
Generic header file.

```
#include "../../Boot/App/blt_conf.h"
#include "stm32f1xx.h"
#include "app.h"
#include "boot.h"
#include "led.h"
#include "timer.h"
```

Include dependency graph for ARMCM3_STM32F1_Olimexino_STM32_CubeIDE/Prog/App/header.h:



This graph shows which files directly or indirectly include this file:



7.501.1 Detailed Description

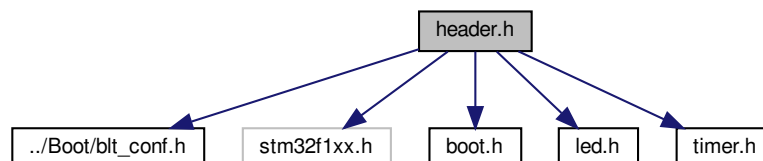
Generic header file.

7.502 header.h File Reference

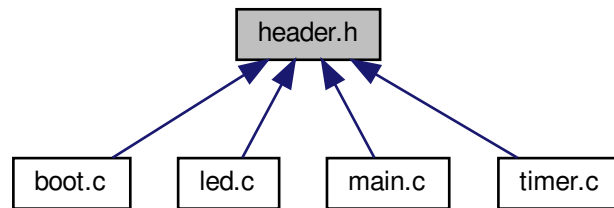
Generic header file.

```
#include "../Boot/blt_conf.h"
#include "stm32f1xx.h"
#include "boot.h"
#include "led.h"
#include "timer.h"
```

Include dependency graph for ARMC32F1_Olimexino_STM32_GCC/Prog/header.h:



This graph shows which files directly or indirectly include this file:



7.502.1 Detailed Description

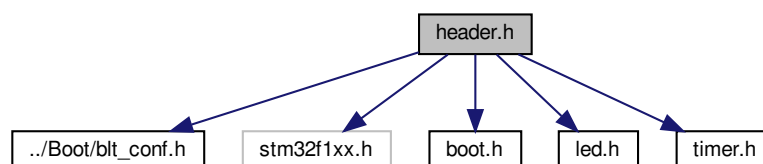
Generic header file.

7.503 header.h File Reference

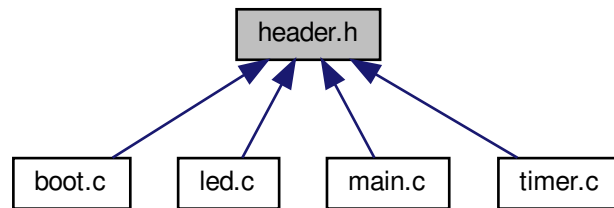
Generic header file.

```
#include "../Boot/blt_conf.h"
#include "stm32f1xx.h"
#include "boot.h"
#include "led.h"
#include "timer.h"
```

Include dependency graph for ARMC32F1_Olimexino_STM32_IAR/Prog/header.h:



This graph shows which files directly or indirectly include this file:



7.503.1 Detailed Description

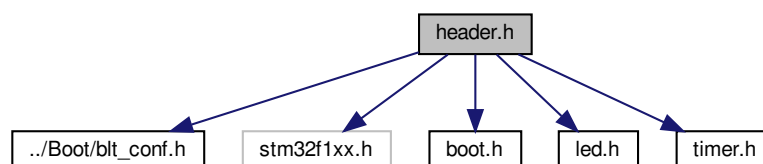
Generic header file.

7.504 header.h File Reference

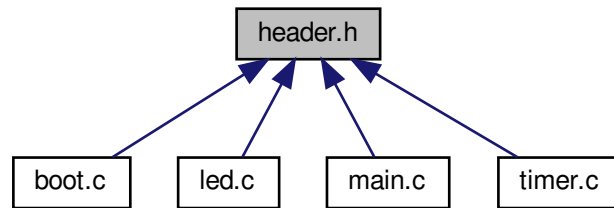
Generic header file.

```
#include "../Boot/blt_conf.h"
#include "stm32f1xx.h"
#include "boot.h"
#include "led.h"
#include "timer.h"
```

Include dependency graph for `ARMCM3_STM32F1_Olimexino_STM32_Keil/Prog/header.h`:



This graph shows which files directly or indirectly include this file:



7.504.1 Detailed Description

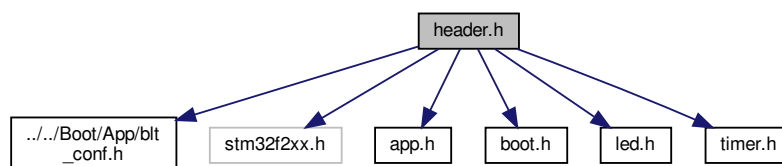
Generic header file.

7.505 header.h File Reference

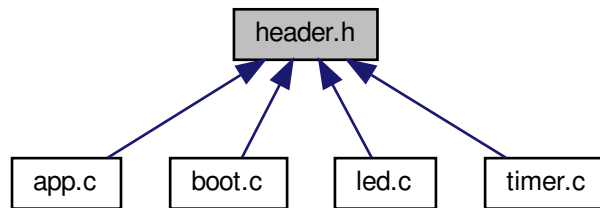
Generic header file.

```
#include "../..../Boot/App/blt_conf.h"
#include "stm32f2xx.h"
#include "app.h"
#include "boot.h"
#include "led.h"
#include "timer.h"
```

Include dependency graph for ARMCM3_STM32F2_Olimex_STM32P207_CubeIDE/Prog/App/header.h:



This graph shows which files directly or indirectly include this file:



7.505.1 Detailed Description

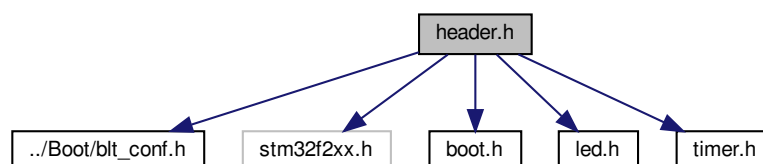
Generic header file.

7.506 header.h File Reference

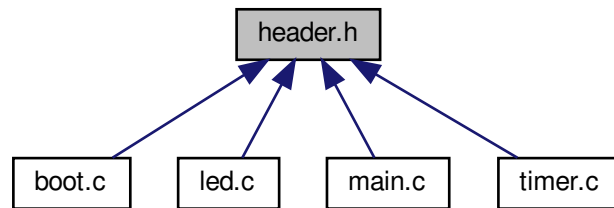
Generic header file.

```
#include "../Boot/blt_conf.h"
#include "stm32f2xx.h"
#include "boot.h"
#include "led.h"
#include "timer.h"
```

Include dependency graph for ARMC32F2_Olimex_STM32P207_GCC/Prog/header.h:



This graph shows which files directly or indirectly include this file:



7.506.1 Detailed Description

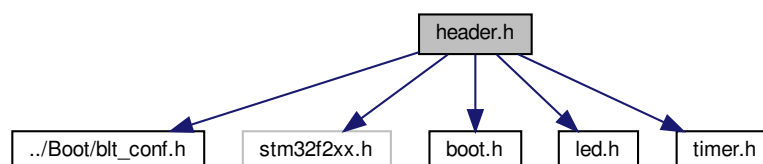
Generic header file.

7.507 header.h File Reference

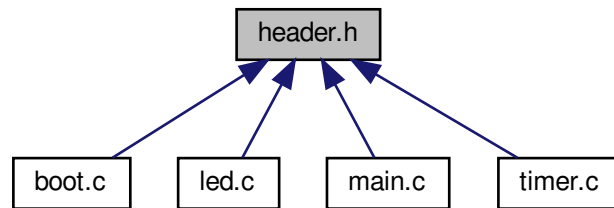
Generic header file.

```
#include "../Boot/blt_conf.h"
#include "stm32f2xx.h"
#include "boot.h"
#include "led.h"
#include "timer.h"
```

Include dependency graph for ARMC32F2_Olimex_STM32P207_IAR/Prog/header.h:



This graph shows which files directly or indirectly include this file:



7.507.1 Detailed Description

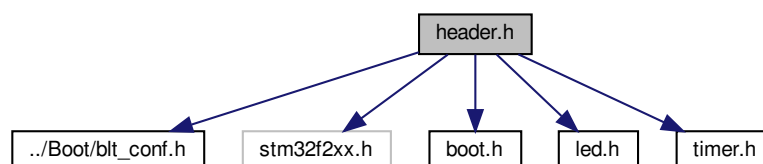
Generic header file.

7.508 header.h File Reference

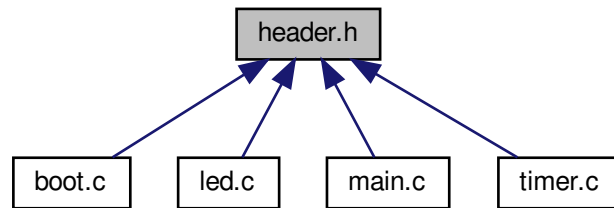
Generic header file.

```
#include "../Boot/blt_conf.h"
#include "stm32f2xx.h"
#include "boot.h"
#include "led.h"
#include "timer.h"
```

Include dependency graph for ARMC32F2_Olimex_STM32P207_Keil/Prog/header.h:



This graph shows which files directly or indirectly include this file:



7.508.1 Detailed Description

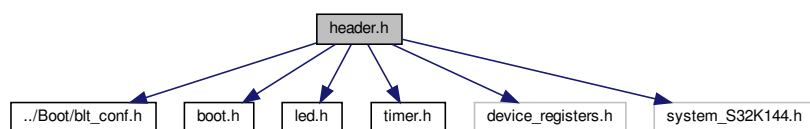
Generic header file.

7.509 header.h File Reference

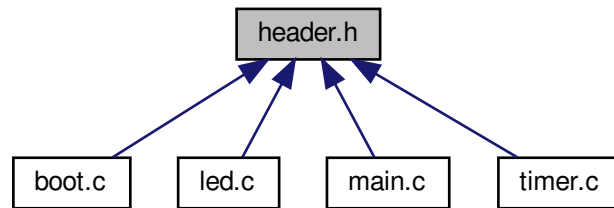
Generic header file.

```
#include "../Boot/blt_conf.h"
#include "boot.h"
#include "led.h"
#include "timer.h"
#include "device_registers.h"
#include "system_S32K144.h"
```

Include dependency graph for ARMCM4_S32K14_S32K144EVB_GCC/Prog/header.h:



This graph shows which files directly or indirectly include this file:



7.509.1 Detailed Description

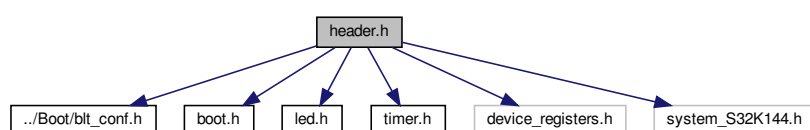
Generic header file.

7.510 header.h File Reference

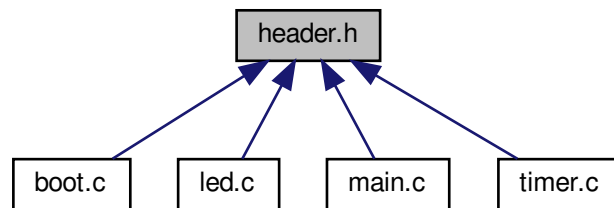
Generic header file.

```
#include "../Boot/blt_conf.h"
#include "boot.h"
#include "led.h"
#include "timer.h"
#include "device_registers.h"
#include "system_S32K144.h"
```

Include dependency graph for ARMCM4_S32K14_S32K144EVB_IAR/Prog/header.h:



This graph shows which files directly or indirectly include this file:



7.510.1 Detailed Description

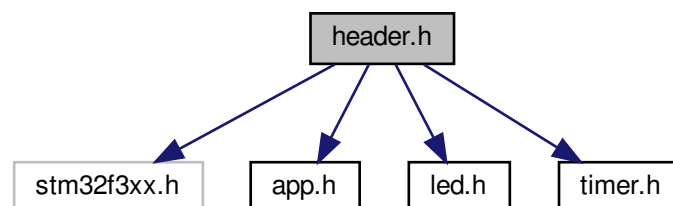
Generic header file.

7.511 header.h File Reference

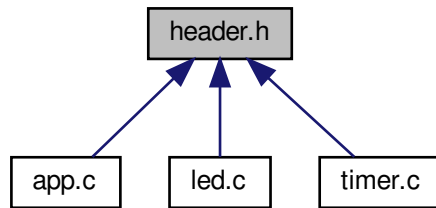
Generic header file.

```
#include "stm32f3xx.h"  
#include "app.h"  
#include "led.h"  
#include "timer.h"
```

Include dependency graph for ARMCM4_STM32F3_Discovery_F303VC_CubeIDE/Prog/App/header.h:



This graph shows which files directly or indirectly include this file:



7.511.1 Detailed Description

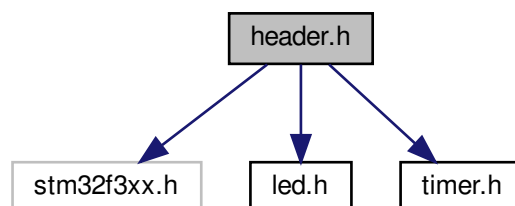
Generic header file.

7.512 header.h File Reference

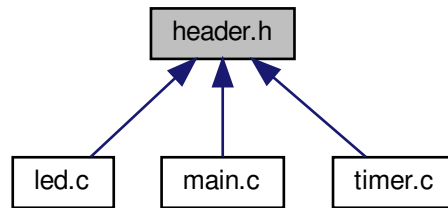
Generic header file.

```
#include "stm32f3xx.h"  
#include "led.h"  
#include "timer.h"
```

Include dependency graph for ARMCM4_STM32F3_Discovery_F303VC_GCC/Prog/header.h:



This graph shows which files directly or indirectly include this file:



7.512.1 Detailed Description

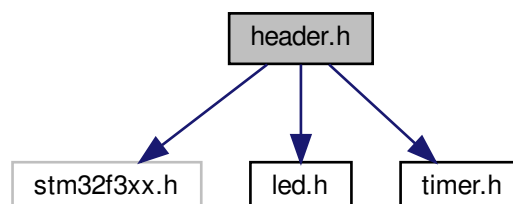
Generic header file.

7.513 header.h File Reference

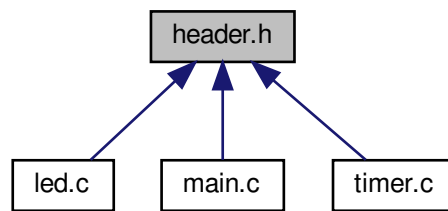
Generic header file.

```
#include "stm32f3xx.h"  
#include "led.h"  
#include "timer.h"
```

Include dependency graph for ARMCM4_STM32F3_Discovery_F303VC_IAR/Prog/header.h:



This graph shows which files directly or indirectly include this file:



7.513.1 Detailed Description

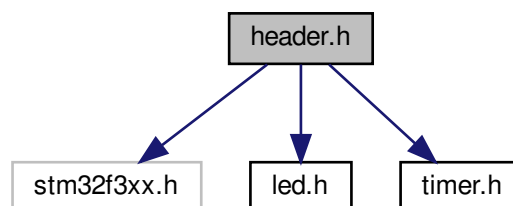
Generic header file.

7.514 header.h File Reference

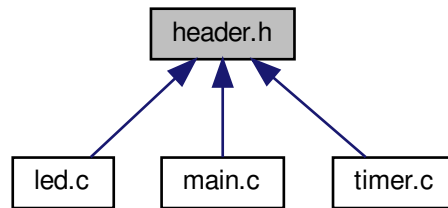
Generic header file.

```
#include "stm32f3xx.h"  
#include "led.h"  
#include "timer.h"
```

Include dependency graph for ARMCM4_STM32F3_Discovery_F303VC_Keil/Prog/header.h:



This graph shows which files directly or indirectly include this file:



7.514.1 Detailed Description

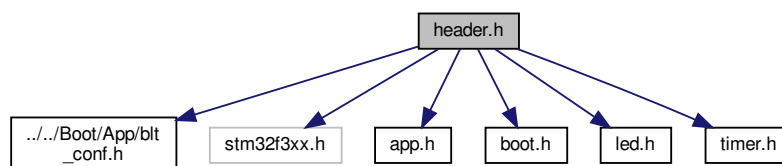
Generic header file.

7.515 header.h File Reference

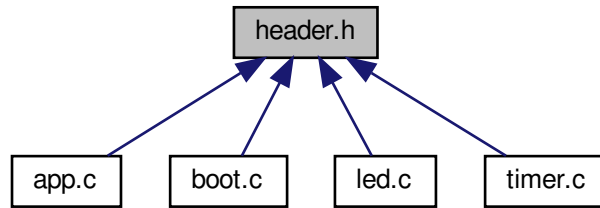
Generic header file.

```
#include "../..../Boot/App/blt_conf.h"
#include "stm32f3xx.h"
#include "app.h"
#include "boot.h"
#include "led.h"
#include "timer.h"
```

Include dependency graph for ARMCM4_STM32F3_Nucleo_F303K8_CubeIDE/Prog/App/header.h:



This graph shows which files directly or indirectly include this file:



7.515.1 Detailed Description

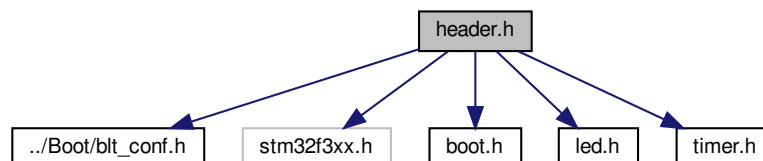
Generic header file.

7.516 header.h File Reference

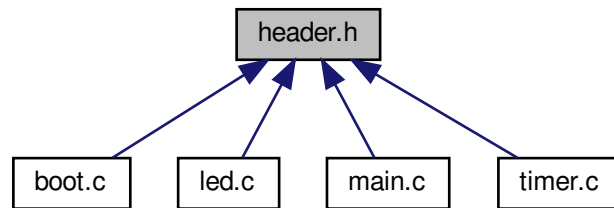
Generic header file.

```
#include "../Boot/blt_conf.h"
#include "stm32f3xx.h"
#include "boot.h"
#include "led.h"
#include "timer.h"
```

Include dependency graph for ARMC4_STM32F3_Nucleo_F303K8_GCC/Prog/header.h:



This graph shows which files directly or indirectly include this file:



7.516.1 Detailed Description

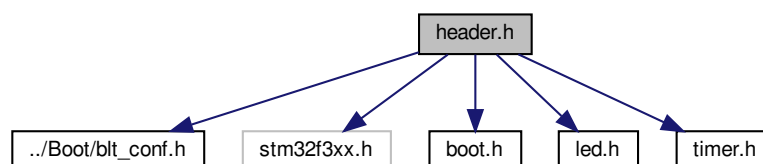
Generic header file.

7.517 header.h File Reference

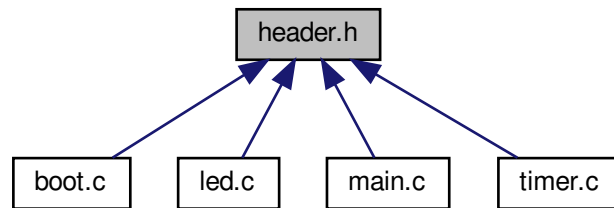
Generic header file.

```
#include "../Boot/blt_conf.h"
#include "stm32f3xx.h"
#include "boot.h"
#include "led.h"
#include "timer.h"
```

Include dependency graph for ARMC4_STM32F3_Nucleo_F303K8_IAR/Prog/header.h:



This graph shows which files directly or indirectly include this file:



7.517.1 Detailed Description

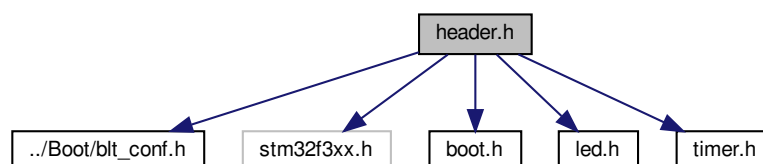
Generic header file.

7.518 header.h File Reference

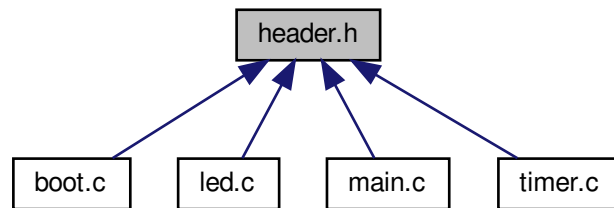
Generic header file.

```
#include "../Boot/blt_conf.h"
#include "stm32f3xx.h"
#include "boot.h"
#include "led.h"
#include "timer.h"
```

Include dependency graph for ARMC4_STM32F3_Nucleo_F303K8_Keil/Prog/header.h:



This graph shows which files directly or indirectly include this file:



7.518.1 Detailed Description

Generic header file.

7.519 header.h File Reference

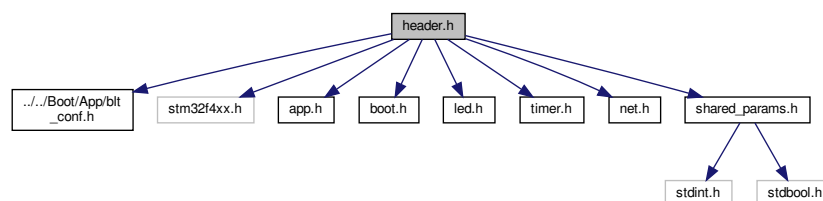
Generic header file.

```

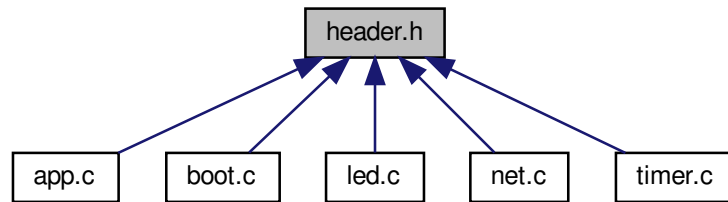
#include "../../Boot/App/blt_conf.h"
#include "stm32f4xx.h"
#include "app.h"
#include "boot.h"
#include "led.h"
#include "timer.h"
#include "net.h"
#include "shared_params.h"

```

Include dependency graph for ARMCM4_STM32F4_Nucleo_F429ZI_CubeIDE/Prog/App/header.h:



This graph shows which files directly or indirectly include this file:



7.519.1 Detailed Description

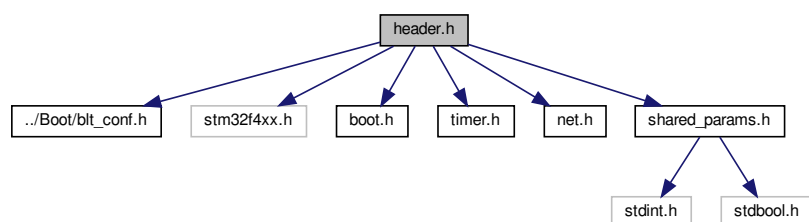
Generic header file.

7.520 header.h File Reference

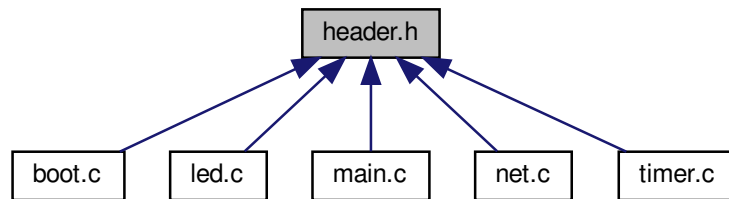
Generic header file.

```
#include "../Boot/blt_conf.h"
#include "stm32f4xx.h"
#include "boot.h"
#include "led.h"
#include "timer.h"
#include "net.h"
#include "shared_params.h"
```

Include dependency graph for ARMCM4_STM32F4_Nucleo_F429ZI_GCC/Prog/header.h:



This graph shows which files directly or indirectly include this file:



7.520.1 Detailed Description

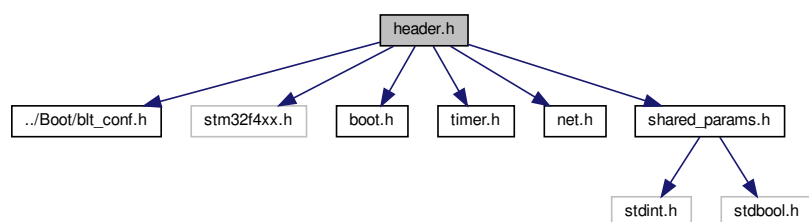
Generic header file.

7.521 header.h File Reference

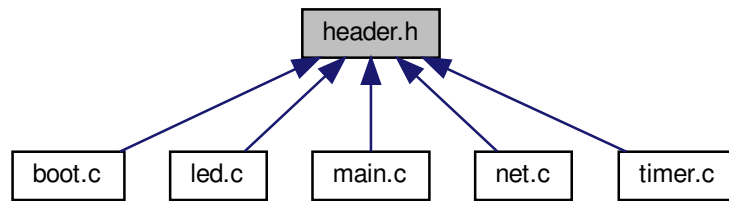
Generic header file.

```
#include "../Boot/blt_conf.h"
#include "stm32f4xx.h"
#include "boot.h"
#include "led.h"
#include "timer.h"
#include "net.h"
#include "shared_params.h"
```

Include dependency graph for ARMCM4_STM32F4_Nucleo_F429ZI_IAR/Prog/header.h:



This graph shows which files directly or indirectly include this file:



7.521.1 Detailed Description

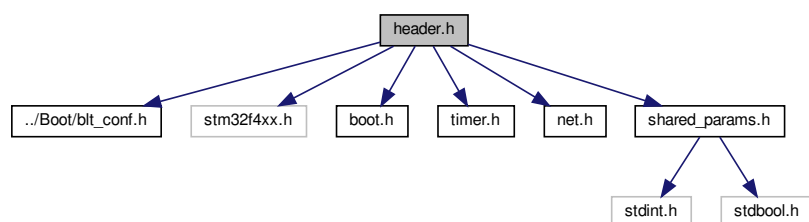
Generic header file.

7.522 header.h File Reference

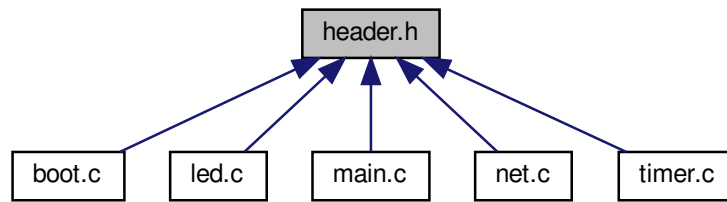
Generic header file.

```
#include "../Boot/blt_conf.h"
#include "stm32f4xx.h"
#include "boot.h"
#include "led.h"
#include "timer.h"
#include "net.h"
#include "shared_params.h"
```

Include dependency graph for ARMCM4_STM32F4_Nucleo_F429ZI_Keil/Prog/header.h:



This graph shows which files directly or indirectly include this file:



7.522.1 Detailed Description

Generic header file.

7.523 header.h File Reference

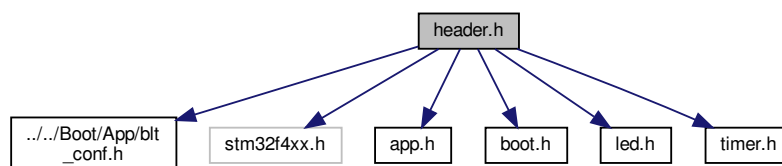
Generic header file.

```

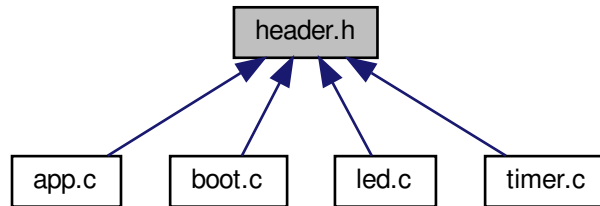
#include "../Boot/App/blt_conf.h"
#include "stm32f4xx.h"
#include "app.h"
#include "boot.h"
#include "led.h"
#include "timer.h"

```

Include dependency graph for ARMCM4_STM32F4_Olimex_STM32P405_CubeIDE/Prog/App/header.h:



This graph shows which files directly or indirectly include this file:



7.523.1 Detailed Description

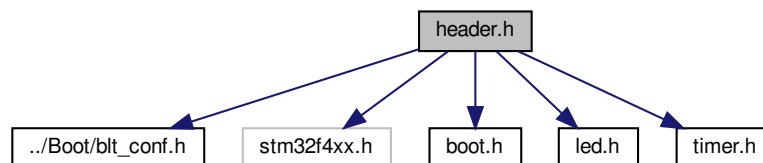
Generic header file.

7.524 header.h File Reference

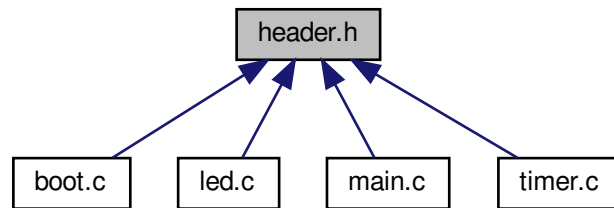
Generic header file.

```
#include "../Boot/blt_conf.h"
#include "stm32f4xx.h"
#include "boot.h"
#include "led.h"
#include "timer.h"
```

Include dependency graph for ARMC4_STM32F4_Olimex_STM32P405_GCC/Prog/header.h:



This graph shows which files directly or indirectly include this file:



7.524.1 Detailed Description

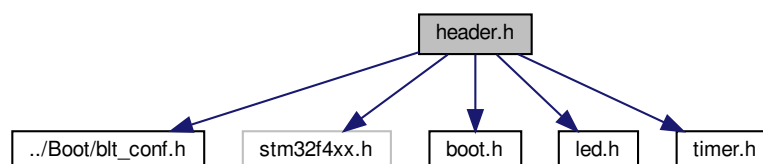
Generic header file.

7.525 header.h File Reference

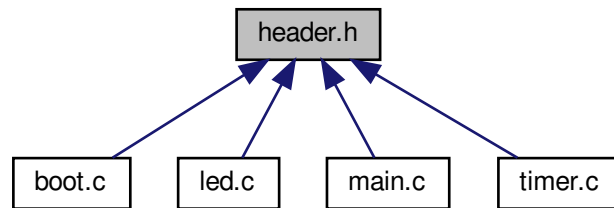
Generic header file.

```
#include "../Boot/blt_conf.h"
#include "stm32f4xx.h"
#include "boot.h"
#include "led.h"
#include "timer.h"
```

Include dependency graph for `ARMCM4_STM32F4_Olimex_STM32P405_IAR/Prog/header.h`:



This graph shows which files directly or indirectly include this file:



7.525.1 Detailed Description

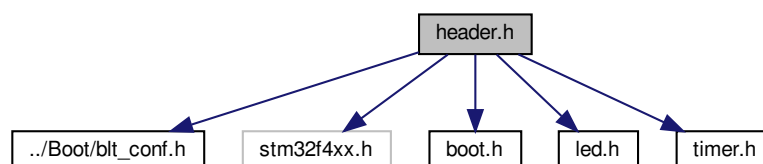
Generic header file.

7.526 header.h File Reference

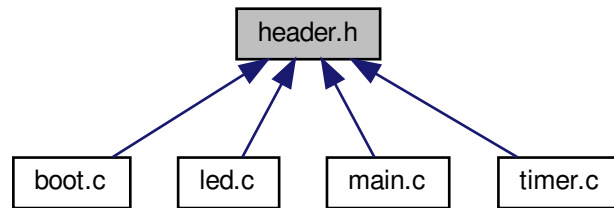
Generic header file.

```
#include "../Boot/blt_conf.h"
#include "stm32f4xx.h"
#include "boot.h"
#include "led.h"
#include "timer.h"
```

Include dependency graph for ARMC4_STM32F4_Olimex_STM32P405_Keil/Prog/header.h:



This graph shows which files directly or indirectly include this file:



7.526.1 Detailed Description

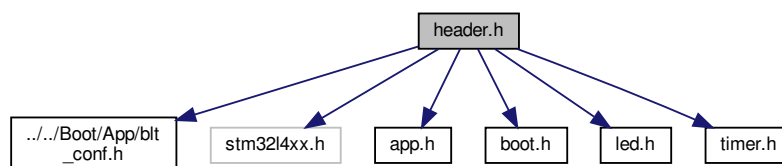
Generic header file.

7.527 header.h File Reference

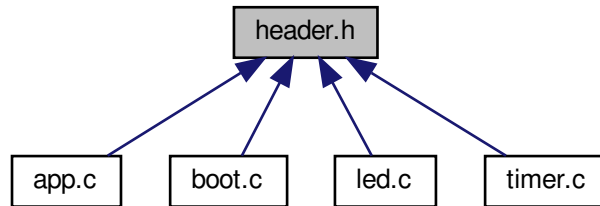
Generic header file.

```
#include "../../Boot/App/blt_conf.h"
#include "stm32l4xx.h"
#include "app.h"
#include "boot.h"
#include "led.h"
#include "timer.h"
```

Include dependency graph for ARMCM4_STM32L4_Nucleo_L476RG_CubeIDE/Prog/App/header.h:



This graph shows which files directly or indirectly include this file:



7.527.1 Detailed Description

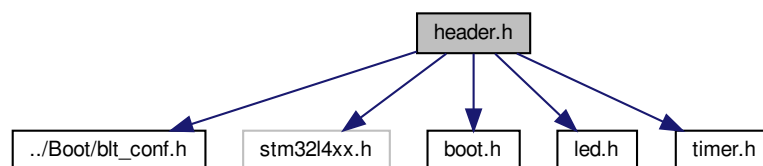
Generic header file.

7.528 header.h File Reference

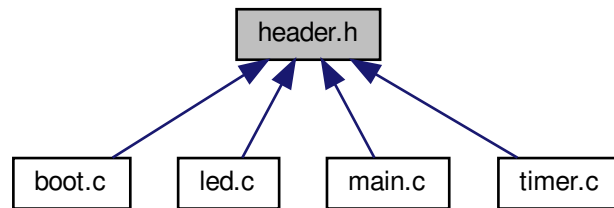
Generic header file.

```
#include "../Boot/blt_conf.h"
#include "stm32l4xx.h"
#include "boot.h"
#include "led.h"
#include "timer.h"
```

Include dependency graph for ARMC4_STM32L4_Nucleo_L476RG_GCC/Prog/header.h:



This graph shows which files directly or indirectly include this file:



7.528.1 Detailed Description

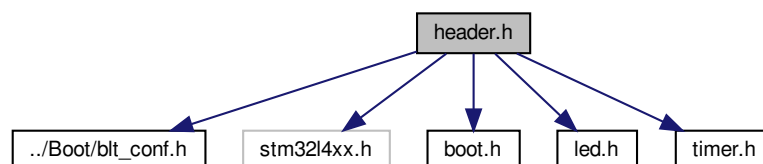
Generic header file.

7.529 header.h File Reference

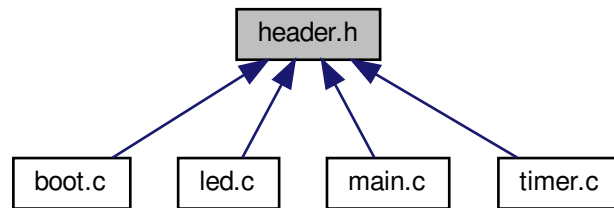
Generic header file.

```
#include "../Boot/blt_conf.h"
#include "stm32l4xx.h"
#include "boot.h"
#include "led.h"
#include "timer.h"
```

Include dependency graph for ARMC4_STM32L4_Nucleo_L476RG_IAR/Prog/header.h:



This graph shows which files directly or indirectly include this file:



7.529.1 Detailed Description

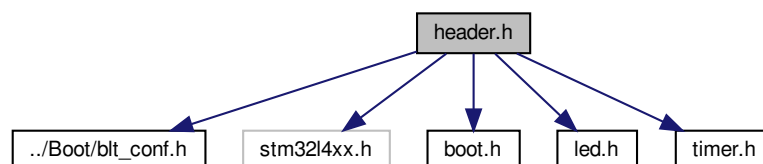
Generic header file.

7.530 header.h File Reference

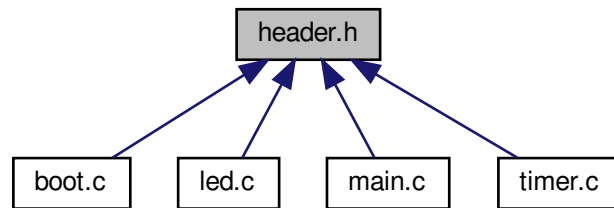
Generic header file.

```
#include "../Boot/blt_conf.h"  
#include "stm32l4xx.h"  
#include "boot.h"  
#include "led.h"  
#include "timer.h"
```

Include dependency graph for ARMC4_STM32L4_Nucleo_L476RG_Keil/Prog/header.h:



This graph shows which files directly or indirectly include this file:



7.530.1 Detailed Description

Generic header file.

7.531 header.h File Reference

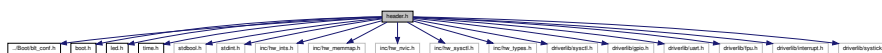
Generic header file.

```

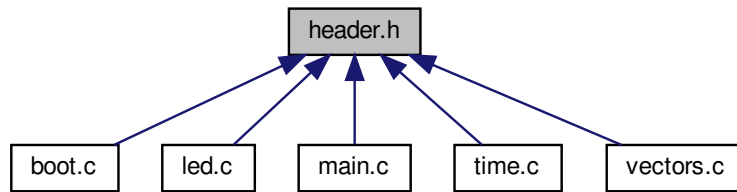
#include "../Boot/blt_conf.h"
#include "boot.h"
#include "led.h"
#include "time.h"
#include <stdbool.h>
#include <stdint.h>
#include "inc/hw_ints.h"
#include "inc/hw_memmap.h"
#include "inc/hw_nvic.h"
#include "inc/hw_sysctl.h"
#include "inc/hw_types.h"
#include "driverlib/sysctl.h"
#include "driverlib/gpio.h"
#include "driverlib/uart.h"
#include "driverlib/fpu.h"
#include "driverlib/interrupt.h"
#include "driverlib/systick.h"

```

Include dependency graph for ARMCM4_TM4C_DK_TM4C123G_IAR/Prog/header.h:



This graph shows which files directly or indirectly include this file:



7.531.1 Detailed Description

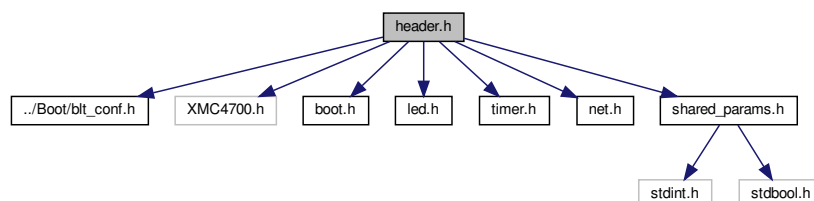
Generic header file.

7.532 header.h File Reference

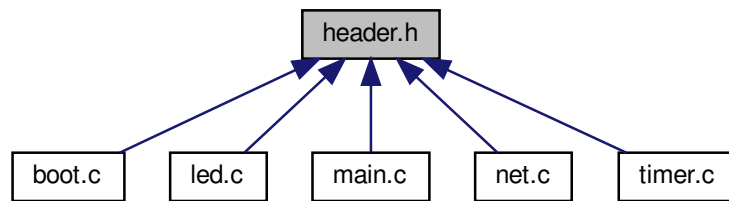
Generic header file.

```
#include "../Boot/blt_conf.h"
#include "XMC4700.h"
#include "boot.h"
#include "led.h"
#include "timer.h"
#include "net.h"
#include "shared_params.h"
```

Include dependency graph for ARMCM4_XMC4_XMC4700_Relax_Kit_GCC/Prog/header.h:



This graph shows which files directly or indirectly include this file:



7.532.1 Detailed Description

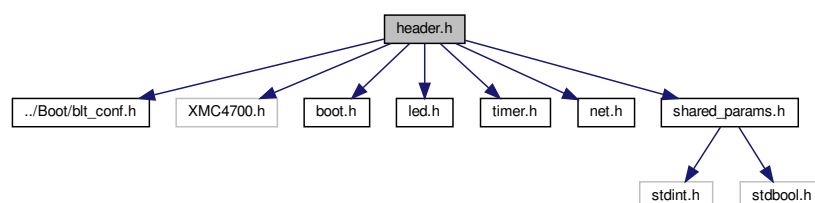
Generic header file.

7.533 header.h File Reference

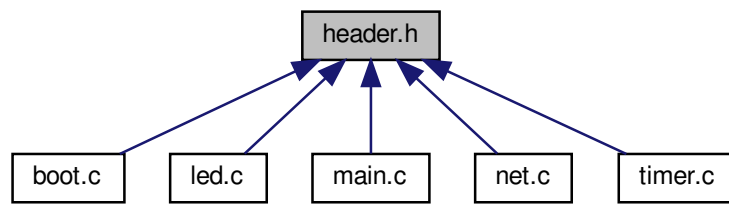
Generic header file.

```
#include "../Boot/blt_conf.h"
#include "XMC4700.h"
#include "boot.h"
#include "led.h"
#include "timer.h"
#include "net.h"
#include "shared_params.h"
```

Include dependency graph for `ARMCM4_XMC4_XMC4700_Relax_Kit_IAR/Prog/header.h`:



This graph shows which files directly or indirectly include this file:



7.533.1 Detailed Description

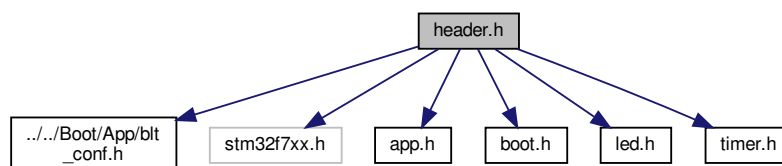
Generic header file.

7.534 header.h File Reference

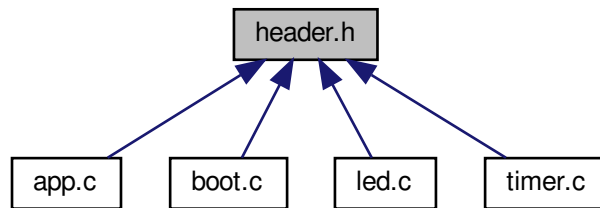
Generic header file.

```
#include "../..../Boot/App/blt_conf.h"
#include "stm32f7xx.h"
#include "app.h"
#include "boot.h"
#include "led.h"
#include "timer.h"
```

Include dependency graph for ARMCM7_STM32F7_Nucleo_F746ZG_CubeIDE/Prog/App/header.h:



This graph shows which files directly or indirectly include this file:



7.534.1 Detailed Description

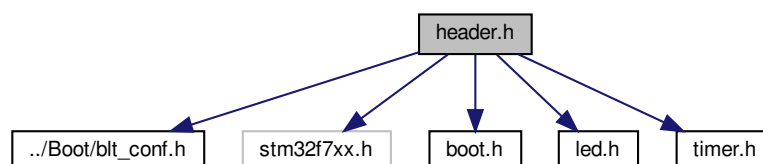
Generic header file.

7.535 header.h File Reference

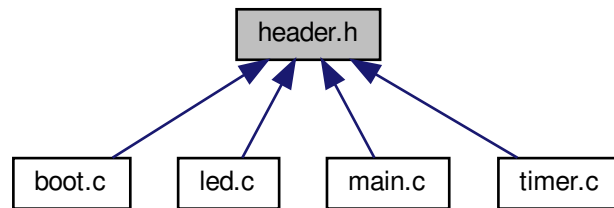
Generic header file.

```
#include "../Boot/blt_conf.h"
#include "stm32f7xx.h"
#include "boot.h"
#include "led.h"
#include "timer.h"
```

Include dependency graph for `ARMCM7_STM32F7_Nucleo_F746ZG_GCC/Prog/header.h`:



This graph shows which files directly or indirectly include this file:



7.535.1 Detailed Description

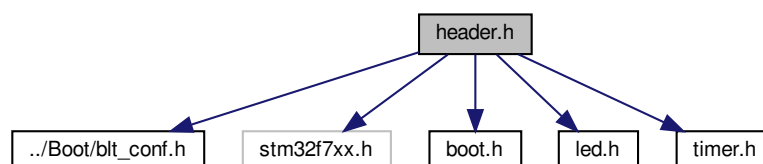
Generic header file.

7.536 header.h File Reference

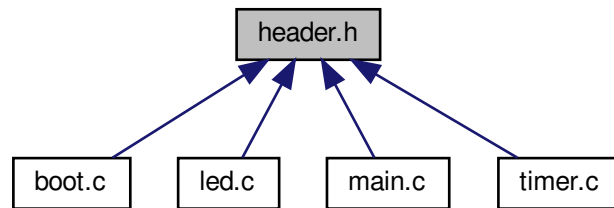
Generic header file.

```
#include "../Boot/blt_conf.h"
#include "stm32f7xx.h"
#include "boot.h"
#include "led.h"
#include "timer.h"
```

Include dependency graph for ARMC7_STM32F7_Nucleo_F746ZG_IAR/Prog/header.h:



This graph shows which files directly or indirectly include this file:



7.536.1 Detailed Description

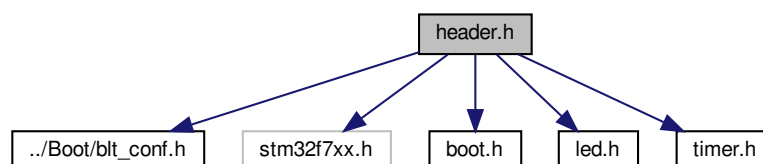
Generic header file.

7.537 header.h File Reference

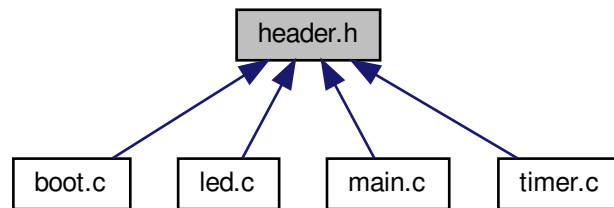
Generic header file.

```
#include "../Boot/blt_conf.h"
#include "stm32f7xx.h"
#include "boot.h"
#include "led.h"
#include "timer.h"
```

Include dependency graph for ARMC7_STM32F7_Nucleo_F746ZG_Keil/Prog/header.h:



This graph shows which files directly or indirectly include this file:



7.537.1 Detailed Description

Generic header file.

7.538 header.h File Reference

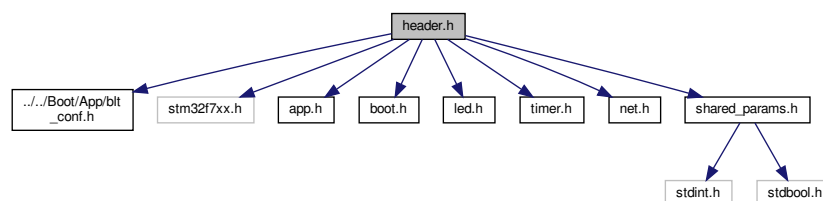
Generic header file.

```

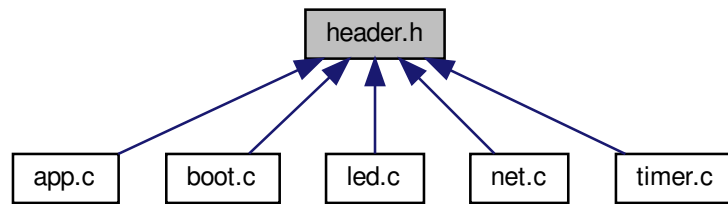
#include "../Boot/App/blt_conf.h"
#include "stm32f7xx.h"
#include "app.h"
#include "boot.h"
#include "led.h"
#include "timer.h"
#include "net.h"
#include "shared_params.h"

```

Include dependency graph for ARMCM7_STM32F7_Nucleo_F767ZI_CubeIDE/Prog/App/header.h:



This graph shows which files directly or indirectly include this file:



7.538.1 Detailed Description

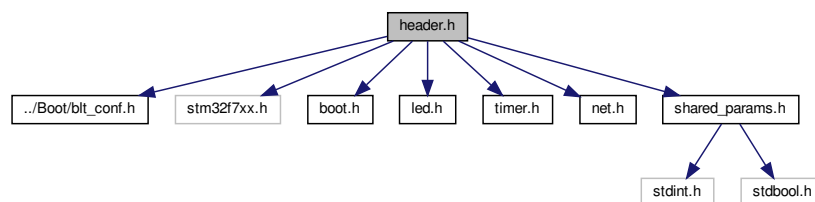
Generic header file.

7.539 `header.h` File Reference

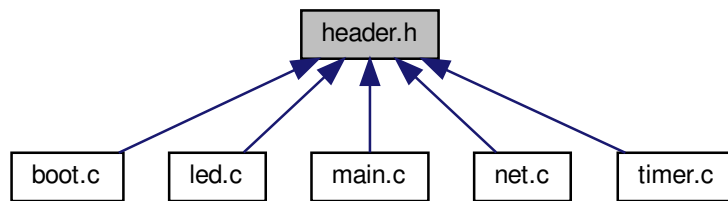
Generic header file.

```
#include "../Boot/blt_conf.h"
#include "stm32f7xx.h"
#include "boot.h"
#include "led.h"
#include "timer.h"
#include "net.h"
#include "shared_params.h"
```

Include dependency graph for `ARMCM7_STM32F7_Nucleo_F767ZI_GCC/Prog/header.h`:



This graph shows which files directly or indirectly include this file:



7.539.1 Detailed Description

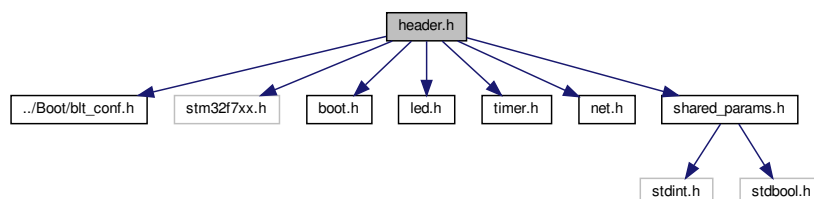
Generic header file.

7.540 header.h File Reference

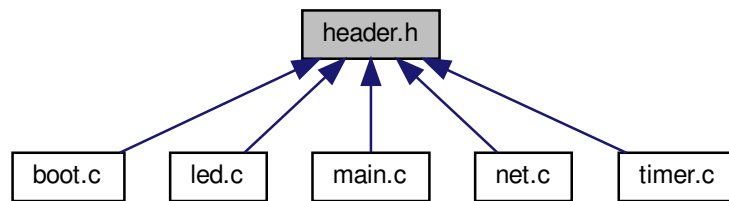
Generic header file.

```
#include "../Boot/blt_conf.h"
#include "stm32f7xx.h"
#include "boot.h"
#include "led.h"
#include "timer.h"
#include "net.h"
#include "shared_params.h"
```

Include dependency graph for `ARMCM7_STM32F7_Nucleo_F767ZI_IAR/Prog/header.h`:



This graph shows which files directly or indirectly include this file:



7.540.1 Detailed Description

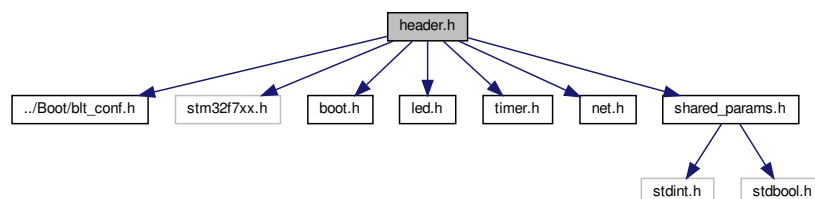
Generic header file.

7.541 header.h File Reference

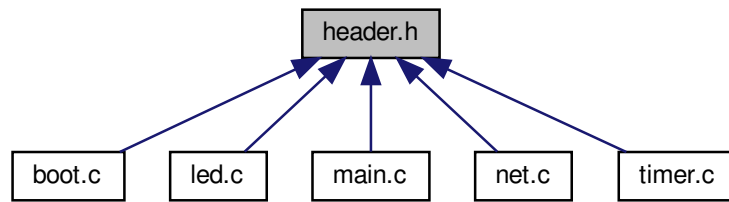
Generic header file.

```
#include "../Boot/blt_conf.h"
#include "stm32f7xx.h"
#include "boot.h"
#include "led.h"
#include "timer.h"
#include "net.h"
#include "shared_params.h"
```

Include dependency graph for `ARMCM7_STM32F7_Nucleo_F767ZI_Keil/Prog/header.h`:



This graph shows which files directly or indirectly include this file:



7.541.1 Detailed Description

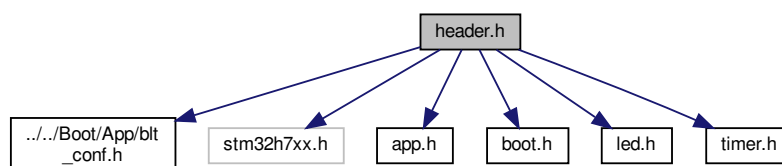
Generic header file.

7.542 header.h File Reference

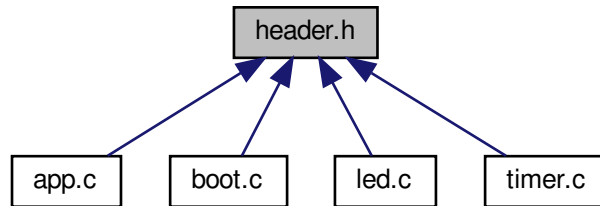
Generic header file.

```
#include "../..../Boot/App/blt_conf.h"
#include "stm32h7xx.h"
#include "app.h"
#include "boot.h"
#include "led.h"
#include "timer.h"
```

Include dependency graph for ARMCM7_STM32H7_Nucleo_H743ZI_CubeIDE/Prog/App/header.h:



This graph shows which files directly or indirectly include this file:



7.542.1 Detailed Description

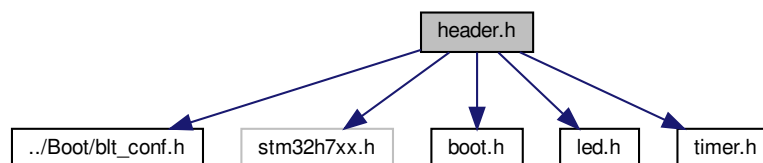
Generic header file.

7.543 header.h File Reference

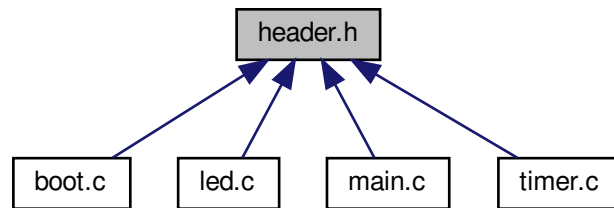
Generic header file.

```
#include "../Boot/blt_conf.h"
#include "stm32h7xx.h"
#include "boot.h"
#include "led.h"
#include "timer.h"
```

Include dependency graph for `ARMCM7_STM32H7_Nucleo_H743ZI_GCC/Prog/header.h`:



This graph shows which files directly or indirectly include this file:



7.543.1 Detailed Description

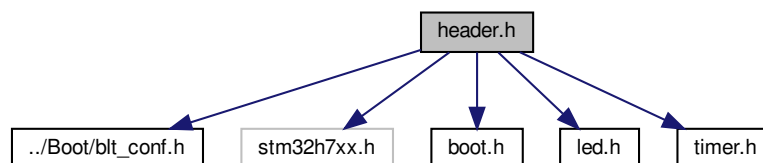
Generic header file.

7.544 header.h File Reference

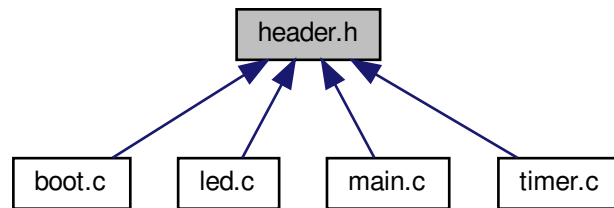
Generic header file.

```
#include "../Boot/blt_conf.h"
#include "stm32h7xx.h"
#include "boot.h"
#include "led.h"
#include "timer.h"
```

Include dependency graph for ARMC7_STM32H7_Nucleo_H743ZI_IAR/Prog/header.h:



This graph shows which files directly or indirectly include this file:



7.544.1 Detailed Description

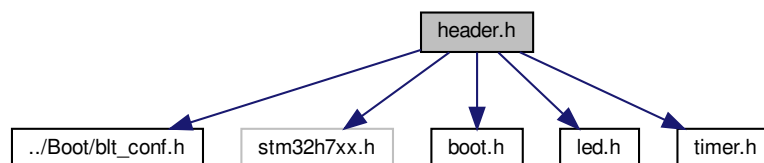
Generic header file.

7.545 header.h File Reference

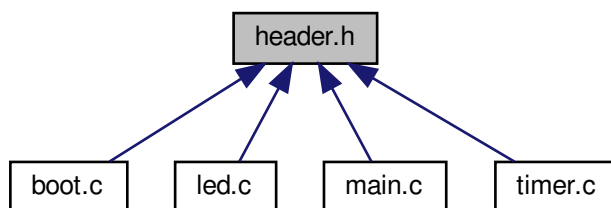
Generic header file.

```
#include "../Boot/blt_conf.h"
#include "stm32h7xx.h"
#include "boot.h"
#include "led.h"
#include "timer.h"
```

Include dependency graph for ARMC7_STM32H7_Nucleo_H743ZI_Keil/Prog/header.h:



This graph shows which files directly or indirectly include this file:



7.545.1 Detailed Description

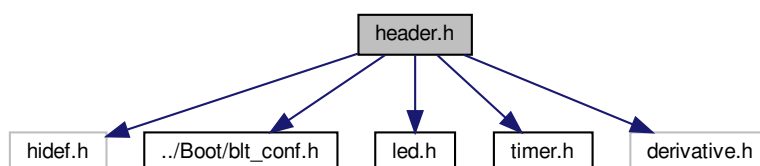
Generic header file.

7.546 header.h File Reference

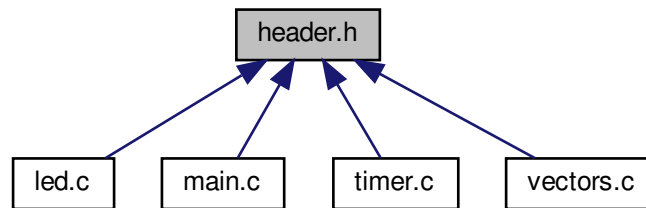
Generic header file.

```
#include <hidef.h>
#include "../Boot/blt_conf.h"
#include "boot.h"
#include "led.h"
#include "timer.h"
#include "derivative.h"
```

Include dependency graph for HCS12_DevKit_S12G128_CodeWarrior/Prog/header.h:



This graph shows which files directly or indirectly include this file:



Macros

- `#define BDM_DEBUGGING_ENABLED (0)`

Configuration switch to enable programming and debugging with a BDM interface.

7.546.1 Detailed Description

Generic header file.

7.546.2 Macro Definition Documentation

7.546.2.1 BDM_DEBUGGING_ENABLED

```
#define BDM_DEBUGGING_ENABLED (0)
```

Configuration switch to enable programming and debugging with a BDM interface.

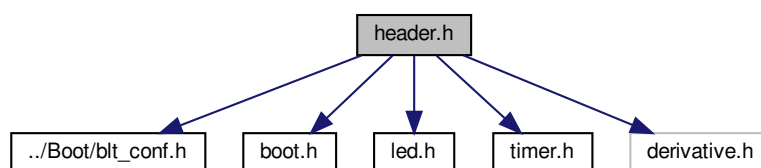
To configure the program for downloading with the OpenBLT bootloader, set this value to 0. This is typically done for release versions. If support for programming and debugging with a BDM debugger interface is desired during development, then set this value to 1.

7.547 header.h File Reference

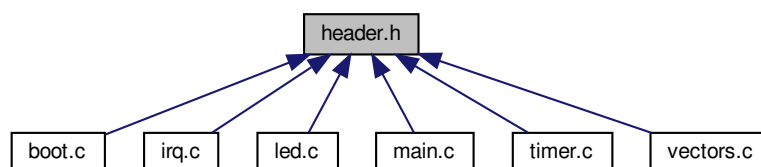
Generic header file.

```
#include "../Boot/blt_conf.h"
#include "boot.h"
#include "led.h"
#include "timer.h"
#include "derivative.h"
```

Include dependency graph for HCS12_Evbplus_Dragon12p_CodeWarrior/Prog/header.h:



This graph shows which files directly or indirectly include this file:



Macros

- `#define BDM_DEBUGGING_ENABLED (0)`
Configuration switch to enable programming and debugging with a BDM interface.

7.547.1 Detailed Description

Generic header file.

7.547.2 Macro Definition Documentation

7.547.2.1 BDM_DEBUGGING_ENABLED

```
#define BDM_DEBUGGING_ENABLED (0)
```

Configuration switch to enable programming and debugging with a BDM interface.

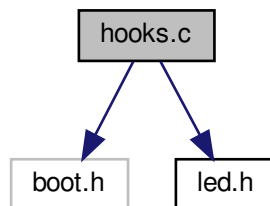
To configure the program for downloading with the OpenBLT bootloader, set this value to 0. This is typically done for release versions. If support for programming and debugging with a BDM debugger interface is desired during development, then set this value to 1.

7.548 hooks.c File Reference

Bootloader callback source file.

```
#include "boot.h"
#include "led.h"
```

Include dependency graph for `_template/Boot/hooks.c`:



Functions

- void [BackDoorInitHook](#) (void)
Initializes the backdoor entry option.
- [blt_bool BackDoorEntryHook](#) (void)
Checks if a backdoor entry is requested.
- [blt_bool CpuUserProgramStartHook](#) (void)
Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.
- void [CopInitHook](#) (void)
Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.
- void [CopServiceHook](#) (void)
Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.
- void [NvmInitHook](#) (void)
Callback that gets called at the start of the internal NVM driver initialization routine.
- void [NvmReinitHook](#) (void)

Callback that gets called at the start of a firmware update to reinitialize the NVM driver.

- `blt_int8u NvmWriteHook (blt_addr addr, blt_int32u len, blt_int8u *data)`

Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then `BLT_NVM_NOT_IN_RANGE` must be returned to indicate that the data hasn't been written yet.

- `blt_int8u NvmEraseHook (blt_addr addr, blt_int32u len)`

Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then `BLT_NVM_NOT_IN_RANGE` must be returned to indicate that the memory hasn't been erased yet.

- `blt_bool NvmDoneHook (void)`

Callback that gets called at the end of the NVM programming session.

- `blt_int8u XcpGetSeedHook (blt_int8u resource, blt_int8u *seed)`

Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the `GET_SEED` command.

- `blt_int8u XcpVerifyKeyHook (blt_int8u resource, blt_int8u *key, blt_int8u len)`

Called by the `UNLOCK` command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

7.548.1 Detailed Description

Bootloader callback source file.

7.548.2 Function Documentation

7.548.2.1 BackDoorEntryHook()

```
blt_bool BackDoorEntryHook (
    void )
```

Checks if a backdoor entry is requested.

Returns

`BLT_TRUE` if the backdoor entry is requested, `BLT_FALSE` otherwise.

7.548.2.2 BackDoorInitHook()

```
void BackDoorInitHook (
    void )
```

Initializes the backdoor entry option.

Returns

none.

7.548.2.3 CopInitHook()

```
void CopInitHook (
    void )
```

Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.

Returns

none.

7.548.2.4 CopServiceHook()

```
void CopServiceHook (
    void )
```

Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.

Returns

none.

7.548.2.5 CpuUserProgramStartHook()

```
blt_bool CpuUserProgramStartHook (
    void )
```

Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.

Returns

BLT_TRUE if it is okay to start the user program, BLT_FALSE to keep keep the bootloader active.

7.548.2.6 NvmDoneHook()

```
blt_bool NvmDoneHook (
    void )
```

Callback that gets called at the end of the NVM programming session.

Returns

BLT_TRUE is successful, BLT_FALSE otherwise.

7.548.2.7 NvmEraseHook()

```
blt_int8u NvmEraseHook (
    blt_addr addr,
    blt_int32u len )
```

Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the memory hasn't been erased yet.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.

Returns

BLT_NVM_OKAY if successful, BLT_NVM_NOT_IN_RANGE if the address is not within the supported memory range, or BLT_NVM_ERROR if the erase operation failed.

7.548.2.8 NvmInitHook()

```
void NvmInitHook (
    void )
```

Callback that gets called at the start of the internal NVM driver initialization routine.

Returns

none.

7.548.2.9 NvmReinitHook()

```
void NvmReinitHook (
    void )
```

Callback that gets called at the start of a firmware update to reinitialize the NVM driver.

Returns

none.

7.548.2.10 NvmWriteHook()

```
blt_int8u NvmWriteHook (
    blt_addr addr,
    blt_int32u len,
    blt_int8u * data )
```

Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the data hasn't been written yet.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.
<i>data</i>	Pointer to the data buffer.

Returns

BLT_NVM_OKAY if successful, BLT_NVM_NOT_IN_RANGE if the address is not within the supported memory range, or BLT_NVM_ERROR if the write operation failed.

7.548.2.11 XcpGetSeedHook()

```
blt_int8u XcpGetSeedHook (
    blt_int8u resource,
    blt_int8u * seed )
```

Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the GET_SEED command.

Parameters

<i>resource</i>	Resource that the seed is requested for (XCP_RES_XXX).
<i>seed</i>	Pointer to byte buffer where the seed will be stored.

Returns

Length of the seed in bytes.

7.548.2.12 XcpVerifyKeyHook()

```
blt_int8u XcpVerifyKeyHook (
    blt_int8u resource,
    blt_int8u * key,
    blt_int8u len )
```

Called by the UNLOCK command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

Parameters

<i>resource</i>	resource to unlock (XCP_RES_XXX).
<i>key</i>	pointer to the byte buffer holding the key.
<i>len</i>	length of the key in bytes.

Returns

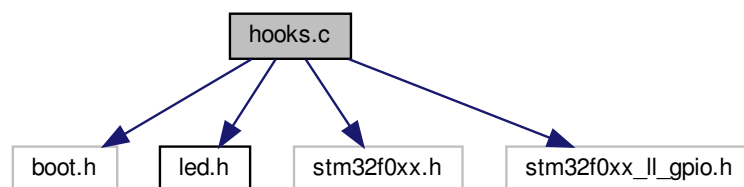
1 if the key was correct, 0 otherwise.

7.549 hooks.c File Reference

Bootloader callback source file.

```
#include "boot.h"
#include "led.h"
#include "stm32f0xx.h"
#include "stm32f0xx_ll_gpio.h"
```

Include dependency graph for ARMCM0_STM32F0_Discovery_STM32F051_CubeIDE/Boot/App/hooks.c:



Functions

- void [BackDoorInitHook](#) (void)
Initializes the backdoor entry option.
- [blt_bool](#) [BackDoorEntryHook](#) (void)
Checks if a backdoor entry is requested.
- [blt_bool](#) [CpuUserProgramStartHook](#) (void)
Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.
- void [CopInitHook](#) (void)
Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.
- void [CopServiceHook](#) (void)
Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.
- void [NvmInitHook](#) (void)
Callback that gets called at the start of the internal NVM driver initialization routine.
- void [NvmReinitHook](#) (void)
Callback that gets called at the start of a firmware update to reinitialize the NVM driver.
- [blt_int8u](#) [NvmWriteHook](#) ([blt_addr](#) addr, [blt_int32u](#) len, [blt_int8u](#) *data)
Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the data hasn't been written yet.
- [blt_int8u](#) [NvmEraseHook](#) ([blt_addr](#) addr, [blt_int32u](#) len)

Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then `BLT_NVM_NOT_IN_RANGE` must be returned to indicate that the memory hasn't been erased yet.

- `blt_bool NvmDoneHook` (void)

Callback that gets called at the end of the NVM programming session.

- `blt_int8u XcpGetSeedHook` (`blt_int8u` resource, `blt_int8u *seed`)

Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the `GET_SEED` command.

- `blt_int8u XcpVerifyKeyHook` (`blt_int8u` resource, `blt_int8u *key`, `blt_int8u len`)

Called by the `UNLOCK` command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

7.549.1 Detailed Description

Bootloader callback source file.

7.549.2 Function Documentation

7.549.2.1 BackDoorEntryHook()

```
blt_bool BackDoorEntryHook (
    void )
```

Checks if a backdoor entry is requested.

Returns

`BLT_TRUE` if the backdoor entry is requested, `BLT_FALSE` otherwise.

7.549.2.2 BackDoorInitHook()

```
void BackDoorInitHook (
    void )
```

Initializes the backdoor entry option.

Returns

none.

7.549.2.3 CopInitHook()

```
void CopInitHook (
    void )
```

Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.

Returns

none.

7.549.2.4 CopServiceHook()

```
void CopServiceHook (
    void )
```

Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.

Returns

none.

7.549.2.5 CpuUserProgramStartHook()

```
blt_bool CpuUserProgramStartHook (
    void )
```

Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.

Returns

BLT_TRUE if it is okay to start the user program, BLT_FALSE to keep keep the bootloader active.

7.549.2.6 NvmDoneHook()

```
blt_bool NvmDoneHook (
    void )
```

Callback that gets called at the end of the NVM programming session.

Returns

BLT_TRUE is successful, BLT_FALSE otherwise.

7.549.2.7 NvmEraseHook()

```
blt_int8u NvmEraseHook (
    blt_addr addr,
    blt_int32u len )
```

Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the memory hasn't been erased yet.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.

Returns

BLT_NVM_OKAY if successful, BLT_NVM_NOT_IN_RANGE if the address is not within the supported memory range, or BLT_NVM_ERROR if the erase operation failed.

7.549.2.8 NvmInitHook()

```
void NvmInitHook (
    void )
```

Callback that gets called at the start of the internal NVM driver initialization routine.

Returns

none.

7.549.2.9 NvmReinitHook()

```
void NvmReinitHook (
    void )
```

Callback that gets called at the start of a firmware update to reinitialize the NVM driver.

Returns

none.

7.549.2.10 NvmWriteHook()

```
blt_int8u NvmWriteHook (
    blt_addr addr,
    blt_int32u len,
    blt_int8u * data )
```

Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the data hasn't been written yet.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.
<i>data</i>	Pointer to the data buffer.

Returns

BLT_NVM_OKAY if successful, BLT_NVM_NOT_IN_RANGE if the address is not within the supported memory range, or BLT_NVM_ERROR if the write operation failed.

7.549.2.11 XcpGetSeedHook()

```
blt_int8u XcpGetSeedHook (
    blt_int8u resource,
    blt_int8u * seed )
```

Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the GET_SEED command.

Parameters

<i>resource</i>	Resource that the seed is requested for (XCP_RES_XXX).
<i>seed</i>	Pointer to byte buffer where the seed will be stored.

Returns

Length of the seed in bytes.

7.549.2.12 XcpVerifyKeyHook()

```
blt_int8u XcpVerifyKeyHook (
    blt_int8u resource,
    blt_int8u * key,
    blt_int8u len )
```

Called by the UNLOCK command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

Parameters

<i>resource</i>	resource to unlock (XCP_RES_XXX).
<i>key</i>	pointer to the byte buffer holding the key.
<i>len</i>	length of the key in bytes.

Returns

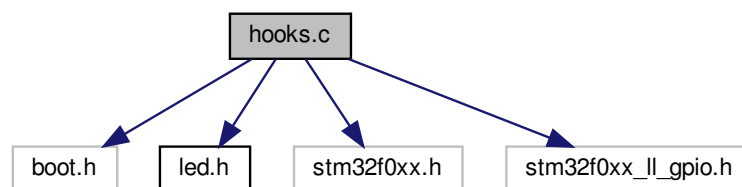
1 if the key was correct, 0 otherwise.

7.550 hooks.c File Reference

Bootloader callback source file.

```
#include "boot.h"
#include "led.h"
#include "stm32f0xx.h"
#include "stm32f0xx_ll_gpio.h"
```

Include dependency graph for ARMCM0_STM32F0_Discovery_STM32F051_GCC/Boot/hooks.c:



Functions

- void [BackDoorInitHook](#) (void)
Initializes the backdoor entry option.
- [blt_bool](#) [BackDoorEntryHook](#) (void)
Checks if a backdoor entry is requested.
- [blt_bool](#) [CpuUserProgramStartHook](#) (void)
Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.
- void [CopInitHook](#) (void)
Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.
- void [CopServiceHook](#) (void)
Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.
- void [NvmInitHook](#) (void)
Callback that gets called at the start of the internal NVM driver initialization routine.
- void [NvmReinitHook](#) (void)
Callback that gets called at the start of a firmware update to reinitialize the NVM driver.
- [blt_int8u](#) [NvmWriteHook](#) ([blt_addr](#) addr, [blt_int32u](#) len, [blt_int8u](#) *data)
Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the data hasn't been written yet.
- [blt_int8u](#) [NvmEraseHook](#) ([blt_addr](#) addr, [blt_int32u](#) len)

Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then `BLT_NVM_NOT_IN_RANGE` must be returned to indicate that the memory hasn't been erased yet.

- `blt_bool NvmDoneHook` (void)

Callback that gets called at the end of the NVM programming session.

- `blt_int8u XcpGetSeedHook` (`blt_int8u` resource, `blt_int8u *`seed)

Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the `GET_SEED` command.

- `blt_int8u XcpVerifyKeyHook` (`blt_int8u` resource, `blt_int8u *`key, `blt_int8u` len)

Called by the `UNLOCK` command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

7.550.1 Detailed Description

Bootloader callback source file.

7.550.2 Function Documentation

7.550.2.1 BackDoorEntryHook()

```
blt_bool BackDoorEntryHook (
    void )
```

Checks if a backdoor entry is requested.

Returns

`BLT_TRUE` if the backdoor entry is requested, `BLT_FALSE` otherwise.

7.550.2.2 BackDoorInitHook()

```
void BackDoorInitHook (
    void )
```

Initializes the backdoor entry option.

Returns

none.

7.550.2.3 CopInitHook()

```
void CopInitHook (
    void )
```

Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.

Returns

none.

7.550.2.4 CopServiceHook()

```
void CopServiceHook (
    void )
```

Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.

Returns

none.

7.550.2.5 CpuUserProgramStartHook()

```
blt_bool CpuUserProgramStartHook (
    void )
```

Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.

Returns

BLT_TRUE if it is okay to start the user program, BLT_FALSE to keep keep the bootloader active.

7.550.2.6 NvmDoneHook()

```
blt_bool NvmDoneHook (
    void )
```

Callback that gets called at the end of the NVM programming session.

Returns

BLT_TRUE is successful, BLT_FALSE otherwise.

7.550.2.7 NvmEraseHook()

```
blt_int8u NvmEraseHook (
    blt_addr addr,
    blt_int32u len )
```

Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the memory hasn't been erased yet.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.

Returns

BLT_NVM_OKAY if successful, BLT_NVM_NOT_IN_RANGE if the address is not within the supported memory range, or BLT_NVM_ERROR if the erase operation failed.

7.550.2.8 NvmInitHook()

```
void NvmInitHook (
    void )
```

Callback that gets called at the start of the internal NVM driver initialization routine.

Returns

none.

7.550.2.9 NvmReinitHook()

```
void NvmReinitHook (
    void )
```

Callback that gets called at the start of a firmware update to reinitialize the NVM driver.

Returns

none.

7.550.2.10 NvmWriteHook()

```
blt_int8u NvmWriteHook (
    blt_addr addr,
    blt_int32u len,
    blt_int8u * data )
```

Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the data hasn't been written yet.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.
<i>data</i>	Pointer to the data buffer.

Returns

BLT_NVM_OKAY if successful, BLT_NVM_NOT_IN_RANGE if the address is not within the supported memory range, or BLT_NVM_ERROR if the write operation failed.

7.550.2.11 XcpGetSeedHook()

```
blt_int8u XcpGetSeedHook (
    blt_int8u resource,
    blt_int8u * seed )
```

Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the GET_SEED command.

Parameters

<i>resource</i>	Resource that the seed is requested for (XCP_RES_XXX).
<i>seed</i>	Pointer to byte buffer where the seed will be stored.

Returns

Length of the seed in bytes.

7.550.2.12 XcpVerifyKeyHook()

```
blt_int8u XcpVerifyKeyHook (
    blt_int8u resource,
    blt_int8u * key,
    blt_int8u len )
```

Called by the UNLOCK command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

Parameters

<i>resource</i>	resource to unlock (XCP_RES_XXX).
<i>key</i>	pointer to the byte buffer holding the key.
<i>len</i>	length of the key in bytes.

Returns

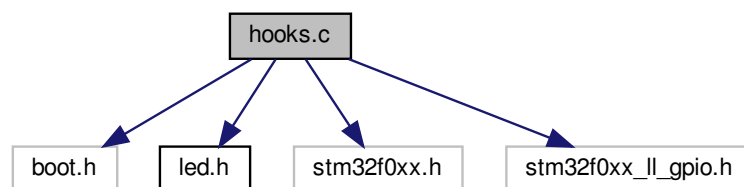
1 if the key was correct, 0 otherwise.

7.551 hooks.c File Reference

Bootloader callback source file.

```
#include "boot.h"
#include "led.h"
#include "stm32f0xx.h"
#include "stm32f0xx_ll_gpio.h"
```

Include dependency graph for ARMCM0_STM32F0_Discovery_STM32F051_IAR/Boot/hooks.c:



Functions

- void [BackDoorInitHook](#) (void)
Initializes the backdoor entry option.
- [blt_bool](#) [BackDoorEntryHook](#) (void)
Checks if a backdoor entry is requested.
- [blt_bool](#) [CpuUserProgramStartHook](#) (void)
Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.
- void [CopInitHook](#) (void)
Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.
- void [CopServiceHook](#) (void)
Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.
- void [NvmInitHook](#) (void)
Callback that gets called at the start of the internal NVM driver initialization routine.
- void [NvmReinitHook](#) (void)
Callback that gets called at the start of a firmware update to reinitialize the NVM driver.
- [blt_int8u](#) [NvmWriteHook](#) ([blt_addr](#) addr, [blt_int32u](#) len, [blt_int8u](#) *data)
Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the data hasn't been written yet.
- [blt_int8u](#) [NvmEraseHook](#) ([blt_addr](#) addr, [blt_int32u](#) len)

Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then `BLT_NVM_NOT_IN_RANGE` must be returned to indicate that the memory hasn't been erased yet.

- `blt_bool NvmDoneHook` (void)

Callback that gets called at the end of the NVM programming session.

- `blt_int8u XcpGetSeedHook` (`blt_int8u` resource, `blt_int8u *`seed)

Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the `GET_SEED` command.

- `blt_int8u XcpVerifyKeyHook` (`blt_int8u` resource, `blt_int8u *`key, `blt_int8u` len)

Called by the `UNLOCK` command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

7.551.1 Detailed Description

Bootloader callback source file.

7.551.2 Function Documentation

7.551.2.1 BackDoorEntryHook()

```
blt_bool BackDoorEntryHook (
    void )
```

Checks if a backdoor entry is requested.

Returns

`BLT_TRUE` if the backdoor entry is requested, `BLT_FALSE` otherwise.

7.551.2.2 BackDoorInitHook()

```
void BackDoorInitHook (
    void )
```

Initializes the backdoor entry option.

Returns

none.

7.551.2.3 CopInitHook()

```
void CopInitHook (
    void )
```

Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.

Returns

none.

7.551.2.4 CopServiceHook()

```
void CopServiceHook (
    void )
```

Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.

Returns

none.

7.551.2.5 CpuUserProgramStartHook()

```
blt_bool CpuUserProgramStartHook (
    void )
```

Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.

Returns

BLT_TRUE if it is okay to start the user program, BLT_FALSE to keep keep the bootloader active.

7.551.2.6 NvmDoneHook()

```
blt_bool NvmDoneHook (
    void )
```

Callback that gets called at the end of the NVM programming session.

Returns

BLT_TRUE is successful, BLT_FALSE otherwise.

7.551.2.7 NvmEraseHook()

```
blt_int8u NvmEraseHook (
    blt_addr addr,
    blt_int32u len )
```

Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the memory hasn't been erased yet.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.

Returns

BLT_NVM_OKAY if successful, BLT_NVM_NOT_IN_RANGE if the address is not within the supported memory range, or BLT_NVM_ERROR if the erase operation failed.

7.551.2.8 NvmInitHook()

```
void NvmInitHook (
    void )
```

Callback that gets called at the start of the internal NVM driver initialization routine.

Returns

none.

7.551.2.9 NvmReinitHook()

```
void NvmReinitHook (
    void )
```

Callback that gets called at the start of a firmware update to reinitialize the NVM driver.

Returns

none.

7.551.2.10 NvmWriteHook()

```
blt_int8u NvmWriteHook (
    blt_addr addr,
    blt_int32u len,
    blt_int8u * data )
```

Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the data hasn't been written yet.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.
<i>data</i>	Pointer to the data buffer.

Returns

BLT_NVM_OKAY if successful, BLT_NVM_NOT_IN_RANGE if the address is not within the supported memory range, or BLT_NVM_ERROR if the write operation failed.

7.551.2.11 XcpGetSeedHook()

```
blt_int8u XcpGetSeedHook (
    blt_int8u resource,
    blt_int8u * seed )
```

Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the GET_SEED command.

Parameters

<i>resource</i>	Resource that the seed is requested for (XCP_RES_XXX).
<i>seed</i>	Pointer to byte buffer where the seed will be stored.

Returns

Length of the seed in bytes.

7.551.2.12 XcpVerifyKeyHook()

```
blt_int8u XcpVerifyKeyHook (
    blt_int8u resource,
    blt_int8u * key,
    blt_int8u len )
```

Called by the UNLOCK command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

Parameters

<i>resource</i>	resource to unlock (XCP_RES_XXX).
<i>key</i>	pointer to the byte buffer holding the key.
<i>len</i>	length of the key in bytes.

Returns

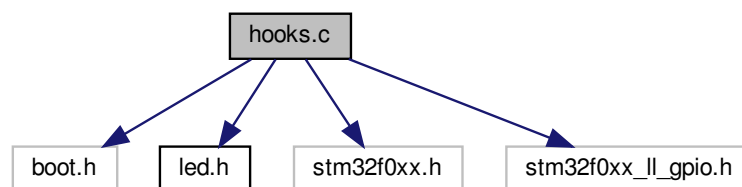
1 if the key was correct, 0 otherwise.

7.552 hooks.c File Reference

Bootloader callback source file.

```
#include "boot.h"
#include "led.h"
#include "stm32f0xx.h"
#include "stm32f0xx_ll_gpio.h"
```

Include dependency graph for ARMCM0_STM32F0_Discovery_STM32F051_Keil/Boot/hooks.c:



Functions

- void [BackDoorInitHook](#) (void)
Initializes the backdoor entry option.
- [blt_bool](#) [BackDoorEntryHook](#) (void)
Checks if a backdoor entry is requested.
- [blt_bool](#) [CpuUserProgramStartHook](#) (void)
Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.
- void [CopInitHook](#) (void)
Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.
- void [CopServiceHook](#) (void)
Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.
- void [NvmInitHook](#) (void)
Callback that gets called at the start of the internal NVM driver initialization routine.
- void [NvmReinitHook](#) (void)
Callback that gets called at the start of a firmware update to reinitialize the NVM driver.
- [blt_int8u](#) [NvmWriteHook](#) ([blt_addr](#) addr, [blt_int32u](#) len, [blt_int8u](#) *data)
Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the data hasn't been written yet.
- [blt_int8u](#) [NvmEraseHook](#) ([blt_addr](#) addr, [blt_int32u](#) len)

Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then `BLT_NVM_NOT_IN_RANGE` must be returned to indicate that the memory hasn't been erased yet.

- `blt_bool NvmDoneHook` (void)

Callback that gets called at the end of the NVM programming session.

- `blt_int8u XcpGetSeedHook` (`blt_int8u` resource, `blt_int8u *seed`)

Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the `GET_SEED` command.

- `blt_int8u XcpVerifyKeyHook` (`blt_int8u` resource, `blt_int8u *key`, `blt_int8u len`)

Called by the `UNLOCK` command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

7.552.1 Detailed Description

Bootloader callback source file.

7.552.2 Function Documentation

7.552.2.1 BackDoorEntryHook()

```
blt_bool BackDoorEntryHook (  
    void )
```

Checks if a backdoor entry is requested.

Returns

`BLT_TRUE` if the backdoor entry is requested, `BLT_FALSE` otherwise.

7.552.2.2 BackDoorInitHook()

```
void BackDoorInitHook (  
    void )
```

Initializes the backdoor entry option.

Returns

none.

7.552.2.3 CopInitHook()

```
void CopInitHook (  
    void )
```

Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.

Returns

none.

7.552.2.4 CopServiceHook()

```
void CopServiceHook (  
    void )
```

Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.

Returns

none.

7.552.2.5 CpuUserProgramStartHook()

```
blt_bool CpuUserProgramStartHook (  
    void )
```

Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.

Returns

BLT_TRUE if it is okay to start the user program, BLT_FALSE to keep keep the bootloader active.

7.552.2.6 NvmDoneHook()

```
blt_bool NvmDoneHook (  
    void )
```

Callback that gets called at the end of the NVM programming session.

Returns

BLT_TRUE is successful, BLT_FALSE otherwise.

7.552.2.7 NvmEraseHook()

```
blt_int8u NvmEraseHook (
    blt_addr addr,
    blt_int32u len )
```

Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the memory hasn't been erased yet.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.

Returns

BLT_NVM_OKAY if successful, BLT_NVM_NOT_IN_RANGE if the address is not within the supported memory range, or BLT_NVM_ERROR if the erase operation failed.

7.552.2.8 NvmInitHook()

```
void NvmInitHook (
    void )
```

Callback that gets called at the start of the internal NVM driver initialization routine.

Returns

none.

7.552.2.9 NvmReinitHook()

```
void NvmReinitHook (
    void )
```

Callback that gets called at the start of a firmware update to reinitialize the NVM driver.

Returns

none.

7.552.2.10 NvmWriteHook()

```
blt_int8u NvmWriteHook (
    blt_addr addr,
    blt_int32u len,
    blt_int8u * data )
```

Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the data hasn't been written yet.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.
<i>data</i>	Pointer to the data buffer.

Returns

BLT_NVM_OKAY if successful, BLT_NVM_NOT_IN_RANGE if the address is not within the supported memory range, or BLT_NVM_ERROR if the write operation failed.

7.552.2.11 XcpGetSeedHook()

```
blt_int8u XcpGetSeedHook (
    blt_int8u resource,
    blt_int8u * seed )
```

Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the GET_SEED command.

Parameters

<i>resource</i>	Resource that the seed is requested for (XCP_RES_XXX).
<i>seed</i>	Pointer to byte buffer where the seed will be stored.

Returns

Length of the seed in bytes.

7.552.2.12 XcpVerifyKeyHook()

```
blt_int8u XcpVerifyKeyHook (
    blt_int8u resource,
    blt_int8u * key,
    blt_int8u len )
```

Called by the UNLOCK command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

Parameters

<i>resource</i>	resource to unlock (XCP_RES_XXX).
<i>key</i>	pointer to the byte buffer holding the key.
<i>len</i>	length of the key in bytes.

Returns

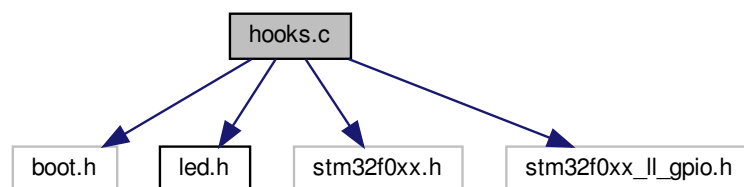
1 if the key was correct, 0 otherwise.

7.553 hooks.c File Reference

Bootloader callback source file.

```
#include "boot.h"
#include "led.h"
#include "stm32f0xx.h"
#include "stm32f0xx_ll_gpio.h"
```

Include dependency graph for ARMCM0_STM32F0_Nucleo_F091RC_CubeIDE/Boot/App/hooks.c:



Functions

- void [BackDoorInitHook](#) (void)
Initializes the backdoor entry option.
- [blt_bool](#) [BackDoorEntryHook](#) (void)
Checks if a backdoor entry is requested.
- [blt_bool](#) [CpuUserProgramStartHook](#) (void)
Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.
- void [CopInitHook](#) (void)
Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.
- void [CopServiceHook](#) (void)
Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.
- void [NvmInitHook](#) (void)
Callback that gets called at the start of the internal NVM driver initialization routine.
- void [NvmReinitHook](#) (void)
Callback that gets called at the start of a firmware update to reinitialize the NVM driver.
- [blt_int8u](#) [NvmWriteHook](#) ([blt_addr](#) addr, [blt_int32u](#) len, [blt_int8u](#) *data)
Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the data hasn't been written yet.
- [blt_int8u](#) [NvmEraseHook](#) ([blt_addr](#) addr, [blt_int32u](#) len)

Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then `BLT_NVM_NOT_IN_RANGE` must be returned to indicate that the memory hasn't been erased yet.

- `blt_bool NvmDoneHook` (void)

Callback that gets called at the end of the NVM programming session.

- `blt_int8u XcpGetSeedHook` (`blt_int8u` resource, `blt_int8u *seed`)

Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the `GET_SEED` command.

- `blt_int8u XcpVerifyKeyHook` (`blt_int8u` resource, `blt_int8u *key`, `blt_int8u len`)

Called by the `UNLOCK` command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

7.553.1 Detailed Description

Bootloader callback source file.

7.553.2 Function Documentation

7.553.2.1 BackDoorEntryHook()

```
blt_bool BackDoorEntryHook (
    void )
```

Checks if a backdoor entry is requested.

Returns

`BLT_TRUE` if the backdoor entry is requested, `BLT_FALSE` otherwise.

7.553.2.2 BackDoorInitHook()

```
void BackDoorInitHook (
    void )
```

Initializes the backdoor entry option.

Returns

none.

7.553.2.3 CopInitHook()

```
void CopInitHook (
    void )
```

Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.

Returns

none.

7.553.2.4 CopServiceHook()

```
void CopServiceHook (
    void )
```

Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.

Returns

none.

7.553.2.5 CpuUserProgramStartHook()

```
blt_bool CpuUserProgramStartHook (
    void )
```

Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.

Returns

BLT_TRUE if it is okay to start the user program, BLT_FALSE to keep keep the bootloader active.

7.553.2.6 NvmDoneHook()

```
blt_bool NvmDoneHook (
    void )
```

Callback that gets called at the end of the NVM programming session.

Returns

BLT_TRUE is successful, BLT_FALSE otherwise.

7.553.2.7 NvmEraseHook()

```
blt_int8u NvmEraseHook (
    blt_addr addr,
    blt_int32u len )
```

Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the memory hasn't been erased yet.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.

Returns

BLT_NVM_OKAY if successful, BLT_NVM_NOT_IN_RANGE if the address is not within the supported memory range, or BLT_NVM_ERROR if the erase operation failed.

7.553.2.8 NvmInitHook()

```
void NvmInitHook (
    void )
```

Callback that gets called at the start of the internal NVM driver initialization routine.

Returns

none.

7.553.2.9 NvmReinitHook()

```
void NvmReinitHook (
    void )
```

Callback that gets called at the start of a firmware update to reinitialize the NVM driver.

Returns

none.

7.553.2.10 NvmWriteHook()

```
blt_int8u NvmWriteHook (
    blt_addr addr,
    blt_int32u len,
    blt_int8u * data )
```

Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the data hasn't been written yet.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.
<i>data</i>	Pointer to the data buffer.

Returns

BLT_NVM_OKAY if successful, BLT_NVM_NOT_IN_RANGE if the address is not within the supported memory range, or BLT_NVM_ERROR if the write operation failed.

7.553.2.11 XcpGetSeedHook()

```
blt_int8u XcpGetSeedHook (
    blt_int8u resource,
    blt_int8u * seed )
```

Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the GET_SEED command.

Parameters

<i>resource</i>	Resource that the seed is requested for (XCP_RES_XXX).
<i>seed</i>	Pointer to byte buffer where the seed will be stored.

Returns

Length of the seed in bytes.

7.553.2.12 XcpVerifyKeyHook()

```
blt_int8u XcpVerifyKeyHook (
    blt_int8u resource,
    blt_int8u * key,
    blt_int8u len )
```

Called by the UNLOCK command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

Parameters

<i>resource</i>	resource to unlock (XCP_RES_XXX).
<i>key</i>	pointer to the byte buffer holding the key.
<i>len</i>	length of the key in bytes.

Returns

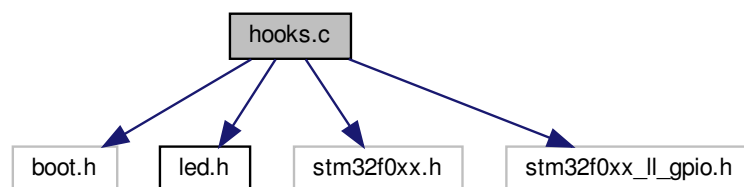
1 if the key was correct, 0 otherwise.

7.554 hooks.c File Reference

Bootloader callback source file.

```
#include "boot.h"
#include "led.h"
#include "stm32f0xx.h"
#include "stm32f0xx_ll_gpio.h"
```

Include dependency graph for ARMCM0_STM32F0_Nucleo_F091RC_GCC/Boot/hooks.c:



Functions

- void [BackDoorInitHook](#) (void)
Initializes the backdoor entry option.
- [blt_bool](#) [BackDoorEntryHook](#) (void)
Checks if a backdoor entry is requested.
- [blt_bool](#) [CpuUserProgramStartHook](#) (void)
Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.
- void [CopInitHook](#) (void)
Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.
- void [CopServiceHook](#) (void)
Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.
- void [NvmInitHook](#) (void)
Callback that gets called at the start of the internal NVM driver initialization routine.
- void [NvmReinitHook](#) (void)
Callback that gets called at the start of a firmware update to reinitialize the NVM driver.
- [blt_int8u](#) [NvmWriteHook](#) ([blt_addr](#) addr, [blt_int32u](#) len, [blt_int8u](#) *data)
Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the data hasn't been written yet.
- [blt_int8u](#) [NvmEraseHook](#) ([blt_addr](#) addr, [blt_int32u](#) len)

Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then `BLT_NVM_NOT_IN_RANGE` must be returned to indicate that the memory hasn't been erased yet.

- `blt_bool NvmDoneHook` (void)

Callback that gets called at the end of the NVM programming session.

- `blt_int8u XcpGetSeedHook` (`blt_int8u` resource, `blt_int8u *`seed)

Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the `GET_SEED` command.

- `blt_int8u XcpVerifyKeyHook` (`blt_int8u` resource, `blt_int8u *`key, `blt_int8u` len)

Called by the `UNLOCK` command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

7.554.1 Detailed Description

Bootloader callback source file.

7.554.2 Function Documentation

7.554.2.1 BackDoorEntryHook()

```
blt_bool BackDoorEntryHook (
    void )
```

Checks if a backdoor entry is requested.

Returns

`BLT_TRUE` if the backdoor entry is requested, `BLT_FALSE` otherwise.

7.554.2.2 BackDoorInitHook()

```
void BackDoorInitHook (
    void )
```

Initializes the backdoor entry option.

Returns

none.

7.554.2.3 CopInitHook()

```
void CopInitHook (
    void )
```

Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.

Returns

none.

7.554.2.4 CopServiceHook()

```
void CopServiceHook (
    void )
```

Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.

Returns

none.

7.554.2.5 CpuUserProgramStartHook()

```
blt_bool CpuUserProgramStartHook (
    void )
```

Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.

Returns

BLT_TRUE if it is okay to start the user program, BLT_FALSE to keep keep the bootloader active.

7.554.2.6 NvmDoneHook()

```
blt_bool NvmDoneHook (
    void )
```

Callback that gets called at the end of the NVM programming session.

Returns

BLT_TRUE is successful, BLT_FALSE otherwise.

7.554.2.7 NvmEraseHook()

```
blt_int8u NvmEraseHook (
    blt_addr addr,
    blt_int32u len )
```

Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the memory hasn't been erased yet.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.

Returns

BLT_NVM_OKAY if successful, BLT_NVM_NOT_IN_RANGE if the address is not within the supported memory range, or BLT_NVM_ERROR if the erase operation failed.

7.554.2.8 NvmInitHook()

```
void NvmInitHook (
    void )
```

Callback that gets called at the start of the internal NVM driver initialization routine.

Returns

none.

7.554.2.9 NvmReinitHook()

```
void NvmReinitHook (
    void )
```

Callback that gets called at the start of a firmware update to reinitialize the NVM driver.

Returns

none.

7.554.2.10 NvmWriteHook()

```
blt_int8u NvmWriteHook (
    blt_addr addr,
    blt_int32u len,
    blt_int8u * data )
```

Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the data hasn't been written yet.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.
<i>data</i>	Pointer to the data buffer.

Returns

BLT_NVM_OKAY if successful, BLT_NVM_NOT_IN_RANGE if the address is not within the supported memory range, or BLT_NVM_ERROR if the write operation failed.

7.554.2.11 XcpGetSeedHook()

```
blt_int8u XcpGetSeedHook (
    blt_int8u resource,
    blt_int8u * seed )
```

Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the GET_SEED command.

Parameters

<i>resource</i>	Resource that the seed is requested for (XCP_RES_XXX).
<i>seed</i>	Pointer to byte buffer where the seed will be stored.

Returns

Length of the seed in bytes.

7.554.2.12 XcpVerifyKeyHook()

```
blt_int8u XcpVerifyKeyHook (
    blt_int8u resource,
    blt_int8u * key,
    blt_int8u len )
```

Called by the UNLOCK command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

Parameters

<i>resource</i>	resource to unlock (XCP_RES_XXX).
<i>key</i>	pointer to the byte buffer holding the key.
<i>len</i>	length of the key in bytes.

Returns

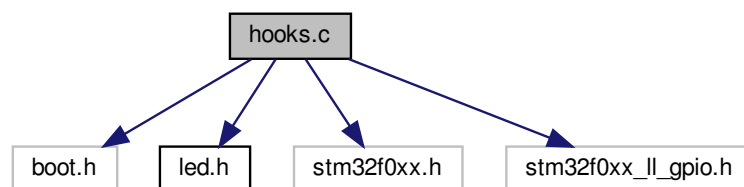
1 if the key was correct, 0 otherwise.

7.555 hooks.c File Reference

Bootloader callback source file.

```
#include "boot.h"
#include "led.h"
#include "stm32f0xx.h"
#include "stm32f0xx_ll_gpio.h"
```

Include dependency graph for ARMCM0_STM32F0_Nucleo_F091RC_IAR/Boot/hooks.c:



Functions

- void [BackDoorInitHook](#) (void)
Initializes the backdoor entry option.
- [blt_bool](#) [BackDoorEntryHook](#) (void)
Checks if a backdoor entry is requested.
- [blt_bool](#) [CpuUserProgramStartHook](#) (void)
Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.
- void [CopInitHook](#) (void)
Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.
- void [CopServiceHook](#) (void)
Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.
- void [NvmInitHook](#) (void)
Callback that gets called at the start of the internal NVM driver initialization routine.
- void [NvmReinitHook](#) (void)
Callback that gets called at the start of a firmware update to reinitialize the NVM driver.
- [blt_int8u](#) [NvmWriteHook](#) ([blt_addr](#) addr, [blt_int32u](#) len, [blt_int8u](#) *data)
Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the data hasn't been written yet.
- [blt_int8u](#) [NvmEraseHook](#) ([blt_addr](#) addr, [blt_int32u](#) len)

Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then `BLT_NVM_NOT_IN_RANGE` must be returned to indicate that the memory hasn't been erased yet.

- `blt_bool NvmDoneHook` (void)

Callback that gets called at the end of the NVM programming session.

- `blt_int8u XcpGetSeedHook` (`blt_int8u` resource, `blt_int8u *seed`)

Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the `GET_SEED` command.

- `blt_int8u XcpVerifyKeyHook` (`blt_int8u` resource, `blt_int8u *key`, `blt_int8u len`)

Called by the `UNLOCK` command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

7.555.1 Detailed Description

Bootloader callback source file.

7.555.2 Function Documentation

7.555.2.1 BackDoorEntryHook()

```
blt_bool BackDoorEntryHook (
    void )
```

Checks if a backdoor entry is requested.

Returns

`BLT_TRUE` if the backdoor entry is requested, `BLT_FALSE` otherwise.

7.555.2.2 BackDoorInitHook()

```
void BackDoorInitHook (
    void )
```

Initializes the backdoor entry option.

Returns

none.

7.555.2.3 CopInitHook()

```
void CopInitHook (  
    void )
```

Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.

Returns

none.

7.555.2.4 CopServiceHook()

```
void CopServiceHook (  
    void )
```

Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.

Returns

none.

7.555.2.5 CpuUserProgramStartHook()

```
blt_bool CpuUserProgramStartHook (  
    void )
```

Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.

Returns

BLT_TRUE if it is okay to start the user program, BLT_FALSE to keep keep the bootloader active.

7.555.2.6 NvmDoneHook()

```
blt_bool NvmDoneHook (  
    void )
```

Callback that gets called at the end of the NVM programming session.

Returns

BLT_TRUE is successful, BLT_FALSE otherwise.

7.555.2.7 NvmEraseHook()

```
blt_int8u NvmEraseHook (
    blt_addr addr,
    blt_int32u len )
```

Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the memory hasn't been erased yet.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.

Returns

BLT_NVM_OKAY if successful, BLT_NVM_NOT_IN_RANGE if the address is not within the supported memory range, or BLT_NVM_ERROR if the erase operation failed.

7.555.2.8 NvmInitHook()

```
void NvmInitHook (
    void )
```

Callback that gets called at the start of the internal NVM driver initialization routine.

Returns

none.

7.555.2.9 NvmReinitHook()

```
void NvmReinitHook (
    void )
```

Callback that gets called at the start of a firmware update to reinitialize the NVM driver.

Returns

none.

7.555.2.10 NvmWriteHook()

```
blt_int8u NvmWriteHook (
    blt_addr addr,
    blt_int32u len,
    blt_int8u * data )
```

Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the data hasn't been written yet.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.
<i>data</i>	Pointer to the data buffer.

Returns

BLT_NVM_OKAY if successful, BLT_NVM_NOT_IN_RANGE if the address is not within the supported memory range, or BLT_NVM_ERROR if the write operation failed.

7.555.2.11 XcpGetSeedHook()

```
blt_int8u XcpGetSeedHook (
    blt_int8u resource,
    blt_int8u * seed )
```

Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the GET_SEED command.

Parameters

<i>resource</i>	Resource that the seed is requested for (XCP_RES_XXX).
<i>seed</i>	Pointer to byte buffer where the seed will be stored.

Returns

Length of the seed in bytes.

7.555.2.12 XcpVerifyKeyHook()

```
blt_int8u XcpVerifyKeyHook (
    blt_int8u resource,
    blt_int8u * key,
    blt_int8u len )
```

Called by the UNLOCK command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

Parameters

<i>resource</i>	resource to unlock (XCP_RES_XXX).
<i>key</i>	pointer to the byte buffer holding the key.
<i>len</i>	length of the key in bytes.

Returns

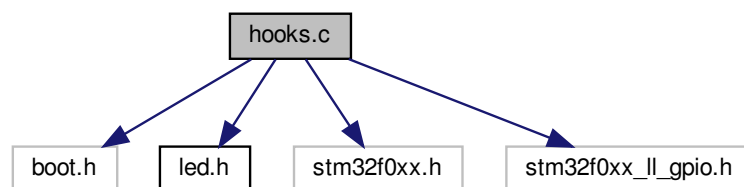
1 if the key was correct, 0 otherwise.

7.556 hooks.c File Reference

Bootloader callback source file.

```
#include "boot.h"
#include "led.h"
#include "stm32f0xx.h"
#include "stm32f0xx_ll_gpio.h"
```

Include dependency graph for ARMCM0_STM32F0_Nucleo_F091RC_Keil/Boot/hooks.c:



Functions

- void [BackDoorInitHook](#) (void)
Initializes the backdoor entry option.
- [blt_bool](#) [BackDoorEntryHook](#) (void)
Checks if a backdoor entry is requested.
- [blt_bool](#) [CpuUserProgramStartHook](#) (void)
Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.
- void [CopInitHook](#) (void)
Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.
- void [CopServiceHook](#) (void)
Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.
- void [NvmInitHook](#) (void)
Callback that gets called at the start of the internal NVM driver initialization routine.
- void [NvmReinitHook](#) (void)
Callback that gets called at the start of a firmware update to reinitialize the NVM driver.
- [blt_int8u](#) [NvmWriteHook](#) ([blt_addr](#) addr, [blt_int32u](#) len, [blt_int8u](#) *data)
Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the data hasn't been written yet.
- [blt_int8u](#) [NvmEraseHook](#) ([blt_addr](#) addr, [blt_int32u](#) len)

Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then `BLT_NVM_NOT_IN_RANGE` must be returned to indicate that the memory hasn't been erased yet.

- `blt_bool NvmDoneHook` (void)

Callback that gets called at the end of the NVM programming session.

- `blt_int8u XcpGetSeedHook` (`blt_int8u` resource, `blt_int8u` *seed)

Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the `GET_SEED` command.

- `blt_int8u XcpVerifyKeyHook` (`blt_int8u` resource, `blt_int8u` *key, `blt_int8u` len)

Called by the `UNLOCK` command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

7.556.1 Detailed Description

Bootloader callback source file.

7.556.2 Function Documentation

7.556.2.1 BackDoorEntryHook()

```
blt_bool BackDoorEntryHook (
    void )
```

Checks if a backdoor entry is requested.

Returns

`BLT_TRUE` if the backdoor entry is requested, `BLT_FALSE` otherwise.

7.556.2.2 BackDoorInitHook()

```
void BackDoorInitHook (
    void )
```

Initializes the backdoor entry option.

Returns

none.

7.556.2.3 CopInitHook()

```
void CopInitHook (
    void )
```

Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.

Returns

none.

7.556.2.4 CopServiceHook()

```
void CopServiceHook (
    void )
```

Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.

Returns

none.

7.556.2.5 CpuUserProgramStartHook()

```
blt_bool CpuUserProgramStartHook (
    void )
```

Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.

Returns

BLT_TRUE if it is okay to start the user program, BLT_FALSE to keep keep the bootloader active.

7.556.2.6 NvmDoneHook()

```
blt_bool NvmDoneHook (
    void )
```

Callback that gets called at the end of the NVM programming session.

Returns

BLT_TRUE is successful, BLT_FALSE otherwise.

7.556.2.7 NvmEraseHook()

```
blt_int8u NvmEraseHook (
    blt_addr addr,
    blt_int32u len )
```

Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the memory hasn't been erased yet.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.

Returns

BLT_NVM_OKAY if successful, BLT_NVM_NOT_IN_RANGE if the address is not within the supported memory range, or BLT_NVM_ERROR if the erase operation failed.

7.556.2.8 NvmInitHook()

```
void NvmInitHook (
    void )
```

Callback that gets called at the start of the internal NVM driver initialization routine.

Returns

none.

7.556.2.9 NvmReinitHook()

```
void NvmReinitHook (
    void )
```

Callback that gets called at the start of a firmware update to reinitialize the NVM driver.

Returns

none.

7.556.2.10 NvmWriteHook()

```
blt_int8u NvmWriteHook (
    blt_addr addr,
    blt_int32u len,
    blt_int8u * data )
```

Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the data hasn't been written yet.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.
<i>data</i>	Pointer to the data buffer.

Returns

BLT_NVM_OKAY if successful, BLT_NVM_NOT_IN_RANGE if the address is not within the supported memory range, or BLT_NVM_ERROR if the write operation failed.

7.556.2.11 XcpGetSeedHook()

```
blt_int8u XcpGetSeedHook (
    blt_int8u resource,
    blt_int8u * seed )
```

Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the GET_SEED command.

Parameters

<i>resource</i>	Resource that the seed is requested for (XCP_RES_XXX).
<i>seed</i>	Pointer to byte buffer where the seed will be stored.

Returns

Length of the seed in bytes.

7.556.2.12 XcpVerifyKeyHook()

```
blt_int8u XcpVerifyKeyHook (
    blt_int8u resource,
    blt_int8u * key,
    blt_int8u len )
```

Called by the UNLOCK command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

Parameters

<i>resource</i>	resource to unlock (XCP_RES_XXX).
<i>key</i>	pointer to the byte buffer holding the key.
<i>len</i>	length of the key in bytes.

Returns

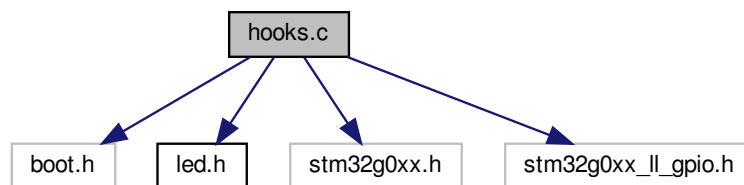
1 if the key was correct, 0 otherwise.

7.557 hooks.c File Reference

Bootloader callback source file.

```
#include "boot.h"
#include "led.h"
#include "stm32g0xx.h"
#include "stm32g0xx_ll_gpio.h"
```

Include dependency graph for ARMCM0_STM32G0_Nucleo_G071RB_CubeIDE/Boot/App/hooks.c:



Functions

- void [BackDoorInitHook](#) (void)
Initializes the backdoor entry option.
- [blt_bool](#) [BackDoorEntryHook](#) (void)
Checks if a backdoor entry is requested.
- [blt_bool](#) [CpuUserProgramStartHook](#) (void)
Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.
- void [CopInitHook](#) (void)
Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.
- void [CopServiceHook](#) (void)
Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.
- void [NvmInitHook](#) (void)
Callback that gets called at the start of the internal NVM driver initialization routine.
- void [NvmReinitHook](#) (void)
Callback that gets called at the start of a firmware update to reinitialize the NVM driver.
- [blt_int8u](#) [NvmWriteHook](#) ([blt_addr](#) addr, [blt_int32u](#) len, [blt_int8u](#) *data)
Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the data hasn't been written yet.
- [blt_int8u](#) [NvmEraseHook](#) ([blt_addr](#) addr, [blt_int32u](#) len)

Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then `BLT_NVM_NOT_IN_RANGE` must be returned to indicate that the memory hasn't been erased yet.

- `blt_bool NvmDoneHook` (void)

Callback that gets called at the end of the NVM programming session.

- `blt_int8u XcpGetSeedHook` (`blt_int8u` resource, `blt_int8u *`seed)

Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the `GET_SEED` command.

- `blt_int8u XcpVerifyKeyHook` (`blt_int8u` resource, `blt_int8u *`key, `blt_int8u` len)

Called by the `UNLOCK` command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

7.557.1 Detailed Description

Bootloader callback source file.

7.557.2 Function Documentation

7.557.2.1 BackDoorEntryHook()

```
blt_bool BackDoorEntryHook (
    void )
```

Checks if a backdoor entry is requested.

Returns

`BLT_TRUE` if the backdoor entry is requested, `BLT_FALSE` otherwise.

7.557.2.2 BackDoorInitHook()

```
void BackDoorInitHook (
    void )
```

Initializes the backdoor entry option.

Returns

none.

7.557.2.3 CopInitHook()

```
void CopInitHook (
    void )
```

Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.

Returns

none.

7.557.2.4 CopServiceHook()

```
void CopServiceHook (
    void )
```

Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.

Returns

none.

7.557.2.5 CpuUserProgramStartHook()

```
blt_bool CpuUserProgramStartHook (
    void )
```

Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.

Returns

BLT_TRUE if it is okay to start the user program, BLT_FALSE to keep keep the bootloader active.

7.557.2.6 NvmDoneHook()

```
blt_bool NvmDoneHook (
    void )
```

Callback that gets called at the end of the NVM programming session.

Returns

BLT_TRUE is successful, BLT_FALSE otherwise.

7.557.2.7 NvmEraseHook()

```
blt_int8u NvmEraseHook (
    blt_addr addr,
    blt_int32u len )
```

Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the memory hasn't been erased yet.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.

Returns

BLT_NVM_OKAY if successful, BLT_NVM_NOT_IN_RANGE if the address is not within the supported memory range, or BLT_NVM_ERROR if the erase operation failed.

7.557.2.8 NvmInitHook()

```
void NvmInitHook (
    void )
```

Callback that gets called at the start of the internal NVM driver initialization routine.

Returns

none.

7.557.2.9 NvmReinitHook()

```
void NvmReinitHook (
    void )
```

Callback that gets called at the start of a firmware update to reinitialize the NVM driver.

Returns

none.

7.557.2.10 NvmWriteHook()

```
blt_int8u NvmWriteHook (
    blt_addr addr,
    blt_int32u len,
    blt_int8u * data )
```

Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the data hasn't been written yet.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.
<i>data</i>	Pointer to the data buffer.

Returns

BLT_NVM_OKAY if successful, BLT_NVM_NOT_IN_RANGE if the address is not within the supported memory range, or BLT_NVM_ERROR if the write operation failed.

7.557.2.11 XcpGetSeedHook()

```
blt_int8u XcpGetSeedHook (
    blt_int8u resource,
    blt_int8u * seed )
```

Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the GET_SEED command.

Parameters

<i>resource</i>	Resource that the seed is requested for (XCP_RES_XXX).
<i>seed</i>	Pointer to byte buffer where the seed will be stored.

Returns

Length of the seed in bytes.

7.557.2.12 XcpVerifyKeyHook()

```
blt_int8u XcpVerifyKeyHook (
    blt_int8u resource,
    blt_int8u * key,
    blt_int8u len )
```

Called by the UNLOCK command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

Parameters

<i>resource</i>	resource to unlock (XCP_RES_XXX).
<i>key</i>	pointer to the byte buffer holding the key.
<i>len</i>	length of the key in bytes.

Returns

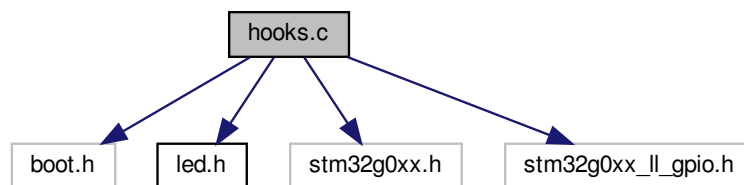
1 if the key was correct, 0 otherwise.

7.558 hooks.c File Reference

Bootloader callback source file.

```
#include "boot.h"
#include "led.h"
#include "stm32g0xx.h"
#include "stm32g0xx_ll_gpio.h"
```

Include dependency graph for ARMCM0_STM32G0_Nucleo_G071RB_GCC/Boot/hooks.c:



Functions

- void [BackDoorInitHook](#) (void)
Initializes the backdoor entry option.
- [blt_bool](#) [BackDoorEntryHook](#) (void)
Checks if a backdoor entry is requested.
- [blt_bool](#) [CpuUserProgramStartHook](#) (void)
Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.
- void [CopInitHook](#) (void)
Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.
- void [CopServiceHook](#) (void)
Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.
- void [NvmInitHook](#) (void)
Callback that gets called at the start of the internal NVM driver initialization routine.
- void [NvmReinitHook](#) (void)
Callback that gets called at the start of a firmware update to reinitialize the NVM driver.
- [blt_int8u](#) [NvmWriteHook](#) ([blt_addr](#) addr, [blt_int32u](#) len, [blt_int8u](#) *data)
Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the data hasn't been written yet.
- [blt_int8u](#) [NvmEraseHook](#) ([blt_addr](#) addr, [blt_int32u](#) len)

Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then `BLT_NVM_NOT_IN_RANGE` must be returned to indicate that the memory hasn't been erased yet.

- `blt_bool NvmDoneHook` (void)

Callback that gets called at the end of the NVM programming session.

- `blt_int8u XcpGetSeedHook` (`blt_int8u` resource, `blt_int8u *seed`)

Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the `GET_SEED` command.

- `blt_int8u XcpVerifyKeyHook` (`blt_int8u` resource, `blt_int8u *key`, `blt_int8u len`)

Called by the `UNLOCK` command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

7.558.1 Detailed Description

Bootloader callback source file.

7.558.2 Function Documentation

7.558.2.1 BackDoorEntryHook()

```
blt_bool BackDoorEntryHook (
    void )
```

Checks if a backdoor entry is requested.

Returns

`BLT_TRUE` if the backdoor entry is requested, `BLT_FALSE` otherwise.

7.558.2.2 BackDoorInitHook()

```
void BackDoorInitHook (
    void )
```

Initializes the backdoor entry option.

Returns

none.

7.558.2.3 CopInitHook()

```
void CopInitHook (
    void )
```

Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.

Returns

none.

7.558.2.4 CopServiceHook()

```
void CopServiceHook (
    void )
```

Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.

Returns

none.

7.558.2.5 CpuUserProgramStartHook()

```
blt_bool CpuUserProgramStartHook (
    void )
```

Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.

Returns

BLT_TRUE if it is okay to start the user program, BLT_FALSE to keep keep the bootloader active.

7.558.2.6 NvmDoneHook()

```
blt_bool NvmDoneHook (
    void )
```

Callback that gets called at the end of the NVM programming session.

Returns

BLT_TRUE is successful, BLT_FALSE otherwise.

7.558.2.7 NvmEraseHook()

```
blt_int8u NvmEraseHook (
    blt_addr addr,
    blt_int32u len )
```

Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the memory hasn't been erased yet.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.

Returns

BLT_NVM_OKAY if successful, BLT_NVM_NOT_IN_RANGE if the address is not within the supported memory range, or BLT_NVM_ERROR if the erase operation failed.

7.558.2.8 NvmInitHook()

```
void NvmInitHook (
    void )
```

Callback that gets called at the start of the internal NVM driver initialization routine.

Returns

none.

7.558.2.9 NvmReinitHook()

```
void NvmReinitHook (
    void )
```

Callback that gets called at the start of a firmware update to reinitialize the NVM driver.

Returns

none.

7.558.2.10 NvmWriteHook()

```
blt_int8u NvmWriteHook (
    blt_addr addr,
    blt_int32u len,
    blt_int8u * data )
```

Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the data hasn't been written yet.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.
<i>data</i>	Pointer to the data buffer.

Returns

BLT_NVM_OKAY if successful, BLT_NVM_NOT_IN_RANGE if the address is not within the supported memory range, or BLT_NVM_ERROR if the write operation failed.

7.558.2.11 XcpGetSeedHook()

```
blt_int8u XcpGetSeedHook (
    blt_int8u resource,
    blt_int8u * seed )
```

Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the GET_SEED command.

Parameters

<i>resource</i>	Resource that the seed is requested for (XCP_RES_XXX).
<i>seed</i>	Pointer to byte buffer where the seed will be stored.

Returns

Length of the seed in bytes.

7.558.2.12 XcpVerifyKeyHook()

```
blt_int8u XcpVerifyKeyHook (
    blt_int8u resource,
    blt_int8u * key,
    blt_int8u len )
```

Called by the UNLOCK command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

Parameters

<i>resource</i>	resource to unlock (XCP_RES_XXX).
<i>key</i>	pointer to the byte buffer holding the key.
<i>len</i>	length of the key in bytes.

Returns

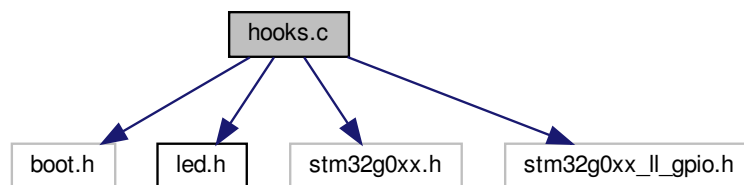
1 if the key was correct, 0 otherwise.

7.559 hooks.c File Reference

Bootloader callback source file.

```
#include "boot.h"
#include "led.h"
#include "stm32g0xx.h"
#include "stm32g0xx_ll_gpio.h"
```

Include dependency graph for ARMCM0_STM32G0_Nucleo_G071RB_IAR/Boot/hooks.c:



Functions

- void [BackDoorInitHook](#) (void)
Initializes the backdoor entry option.
- [blt_bool](#) [BackDoorEntryHook](#) (void)
Checks if a backdoor entry is requested.
- [blt_bool](#) [CpuUserProgramStartHook](#) (void)
Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.
- void [CopInitHook](#) (void)
Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.
- void [CopServiceHook](#) (void)
Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.
- void [NvmInitHook](#) (void)
Callback that gets called at the start of the internal NVM driver initialization routine.
- void [NvmReinitHook](#) (void)
Callback that gets called at the start of a firmware update to reinitialize the NVM driver.
- [blt_int8u](#) [NvmWriteHook](#) ([blt_addr](#) addr, [blt_int32u](#) len, [blt_int8u](#) *data)
Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the data hasn't been written yet.
- [blt_int8u](#) [NvmEraseHook](#) ([blt_addr](#) addr, [blt_int32u](#) len)

Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then `BLT_NVM_NOT_IN_RANGE` must be returned to indicate that the memory hasn't been erased yet.

- `blt_bool NvmDoneHook` (void)

Callback that gets called at the end of the NVM programming session.

- `blt_int8u XcpGetSeedHook` (`blt_int8u` resource, `blt_int8u *seed`)

Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the `GET_SEED` command.

- `blt_int8u XcpVerifyKeyHook` (`blt_int8u` resource, `blt_int8u *key`, `blt_int8u len`)

Called by the `UNLOCK` command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

7.559.1 Detailed Description

Bootloader callback source file.

7.559.2 Function Documentation

7.559.2.1 BackDoorEntryHook()

```
blt_bool BackDoorEntryHook (
    void )
```

Checks if a backdoor entry is requested.

Returns

`BLT_TRUE` if the backdoor entry is requested, `BLT_FALSE` otherwise.

7.559.2.2 BackDoorInitHook()

```
void BackDoorInitHook (
    void )
```

Initializes the backdoor entry option.

Returns

none.

7.559.2.3 CopInitHook()

```
void CopInitHook (
    void )
```

Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.

Returns

none.

7.559.2.4 CopServiceHook()

```
void CopServiceHook (
    void )
```

Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.

Returns

none.

7.559.2.5 CpuUserProgramStartHook()

```
blt_bool CpuUserProgramStartHook (
    void )
```

Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.

Returns

BLT_TRUE if it is okay to start the user program, BLT_FALSE to keep keep the bootloader active.

7.559.2.6 NvmDoneHook()

```
blt_bool NvmDoneHook (
    void )
```

Callback that gets called at the end of the NVM programming session.

Returns

BLT_TRUE is successful, BLT_FALSE otherwise.

7.559.2.7 NvmEraseHook()

```
blt_int8u NvmEraseHook (
    blt_addr addr,
    blt_int32u len )
```

Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the memory hasn't been erased yet.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.

Returns

BLT_NVM_OKAY if successful, BLT_NVM_NOT_IN_RANGE if the address is not within the supported memory range, or BLT_NVM_ERROR if the erase operation failed.

7.559.2.8 NvmInitHook()

```
void NvmInitHook (
    void )
```

Callback that gets called at the start of the internal NVM driver initialization routine.

Returns

none.

7.559.2.9 NvmReinitHook()

```
void NvmReinitHook (
    void )
```

Callback that gets called at the start of a firmware update to reinitialize the NVM driver.

Returns

none.

7.559.2.10 NvmWriteHook()

```
blt_int8u NvmWriteHook (
    blt_addr addr,
    blt_int32u len,
    blt_int8u * data )
```

Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the data hasn't been written yet.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.
<i>data</i>	Pointer to the data buffer.

Returns

BLT_NVM_OKAY if successful, BLT_NVM_NOT_IN_RANGE if the address is not within the supported memory range, or BLT_NVM_ERROR if the write operation failed.

7.559.2.11 XcpGetSeedHook()

```
blt_int8u XcpGetSeedHook (
    blt_int8u resource,
    blt_int8u * seed )
```

Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the GET_SEED command.

Parameters

<i>resource</i>	Resource that the seed is requested for (XCP_RES_XXX).
<i>seed</i>	Pointer to byte buffer where the seed will be stored.

Returns

Length of the seed in bytes.

7.559.2.12 XcpVerifyKeyHook()

```
blt_int8u XcpVerifyKeyHook (
    blt_int8u resource,
    blt_int8u * key,
    blt_int8u len )
```

Called by the UNLOCK command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

Parameters

<i>resource</i>	resource to unlock (XCP_RES_XXX).
<i>key</i>	pointer to the byte buffer holding the key.
<i>len</i>	length of the key in bytes.

Returns

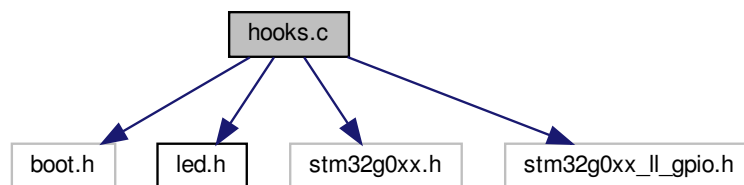
1 if the key was correct, 0 otherwise.

7.560 hooks.c File Reference

Bootloader callback source file.

```
#include "boot.h"
#include "led.h"
#include "stm32g0xx.h"
#include "stm32g0xx_ll_gpio.h"
```

Include dependency graph for ARMCM0_STM32G0_Nucleo_G071RB_Keil/Boot/hooks.c:



Functions

- void [BackDoorInitHook](#) (void)
Initializes the backdoor entry option.
- [blt_bool](#) [BackDoorEntryHook](#) (void)
Checks if a backdoor entry is requested.
- [blt_bool](#) [CpuUserProgramStartHook](#) (void)
Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.
- void [CopInitHook](#) (void)
Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.
- void [CopServiceHook](#) (void)
Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.
- void [NvmInitHook](#) (void)
Callback that gets called at the start of the internal NVM driver initialization routine.
- void [NvmReinitHook](#) (void)
Callback that gets called at the start of a firmware update to reinitialize the NVM driver.
- [blt_int8u](#) [NvmWriteHook](#) ([blt_addr](#) addr, [blt_int32u](#) len, [blt_int8u](#) *data)
Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the data hasn't been written yet.
- [blt_int8u](#) [NvmEraseHook](#) ([blt_addr](#) addr, [blt_int32u](#) len)

Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then `BLT_NVM_NOT_IN_RANGE` must be returned to indicate that the memory hasn't been erased yet.

- `blt_bool NvmDoneHook` (void)

Callback that gets called at the end of the NVM programming session.

- `blt_int8u XcpGetSeedHook` (`blt_int8u` resource, `blt_int8u *seed`)

Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the `GET_SEED` command.

- `blt_int8u XcpVerifyKeyHook` (`blt_int8u` resource, `blt_int8u *key`, `blt_int8u len`)

Called by the `UNLOCK` command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

7.560.1 Detailed Description

Bootloader callback source file.

7.560.2 Function Documentation

7.560.2.1 BackDoorEntryHook()

```
blt_bool BackDoorEntryHook (
    void )
```

Checks if a backdoor entry is requested.

Returns

`BLT_TRUE` if the backdoor entry is requested, `BLT_FALSE` otherwise.

7.560.2.2 BackDoorInitHook()

```
void BackDoorInitHook (
    void )
```

Initializes the backdoor entry option.

Returns

none.

7.560.2.3 CopInitHook()

```
void CopInitHook (  
    void )
```

Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.

Returns

none.

7.560.2.4 CopServiceHook()

```
void CopServiceHook (  
    void )
```

Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.

Returns

none.

7.560.2.5 CpuUserProgramStartHook()

```
blt_bool CpuUserProgramStartHook (  
    void )
```

Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.

Returns

BLT_TRUE if it is okay to start the user program, BLT_FALSE to keep keep the bootloader active.

7.560.2.6 NvmDoneHook()

```
blt_bool NvmDoneHook (  
    void )
```

Callback that gets called at the end of the NVM programming session.

Returns

BLT_TRUE is successful, BLT_FALSE otherwise.

7.560.2.7 NvmEraseHook()

```
blt_int8u NvmEraseHook (
    blt_addr addr,
    blt_int32u len )
```

Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the memory hasn't been erased yet.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.

Returns

BLT_NVM_OKAY if successful, BLT_NVM_NOT_IN_RANGE if the address is not within the supported memory range, or BLT_NVM_ERROR if the erase operation failed.

7.560.2.8 NvmInitHook()

```
void NvmInitHook (
    void )
```

Callback that gets called at the start of the internal NVM driver initialization routine.

Returns

none.

7.560.2.9 NvmReinitHook()

```
void NvmReinitHook (
    void )
```

Callback that gets called at the start of a firmware update to reinitialize the NVM driver.

Returns

none.

7.560.2.10 NvmWriteHook()

```
blt_int8u NvmWriteHook (
    blt_addr addr,
    blt_int32u len,
    blt_int8u * data )
```

Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the data hasn't been written yet.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.
<i>data</i>	Pointer to the data buffer.

Returns

BLT_NVM_OKAY if successful, BLT_NVM_NOT_IN_RANGE if the address is not within the supported memory range, or BLT_NVM_ERROR if the write operation failed.

7.560.2.11 XcpGetSeedHook()

```
blt_int8u XcpGetSeedHook (
    blt_int8u resource,
    blt_int8u * seed )
```

Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the GET_SEED command.

Parameters

<i>resource</i>	Resource that the seed is requested for (XCP_RES_XXX).
<i>seed</i>	Pointer to byte buffer where the seed will be stored.

Returns

Length of the seed in bytes.

7.560.2.12 XcpVerifyKeyHook()

```
blt_int8u XcpVerifyKeyHook (
    blt_int8u resource,
    blt_int8u * key,
    blt_int8u len )
```

Called by the UNLOCK command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

Parameters

<i>resource</i>	resource to unlock (XCP_RES_XXX).
<i>key</i>	pointer to the byte buffer holding the key.
<i>len</i>	length of the key in bytes.

Returns

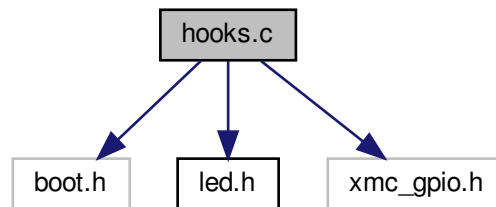
1 if the key was correct, 0 otherwise.

7.561 hooks.c File Reference

Bootloader callback source file.

```
#include "boot.h"
#include "led.h"
#include "xmc_gpio.h"
```

Include dependency graph for ARMCM0_XMC1_XMC1400_Boot_Kit_GCC/Boot/hooks.c:



Functions

- void [BackDoorInitHook](#) (void)
Initializes the backdoor entry option.
- [blt_bool](#) [BackDoorEntryHook](#) (void)
Checks if a backdoor entry is requested.
- [blt_bool](#) [CpuUserProgramStartHook](#) (void)
Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.
- void [CopInitHook](#) (void)
Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.
- void [CopServiceHook](#) (void)
Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.
- void [NvmInitHook](#) (void)
Callback that gets called at the start of the internal NVM driver initialization routine.
- void [NvmReinitHook](#) (void)
Callback that gets called at the start of a firmware update to reinitialize the NVM driver.
- [blt_int8u](#) [NvmWriteHook](#) ([blt_addr](#) addr, [blt_int32u](#) len, [blt_int8u](#) *data)
Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the data hasn't been written yet.
- [blt_int8u](#) [NvmEraseHook](#) ([blt_addr](#) addr, [blt_int32u](#) len)

Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then `BLT_NVM_NOT_IN_RANGE` must be returned to indicate that the memory hasn't been erased yet.

- `blt_bool NvmDoneHook` (void)

Callback that gets called at the end of the NVM programming session.

- `blt_int8u XcpGetSeedHook` (`blt_int8u` resource, `blt_int8u *seed`)

Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the `GET_SEED` command.

- `blt_int8u XcpVerifyKeyHook` (`blt_int8u` resource, `blt_int8u *key`, `blt_int8u len`)

Called by the `UNLOCK` command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

7.561.1 Detailed Description

Bootloader callback source file.

7.561.2 Function Documentation

7.561.2.1 BackDoorEntryHook()

```
blt_bool BackDoorEntryHook (
    void )
```

Checks if a backdoor entry is requested.

Returns

`BLT_TRUE` if the backdoor entry is requested, `BLT_FALSE` otherwise.

7.561.2.2 BackDoorInitHook()

```
void BackDoorInitHook (
    void )
```

Initializes the backdoor entry option.

Returns

none.

7.561.2.3 CopInitHook()

```
void CopInitHook (
    void )
```

Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.

Returns

none.

7.561.2.4 CopServiceHook()

```
void CopServiceHook (
    void )
```

Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.

Returns

none.

7.561.2.5 CpuUserProgramStartHook()

```
blt_bool CpuUserProgramStartHook (
    void )
```

Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.

Returns

BLT_TRUE if it is okay to start the user program, BLT_FALSE to keep keep the bootloader active.

7.561.2.6 NvmDoneHook()

```
blt_bool NvmDoneHook (
    void )
```

Callback that gets called at the end of the NVM programming session.

Returns

BLT_TRUE is successful, BLT_FALSE otherwise.

7.561.2.7 NvmEraseHook()

```
blt_int8u NvmEraseHook (
    blt_addr addr,
    blt_int32u len )
```

Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the memory hasn't been erased yet.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.

Returns

BLT_NVM_OKAY if successful, BLT_NVM_NOT_IN_RANGE if the address is not within the supported memory range, or BLT_NVM_ERROR if the erase operation failed.

7.561.2.8 NvmInitHook()

```
void NvmInitHook (
    void )
```

Callback that gets called at the start of the internal NVM driver initialization routine.

Returns

none.

7.561.2.9 NvmReinitHook()

```
void NvmReinitHook (
    void )
```

Callback that gets called at the start of a firmware update to reinitialize the NVM driver.

Returns

none.

7.561.2.10 NvmWriteHook()

```
blt_int8u NvmWriteHook (
    blt_addr addr,
    blt_int32u len,
    blt_int8u * data )
```

Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the data hasn't been written yet.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.
<i>data</i>	Pointer to the data buffer.

Returns

BLT_NVM_OKAY if successful, BLT_NVM_NOT_IN_RANGE if the address is not within the supported memory range, or BLT_NVM_ERROR if the write operation failed.

7.561.2.11 XcpGetSeedHook()

```
blt_int8u XcpGetSeedHook (
    blt_int8u resource,
    blt_int8u * seed )
```

Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the GET_SEED command.

Parameters

<i>resource</i>	Resource that the seed is requested for (XCP_RES_XXX).
<i>seed</i>	Pointer to byte buffer where the seed will be stored.

Returns

Length of the seed in bytes.

7.561.2.12 XcpVerifyKeyHook()

```
blt_int8u XcpVerifyKeyHook (
    blt_int8u resource,
    blt_int8u * key,
    blt_int8u len )
```

Called by the UNLOCK command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

Parameters

<i>resource</i>	resource to unlock (XCP_RES_XXX).
<i>key</i>	pointer to the byte buffer holding the key.
<i>len</i>	length of the key in bytes.

Returns

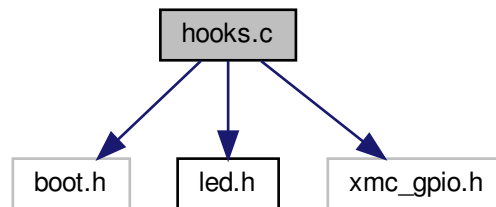
1 if the key was correct, 0 otherwise.

7.562 hooks.c File Reference

Bootloader callback source file.

```
#include "boot.h"
#include "led.h"
#include "xmc_gpio.h"
```

Include dependency graph for ARMCM0_XMC1_XMC1400_Boot_Kit_IAR/Boot/hooks.c:



Functions

- void [BackDoorInitHook](#) (void)
Initializes the backdoor entry option.
- [blt_bool](#) [BackDoorEntryHook](#) (void)
Checks if a backdoor entry is requested.
- [blt_bool](#) [CpuUserProgramStartHook](#) (void)
Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.
- void [CopInitHook](#) (void)
Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.
- void [CopServiceHook](#) (void)
Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.
- void [NvmInitHook](#) (void)
Callback that gets called at the start of the internal NVM driver initialization routine.
- void [NvmReinitHook](#) (void)
Callback that gets called at the start of a firmware update to reinitialize the NVM driver.
- [blt_int8u](#) [NvmWriteHook](#) ([blt_addr](#) addr, [blt_int32u](#) len, [blt_int8u](#) *data)
Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the data hasn't been written yet.
- [blt_int8u](#) [NvmEraseHook](#) ([blt_addr](#) addr, [blt_int32u](#) len)

Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then `BLT_NVM_NOT_IN_RANGE` must be returned to indicate that the memory hasn't been erased yet.

- `blt_bool NvmDoneHook` (void)

Callback that gets called at the end of the NVM programming session.

- `blt_int8u XcpGetSeedHook` (`blt_int8u` resource, `blt_int8u *`seed)

Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the `GET_SEED` command.

- `blt_int8u XcpVerifyKeyHook` (`blt_int8u` resource, `blt_int8u *`key, `blt_int8u` len)

Called by the `UNLOCK` command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

7.562.1 Detailed Description

Bootloader callback source file.

7.562.2 Function Documentation

7.562.2.1 BackDoorEntryHook()

```
blt_bool BackDoorEntryHook (
    void )
```

Checks if a backdoor entry is requested.

Returns

`BLT_TRUE` if the backdoor entry is requested, `BLT_FALSE` otherwise.

7.562.2.2 BackDoorInitHook()

```
void BackDoorInitHook (
    void )
```

Initializes the backdoor entry option.

Returns

none.

7.562.2.3 CopInitHook()

```
void CopInitHook (
    void )
```

Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.

Returns

none.

7.562.2.4 CopServiceHook()

```
void CopServiceHook (
    void )
```

Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.

Returns

none.

7.562.2.5 CpuUserProgramStartHook()

```
blt_bool CpuUserProgramStartHook (
    void )
```

Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.

Returns

BLT_TRUE if it is okay to start the user program, BLT_FALSE to keep keep the bootloader active.

7.562.2.6 NvmDoneHook()

```
blt_bool NvmDoneHook (
    void )
```

Callback that gets called at the end of the NVM programming session.

Returns

BLT_TRUE is successful, BLT_FALSE otherwise.

7.562.2.7 NvmEraseHook()

```
blt_int8u NvmEraseHook (
    blt_addr addr,
    blt_int32u len )
```

Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the memory hasn't been erased yet.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.

Returns

BLT_NVM_OKAY if successful, BLT_NVM_NOT_IN_RANGE if the address is not within the supported memory range, or BLT_NVM_ERROR if the erase operation failed.

7.562.2.8 NvmInitHook()

```
void NvmInitHook (
    void )
```

Callback that gets called at the start of the internal NVM driver initialization routine.

Returns

none.

7.562.2.9 NvmReinitHook()

```
void NvmReinitHook (
    void )
```

Callback that gets called at the start of a firmware update to reinitialize the NVM driver.

Returns

none.

7.562.2.10 NvmWriteHook()

```
blt_int8u NvmWriteHook (
    blt_addr addr,
    blt_int32u len,
    blt_int8u * data )
```

Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the data hasn't been written yet.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.
<i>data</i>	Pointer to the data buffer.

Returns

BLT_NVM_OKAY if successful, BLT_NVM_NOT_IN_RANGE if the address is not within the supported memory range, or BLT_NVM_ERROR if the write operation failed.

7.562.2.11 XcpGetSeedHook()

```
blt_int8u XcpGetSeedHook (
    blt_int8u resource,
    blt_int8u * seed )
```

Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the GET_SEED command.

Parameters

<i>resource</i>	Resource that the seed is requested for (XCP_RES_XXX).
<i>seed</i>	Pointer to byte buffer where the seed will be stored.

Returns

Length of the seed in bytes.

7.562.2.12 XcpVerifyKeyHook()

```
blt_int8u XcpVerifyKeyHook (
    blt_int8u resource,
    blt_int8u * key,
    blt_int8u len )
```

Called by the UNLOCK command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

Parameters

<i>resource</i>	resource to unlock (XCP_RES_XXX).
<i>key</i>	pointer to the byte buffer holding the key.
<i>len</i>	length of the key in bytes.

Returns

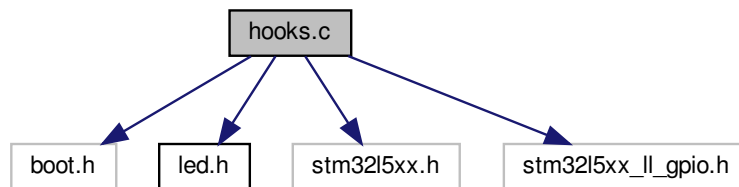
1 if the key was correct, 0 otherwise.

7.563 hooks.c File Reference

Bootloader callback source file.

```
#include "boot.h"
#include "led.h"
#include "stm32l5xx.h"
#include "stm32l5xx_ll_gpio.h"
```

Include dependency graph for ARMCM33_STM32L5_Nucleo_L552ZE_CubeIDE/Boot/App/hooks.c:



Functions

- void [BackDoorInitHook](#) (void)
Initializes the backdoor entry option.
- [blt_bool](#) [BackDoorEntryHook](#) (void)
Checks if a backdoor entry is requested.
- [blt_bool](#) [CpuUserProgramStartHook](#) (void)
Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.
- void [CopInitHook](#) (void)
Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.
- void [CopServiceHook](#) (void)
Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.
- void [UsbConnectHook](#) ([blt_bool](#) connect)
Callback that gets called whenever the USB device should be connected to the USB bus.
- void [UsbEnterLowPowerModeHook](#) (void)
Callback that gets called whenever the USB host requests the device to enter a low power mode.
- void [UsbLeaveLowPowerModeHook](#) (void)
Callback that gets called whenever the USB host requests the device to exit low power mode.
- void [NvmInitHook](#) (void)
Callback that gets called at the start of the internal NVM driver initialization routine.
- void [NvmReinitHook](#) (void)

Callback that gets called at the start of a firmware update to reinitialize the NVM driver.

- `blt_int8u NvmWriteHook (blt_addr addr, blt_int32u len, blt_int8u *data)`

Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then `BLT_NVM_NOT_IN_RANGE` must be returned to indicate that the data hasn't been written yet.

- `blt_int8u NvmEraseHook (blt_addr addr, blt_int32u len)`

Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then `BLT_NVM_NOT_IN_RANGE` must be returned to indicate that the memory hasn't been erased yet.

- `blt_bool NvmDoneHook (void)`

Callback that gets called at the end of the NVM programming session.

- `blt_int8u XcpGetSeedHook (blt_int8u resource, blt_int8u *seed)`

Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the `GET_SEED` command.

- `blt_int8u XcpVerifyKeyHook (blt_int8u resource, blt_int8u *key, blt_int8u len)`

Called by the `UNLOCK` command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

7.563.1 Detailed Description

Bootloader callback source file.

7.563.2 Function Documentation

7.563.2.1 BackDoorEntryHook()

```
blt_bool BackDoorEntryHook (
    void )
```

Checks if a backdoor entry is requested.

Returns

`BLT_TRUE` if the backdoor entry is requested, `BLT_FALSE` otherwise.

Referenced by `UsbLeaveLowPowerModeHook()`.

7.563.2.2 BackDoorInitHook()

```
void BackDoorInitHook (
    void )
```

Initializes the backdoor entry option.

Returns

none.

Referenced by `UsbLeaveLowPowerModeHook()`.

7.563.2.3 CopInitHook()

```
void CopInitHook (
    void )
```

Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.

Returns

none.

7.563.2.4 CopServiceHook()

```
void CopServiceHook (
    void )
```

Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.

Returns

none.

7.563.2.5 CpuUserProgramStartHook()

```
blt_bool CpuUserProgramStartHook (
    void )
```

Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.

Returns

BLT_TRUE if it is okay to start the user program, BLT_FALSE to keep keep the bootloader active.

7.563.2.6 NvmDoneHook()

```
blt_bool NvmDoneHook (
    void )
```

Callback that gets called at the end of the NVM programming session.

Returns

BLT_TRUE is successful, BLT_FALSE otherwise.

7.563.2.7 NvmEraseHook()

```
blt_int8u NvmEraseHook (
    blt_addr addr,
    blt_int32u len )
```

Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the memory hasn't been erased yet.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.

Returns

BLT_NVM_OKAY if successful, BLT_NVM_NOT_IN_RANGE if the address is not within the supported memory range, or BLT_NVM_ERROR if the erase operation failed.

7.563.2.8 NvmInitHook()

```
void NvmInitHook (
    void )
```

Callback that gets called at the start of the internal NVM driver initialization routine.

Returns

none.

7.563.2.9 NvmReinitHook()

```
void NvmReinitHook (
    void )
```

Callback that gets called at the start of a firmware update to reinitialize the NVM driver.

Returns

none.

7.563.2.10 NvmWriteHook()

```
blt_int8u NvmWriteHook (
    blt_addr addr,
    blt_int32u len,
    blt_int8u * data )
```

Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the data hasn't been written yet.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.
<i>data</i>	Pointer to the data buffer.

Returns

BLT_NVM_OKAY if successful, BLT_NVM_NOT_IN_RANGE if the address is not within the supported memory range, or BLT_NVM_ERROR if the write operation failed.

7.563.2.11 UsbConnectHook()

```
void UsbConnectHook (
    blt_bool connect )
```

Callback that gets called whenever the USB device should be connected to the USB bus.

Parameters

<i>connect</i>	BLT_TRUE to connect and BLT_FALSE to disconnect.
----------------	--

Returns

none.

Referenced by UsbFree(), and UsbInit().

7.563.2.12 UsbEnterLowPowerModeHook()

```
void UsbEnterLowPowerModeHook (
    void )
```

Callback that gets called whenever the USB host requests the device to enter a low power mode.

Returns

none.

Referenced by UsbBulkRxHandler().

7.563.2.13 UsbLeaveLowPowerModeHook()

```
void UsbLeaveLowPowerModeHook (  
    void )
```

Callback that gets called whenever the USB host requests the device to exit low power mode.

Returns

none.

Referenced by UsbBulkRxHandler().

7.563.2.14 XcpGetSeedHook()

```
blt_int8u XcpGetSeedHook (  
    blt_int8u resource,  
    blt_int8u * seed )
```

Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the GET_SEED command.

Parameters

<i>resource</i>	Resource that the seed is requested for (XCP_RES_XXX).
<i>seed</i>	Pointer to byte buffer where the seed will be stored.

Returns

Length of the seed in bytes.

7.563.2.15 XcpVerifyKeyHook()

```
blt_int8u XcpVerifyKeyHook (  
    blt_int8u resource,  
    blt_int8u * key,  
    blt_int8u len )
```

Called by the UNLOCK command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

Parameters

<i>resource</i>	resource to unlock (XCP_RES_XXX).
<i>key</i>	pointer to the byte buffer holding the key.
<i>len</i>	length of the key in bytes.

Returns

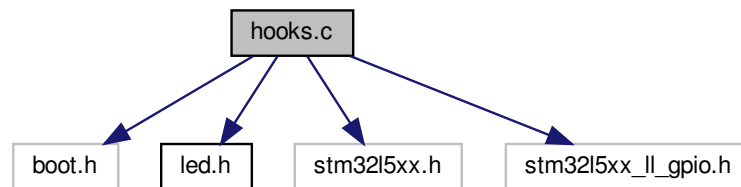
1 if the key was correct, 0 otherwise.

7.564 hooks.c File Reference

Bootloader callback source file.

```
#include "boot.h"
#include "led.h"
#include "stm32l5xx.h"
#include "stm32l5xx_ll_gpio.h"
```

Include dependency graph for ARMCM33_STM32L5_Nucleo_L552ZE_GCC/Boot/hooks.c:



Functions

- void [BackDoorInitHook](#) (void)
Initializes the backdoor entry option.
- [blt_bool](#) [BackDoorEntryHook](#) (void)
Checks if a backdoor entry is requested.
- [blt_bool](#) [CpuUserProgramStartHook](#) (void)
Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.
- void [CopInitHook](#) (void)
Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.
- void [CopServiceHook](#) (void)
Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.
- void [UsbConnectHook](#) ([blt_bool](#) connect)
Callback that gets called whenever the USB device should be connected to the USB bus.
- void [UsbEnterLowPowerModeHook](#) (void)
Callback that gets called whenever the USB host requests the device to enter a low power mode.
- void [UsbLeaveLowPowerModeHook](#) (void)
Callback that gets called whenever the USB host requests the device to exit low power mode.
- void [NvmInitHook](#) (void)
Callback that gets called at the start of the internal NVM driver initialization routine.
- void [NvmReinitHook](#) (void)

Callback that gets called at the start of a firmware update to reinitialize the NVM driver.

- `blt_int8u NvmWriteHook (blt_addr addr, blt_int32u len, blt_int8u *data)`

Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then `BLT_NVM_NOT_IN_RANGE` must be returned to indicate that the data hasn't been written yet.

- `blt_int8u NvmEraseHook (blt_addr addr, blt_int32u len)`

Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then `BLT_NVM_NOT_IN_RANGE` must be returned to indicate that the memory hasn't been erased yet.

- `blt_bool NvmDoneHook (void)`

Callback that gets called at the end of the NVM programming session.

- `blt_int8u XcpGetSeedHook (blt_int8u resource, blt_int8u *seed)`

Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the `GET_SEED` command.

- `blt_int8u XcpVerifyKeyHook (blt_int8u resource, blt_int8u *key, blt_int8u len)`

Called by the `UNLOCK` command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

7.564.1 Detailed Description

Bootloader callback source file.

7.564.2 Function Documentation

7.564.2.1 BackDoorEntryHook()

```
blt_bool BackDoorEntryHook (
    void )
```

Checks if a backdoor entry is requested.

Returns

`BLT_TRUE` if the backdoor entry is requested, `BLT_FALSE` otherwise.

Referenced by `UsbLeaveLowPowerModeHook()`.

7.564.2.2 BackDoorInitHook()

```
void BackDoorInitHook (
    void )
```

Initializes the backdoor entry option.

Returns

none.

Referenced by `UsbLeaveLowPowerModeHook()`.

7.564.2.3 CopInitHook()

```
void CopInitHook (
    void )
```

Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.

Returns

none.

7.564.2.4 CopServiceHook()

```
void CopServiceHook (
    void )
```

Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.

Returns

none.

7.564.2.5 CpuUserProgramStartHook()

```
blt_bool CpuUserProgramStartHook (
    void )
```

Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.

Returns

BLT_TRUE if it is okay to start the user program, BLT_FALSE to keep keep the bootloader active.

7.564.2.6 NvmDoneHook()

```
blt_bool NvmDoneHook (
    void )
```

Callback that gets called at the end of the NVM programming session.

Returns

BLT_TRUE is successful, BLT_FALSE otherwise.

7.564.2.7 NvmEraseHook()

```
blt_int8u NvmEraseHook (
    blt_addr addr,
    blt_int32u len )
```

Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the memory hasn't been erased yet.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.

Returns

BLT_NVM_OKAY if successful, BLT_NVM_NOT_IN_RANGE if the address is not within the supported memory range, or BLT_NVM_ERROR if the erase operation failed.

7.564.2.8 NvmInitHook()

```
void NvmInitHook (
    void )
```

Callback that gets called at the start of the internal NVM driver initialization routine.

Returns

none.

7.564.2.9 NvmReinitHook()

```
void NvmReinitHook (
    void )
```

Callback that gets called at the start of a firmware update to reinitialize the NVM driver.

Returns

none.

7.564.2.10 NvmWriteHook()

```
blt_int8u NvmWriteHook (
    blt_addr addr,
    blt_int32u len,
    blt_int8u * data )
```

Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the data hasn't been written yet.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.
<i>data</i>	Pointer to the data buffer.

Returns

BLT_NVM_OKAY if successful, BLT_NVM_NOT_IN_RANGE if the address is not within the supported memory range, or BLT_NVM_ERROR if the write operation failed.

7.564.2.11 UsbConnectHook()

```
void UsbConnectHook (
    blt_bool connect )
```

Callback that gets called whenever the USB device should be connected to the USB bus.

Parameters

<i>connect</i>	BLT_TRUE to connect and BLT_FALSE to disconnect.
----------------	--

Returns

none.

7.564.2.12 UsbEnterLowPowerModeHook()

```
void UsbEnterLowPowerModeHook (
    void )
```

Callback that gets called whenever the USB host requests the device to enter a low power mode.

Returns

none.

7.564.2.13 UsbLeaveLowPowerModeHook()

```
void UsbLeaveLowPowerModeHook (
    void )
```

Callback that gets called whenever the USB host requests the device to exit low power mode.

Returns

none.

7.564.2.14 XcpGetSeedHook()

```
blt_int8u XcpGetSeedHook (
    blt_int8u resource,
    blt_int8u * seed )
```

Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the GET_SEED command.

Parameters

<i>resource</i>	Resource that the seed is requested for (XCP_RES_XXX).
<i>seed</i>	Pointer to byte buffer where the seed will be stored.

Returns

Length of the seed in bytes.

7.564.2.15 XcpVerifyKeyHook()

```
blt_int8u XcpVerifyKeyHook (
    blt_int8u resource,
    blt_int8u * key,
    blt_int8u len )
```

Called by the UNLOCK command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

Parameters

<i>resource</i>	resource to unlock (XCP_RES_XXX).
<i>key</i>	pointer to the byte buffer holding the key.
<i>len</i>	length of the key in bytes.

Returns

1 if the key was correct, 0 otherwise.

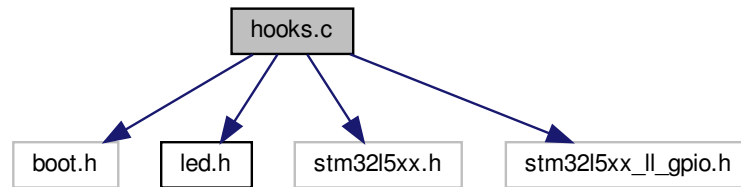
7.565 hooks.c File Reference

Bootloader callback source file.

```
#include "boot.h"
#include "led.h"
#include "stm32l5xx.h"
```

```
#include "stm32l5xx_ll_gpio.h"
```

Include dependency graph for ARMCM33_STM32L5_Nucleo_L552ZE_IAR/Boot/hooks.c:



Functions

- void [BackDoorInitHook](#) (void)
Initializes the backdoor entry option.
- [blt_bool BackDoorEntryHook](#) (void)
Checks if a backdoor entry is requested.
- [blt_bool CpuUserProgramStartHook](#) (void)
Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.
- void [CopInitHook](#) (void)
Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.
- void [CopServiceHook](#) (void)
Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.
- void [UsbConnectHook](#) ([blt_bool](#) connect)
Callback that gets called whenever the USB device should be connected to the USB bus.
- void [UsbEnterLowPowerModeHook](#) (void)
Callback that gets called whenever the USB host requests the device to enter a low power mode.
- void [UsbLeaveLowPowerModeHook](#) (void)
Callback that gets called whenever the USB host requests the device to exit low power mode.
- void [NvmInitHook](#) (void)
Callback that gets called at the start of the internal NVM driver initialization routine.
- void [NvmReinitHook](#) (void)
Callback that gets called at the start of a firmware update to reinitialize the NVM driver.
- [blt_int8u NvmWriteHook](#) ([blt_addr](#) addr, [blt_int32u](#) len, [blt_int8u](#) *data)
Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the data hasn't been written yet.
- [blt_int8u NvmEraseHook](#) ([blt_addr](#) addr, [blt_int32u](#) len)
Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the memory hasn't been erased yet.
- [blt_bool NvmDoneHook](#) (void)
Callback that gets called at the end of the NVM programming session.
- [blt_int8u XcpGetSeedHook](#) ([blt_int8u](#) resource, [blt_int8u](#) *seed)

Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the GET_SEED command.

- `blt_int8u XcpVerifyKeyHook (blt_int8u resource, blt_int8u *key, blt_int8u len)`

Called by the UNLOCK command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

7.565.1 Detailed Description

Bootloader callback source file.

7.565.2 Function Documentation

7.565.2.1 BackDoorEntryHook()

```
blt_bool BackDoorEntryHook (  
    void )
```

Checks if a backdoor entry is requested.

Returns

BLT_TRUE if the backdoor entry is requested, BLT_FALSE otherwise.

Referenced by `UsbLeaveLowPowerModeHook()`.

7.565.2.2 BackDoorInitHook()

```
void BackDoorInitHook (  
    void )
```

Initializes the backdoor entry option.

Returns

none.

Referenced by `UsbLeaveLowPowerModeHook()`.

7.565.2.3 CopInitHook()

```
void CopInitHook (  
    void )
```

Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.

Returns

none.

7.565.2.4 CopServiceHook()

```
void CopServiceHook (  
    void )
```

Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.

Returns

none.

7.565.2.5 CpuUserProgramStartHook()

```
blt_bool CpuUserProgramStartHook (  
    void )
```

Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.

Returns

BLT_TRUE if it is okay to start the user program, BLT_FALSE to keep keep the bootloader active.

7.565.2.6 NvmDoneHook()

```
blt_bool NvmDoneHook (  
    void )
```

Callback that gets called at the end of the NVM programming session.

Returns

BLT_TRUE is successful, BLT_FALSE otherwise.

7.565.2.7 NvmEraseHook()

```
blt_int8u NvmEraseHook (
    blt_addr addr,
    blt_int32u len )
```

Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the memory hasn't been erased yet.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.

Returns

BLT_NVM_OKAY if successful, BLT_NVM_NOT_IN_RANGE if the address is not within the supported memory range, or BLT_NVM_ERROR if the erase operation failed.

7.565.2.8 NvmInitHook()

```
void NvmInitHook (
    void )
```

Callback that gets called at the start of the internal NVM driver initialization routine.

Returns

none.

7.565.2.9 NvmReinitHook()

```
void NvmReinitHook (
    void )
```

Callback that gets called at the start of a firmware update to reinitialize the NVM driver.

Returns

none.

7.565.2.10 NvmWriteHook()

```
blt_int8u NvmWriteHook (
    blt_addr addr,
    blt_int32u len,
    blt_int8u * data )
```

Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the data hasn't been written yet.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.
<i>data</i>	Pointer to the data buffer.

Returns

BLT_NVM_OKAY if successful, BLT_NVM_NOT_IN_RANGE if the address is not within the supported memory range, or BLT_NVM_ERROR if the write operation failed.

7.565.2.11 UsbConnectHook()

```
void UsbConnectHook (
    blt_bool connect )
```

Callback that gets called whenever the USB device should be connected to the USB bus.

Parameters

<i>connect</i>	BLT_TRUE to connect and BLT_FALSE to disconnect.
----------------	--

Returns

none.

7.565.2.12 UsbEnterLowPowerModeHook()

```
void UsbEnterLowPowerModeHook (
    void )
```

Callback that gets called whenever the USB host requests the device to enter a low power mode.

Returns

none.

7.565.2.13 UsbLeaveLowPowerModeHook()

```
void UsbLeaveLowPowerModeHook (
    void )
```

Callback that gets called whenever the USB host requests the device to exit low power mode.

Returns

none.

7.565.2.14 XcpGetSeedHook()

```
blt_int8u XcpGetSeedHook (
    blt_int8u resource,
    blt_int8u * seed )
```

Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the GET_SEED command.

Parameters

<i>resource</i>	Resource that the seed is requested for (XCP_RES_XXX).
<i>seed</i>	Pointer to byte buffer where the seed will be stored.

Returns

Length of the seed in bytes.

7.565.2.15 XcpVerifyKeyHook()

```
blt_int8u XcpVerifyKeyHook (
    blt_int8u resource,
    blt_int8u * key,
    blt_int8u len )
```

Called by the UNLOCK command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

Parameters

<i>resource</i>	resource to unlock (XCP_RES_XXX).
<i>key</i>	pointer to the byte buffer holding the key.
<i>len</i>	length of the key in bytes.

Returns

1 if the key was correct, 0 otherwise.

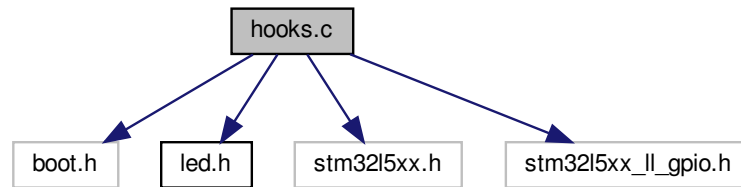
7.566 hooks.c File Reference

Bootloader callback source file.

```
#include "boot.h"
#include "led.h"
#include "stm32l5xx.h"
```

```
#include "stm32l5xx_ll_gpio.h"
```

Include dependency graph for ARMCM33_STM32L5_Nucleo_L552ZE_Keil/Boot/hooks.c:



Functions

- void [BackDoorInitHook](#) (void)
Initializes the backdoor entry option.
- [blt_bool BackDoorEntryHook](#) (void)
Checks if a backdoor entry is requested.
- [blt_bool CpuUserProgramStartHook](#) (void)
Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.
- void [CopInitHook](#) (void)
Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.
- void [CopServiceHook](#) (void)
Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.
- void [UsbConnectHook](#) ([blt_bool](#) connect)
Callback that gets called whenever the USB device should be connected to the USB bus.
- void [UsbEnterLowPowerModeHook](#) (void)
Callback that gets called whenever the USB host requests the device to enter a low power mode.
- void [UsbLeaveLowPowerModeHook](#) (void)
Callback that gets called whenever the USB host requests the device to exit low power mode.
- void [NvmInitHook](#) (void)
Callback that gets called at the start of the internal NVM driver initialization routine.
- void [NvmReinitHook](#) (void)
Callback that gets called at the start of a firmware update to reinitialize the NVM driver.
- [blt_int8u NvmWriteHook](#) ([blt_addr](#) addr, [blt_int32u](#) len, [blt_int8u](#) *data)
Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the data hasn't been written yet.
- [blt_int8u NvmEraseHook](#) ([blt_addr](#) addr, [blt_int32u](#) len)
Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the memory hasn't been erased yet.
- [blt_bool NvmDoneHook](#) (void)
Callback that gets called at the end of the NVM programming session.
- [blt_int8u XcpGetSeedHook](#) ([blt_int8u](#) resource, [blt_int8u](#) *seed)

Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the GET_SEED command.

- `blt_int8u XcpVerifyKeyHook (blt_int8u resource, blt_int8u *key, blt_int8u len)`

Called by the UNLOCK command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

7.566.1 Detailed Description

Bootloader callback source file.

7.566.2 Function Documentation

7.566.2.1 BackDoorEntryHook()

```
blt_bool BackDoorEntryHook (
    void )
```

Checks if a backdoor entry is requested.

Returns

BLT_TRUE if the backdoor entry is requested, BLT_FALSE otherwise.

Referenced by `UsbLeaveLowPowerModeHook()`.

7.566.2.2 BackDoorInitHook()

```
void BackDoorInitHook (
    void )
```

Initializes the backdoor entry option.

Returns

none.

Referenced by `UsbLeaveLowPowerModeHook()`.

7.566.2.3 CopInitHook()

```
void CopInitHook (
    void )
```

Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.

Returns

none.

7.566.2.4 CopServiceHook()

```
void CopServiceHook (
    void )
```

Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.

Returns

none.

7.566.2.5 CpuUserProgramStartHook()

```
blt_bool CpuUserProgramStartHook (
    void )
```

Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.

Returns

BLT_TRUE if it is okay to start the user program, BLT_FALSE to keep keep the bootloader active.

7.566.2.6 NvmDoneHook()

```
blt_bool NvmDoneHook (
    void )
```

Callback that gets called at the end of the NVM programming session.

Returns

BLT_TRUE is successful, BLT_FALSE otherwise.

7.566.2.7 NvmEraseHook()

```
blt_int8u NvmEraseHook (
    blt_addr addr,
    blt_int32u len )
```

Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the memory hasn't been erased yet.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.

Returns

BLT_NVM_OKAY if successful, BLT_NVM_NOT_IN_RANGE if the address is not within the supported memory range, or BLT_NVM_ERROR if the erase operation failed.

7.566.2.8 NvmInitHook()

```
void NvmInitHook (
    void )
```

Callback that gets called at the start of the internal NVM driver initialization routine.

Returns

none.

7.566.2.9 NvmReinitHook()

```
void NvmReinitHook (
    void )
```

Callback that gets called at the start of a firmware update to reinitialize the NVM driver.

Returns

none.

7.566.2.10 NvmWriteHook()

```
blt_int8u NvmWriteHook (
    blt_addr addr,
    blt_int32u len,
    blt_int8u * data )
```

Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the data hasn't been written yet.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.
<i>data</i>	Pointer to the data buffer.

Returns

BLT_NVM_OKAY if successful, BLT_NVM_NOT_IN_RANGE if the address is not within the supported memory range, or BLT_NVM_ERROR if the write operation failed.

7.566.2.11 UsbConnectHook()

```
void UsbConnectHook (
    blt_bool connect )
```

Callback that gets called whenever the USB device should be connected to the USB bus.

Parameters

<i>connect</i>	BLT_TRUE to connect and BLT_FALSE to disconnect.
----------------	--

Returns

none.

7.566.2.12 UsbEnterLowPowerModeHook()

```
void UsbEnterLowPowerModeHook (
    void )
```

Callback that gets called whenever the USB host requests the device to enter a low power mode.

Returns

none.

7.566.2.13 UsbLeaveLowPowerModeHook()

```
void UsbLeaveLowPowerModeHook (
    void )
```

Callback that gets called whenever the USB host requests the device to exit low power mode.

Returns

none.

7.566.2.14 XcpGetSeedHook()

```
blt_int8u XcpGetSeedHook (
    blt_int8u resource,
    blt_int8u * seed )
```

Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the GET_SEED command.

Parameters

<i>resource</i>	Resource that the seed is requested for (XCP_RES_XXX).
<i>seed</i>	Pointer to byte buffer where the seed will be stored.

Returns

Length of the seed in bytes.

7.566.2.15 XcpVerifyKeyHook()

```
blt_int8u XcpVerifyKeyHook (
    blt_int8u resource,
    blt_int8u * key,
    blt_int8u len )
```

Called by the UNLOCK command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

Parameters

<i>resource</i>	resource to unlock (XCP_RES_XXX).
<i>key</i>	pointer to the byte buffer holding the key.
<i>len</i>	length of the key in bytes.

Returns

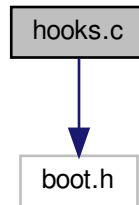
1 if the key was correct, 0 otherwise.

7.567 hooks.c File Reference

Bootloader callback source file.

```
#include "boot.h"
```

Include dependency graph for ARMCM3_EFM32_Olimex_EM32G880F128STK_GCC/Boot/hooks.c:



Functions

- void [BackDoorInitHook](#) (void)
Initializes the backdoor entry option.
- [blt_bool](#) [BackDoorEntryHook](#) (void)
Checks if a backdoor entry is requested.
- [blt_bool](#) [CpuUserProgramStartHook](#) (void)
Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.
- void [NvmInitHook](#) (void)
Callback that gets called at the start of the internal NVM driver initialization routine.
- void [NvmReinitHook](#) (void)
Callback that gets called at the start of a firmware update to reinitialize the NVM driver.
- [blt_int8u](#) [NvmWriteHook](#) ([blt_addr](#) addr, [blt_int32u](#) len, [blt_int8u](#) *data)
Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the data hasn't been written yet.
- [blt_int8u](#) [NvmEraseHook](#) ([blt_addr](#) addr, [blt_int32u](#) len)
Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the memory hasn't been erased yet.
- [blt_bool](#) [NvmDoneHook](#) (void)
Callback that gets called at the end of the NVM programming session.
- void [CopInitHook](#) (void)
Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.
- void [CopServiceHook](#) (void)
Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.
- [blt_int8u](#) [XcpGetSeedHook](#) ([blt_int8u](#) resource, [blt_int8u](#) *seed)
Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the GET_SEED command.
- [blt_int8u](#) [XcpVerifyKeyHook](#) ([blt_int8u](#) resource, [blt_int8u](#) *key, [blt_int8u](#) len)
Called by the UNLOCK command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

7.567.1 Detailed Description

Bootloader callback source file.

7.567.2 Function Documentation

7.567.2.1 BackDoorEntryHook()

```
blt_bool BackDoorEntryHook (  
    void )
```

Checks if a backdoor entry is requested.

Returns

BLT_TRUE if the backdoor entry is requested, BLT_FALSE otherwise.

7.567.2.2 BackDoorInitHook()

```
void BackDoorInitHook (  
    void )
```

Initializes the backdoor entry option.

Returns

none.

7.567.2.3 CopInitHook()

```
void CopInitHook (  
    void )
```

Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.

Returns

none.

7.567.2.4 CopServiceHook()

```
void CopServiceHook (
    void )
```

Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.

Returns

none.

7.567.2.5 CpuUserProgramStartHook()

```
blt_bool CpuUserProgramStartHook (
    void )
```

Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.

Returns

BLT_TRUE if it is okay to start the user program, BLT_FALSE to keep keep the bootloader active.

7.567.2.6 NvmDoneHook()

```
blt_bool NvmDoneHook (
    void )
```

Callback that gets called at the end of the NVM programming session.

Returns

BLT_TRUE is successful, BLT_FALSE otherwise.

7.567.2.7 NvmEraseHook()

```
blt_int8u NvmEraseHook (
    blt_addr addr,
    blt_int32u len )
```

Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the memory hasn't been erased yet.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.

Returns

BLT_NVM_OKAY if successful, BLT_NVM_NOT_IN_RANGE if the address is not within the supported memory range, or BLT_NVM_ERROR if the erase operation failed.

7.567.2.8 NvmInitHook()

```
void NvmInitHook (  
    void )
```

Callback that gets called at the start of the internal NVM driver initialization routine.

Returns

none.

7.567.2.9 NvmReinitHook()

```
void NvmReinitHook (  
    void )
```

Callback that gets called at the start of a firmware update to reinitialize the NVM driver.

Returns

none.

7.567.2.10 NvmWriteHook()

```
blt_int8u NvmWriteHook (  
    blt_addr addr,  
    blt_int32u len,  
    blt_int8u * data )
```

Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the data hasn't been written yet.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.
<i>data</i>	Pointer to the data buffer.

Returns

BLT_NVM_OKAY if successful, BLT_NVM_NOT_IN_RANGE if the address is not within the supported memory range, or BLT_NVM_ERROR if the write operation failed.

7.567.2.11 XcpGetSeedHook()

```
blt_int8u XcpGetSeedHook (
    blt_int8u resource,
    blt_int8u * seed )
```

Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the GET_SEED command.

Parameters

<i>resource</i>	Resource that the seed is requested for (XCP_RES_XXX).
<i>seed</i>	Pointer to byte buffer where the seed will be stored.

Returns

Length of the seed in bytes.

7.567.2.12 XcpVerifyKeyHook()

```
blt_int8u XcpVerifyKeyHook (
    blt_int8u resource,
    blt_int8u * key,
    blt_int8u len )
```

Called by the UNLOCK command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

Parameters

<i>resource</i>	resource to unlock (XCP_RES_XXX).
<i>key</i>	pointer to the byte buffer holding the key.
<i>len</i>	length of the key in bytes.

Returns

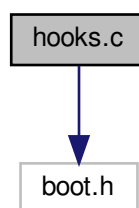
1 if the key was correct, 0 otherwise.

7.568 hooks.c File Reference

Bootloader callback source file.

```
#include "boot.h"
```

Include dependency graph for ARMC32_EFM32_Olimex_EM32G880F128STK_IAR/Boot/hooks.c:



Functions

- void [BackDoorInitHook](#) (void)
Initializes the backdoor entry option.
- [blt_bool BackDoorEntryHook](#) (void)
Checks if a backdoor entry is requested.
- [blt_bool CpuUserProgramStartHook](#) (void)
Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.
- void [NvmInitHook](#) (void)
Callback that gets called at the start of the internal NVM driver initialization routine.
- void [NvmReinitHook](#) (void)
Callback that gets called at the start of a firmware update to reinitialize the NVM driver.
- [blt_int8u NvmWriteHook](#) ([blt_addr](#) addr, [blt_int32u](#) len, [blt_int8u](#) *data)
Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the data hasn't been written yet.
- [blt_int8u NvmEraseHook](#) ([blt_addr](#) addr, [blt_int32u](#) len)
Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the memory hasn't been erased yet.
- [blt_bool NvmDoneHook](#) (void)
Callback that gets called at the end of the NVM programming session.
- void [CopInitHook](#) (void)
Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.

- void [CopServiceHook](#) (void)
Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.
- [blt_int8u XcpGetSeedHook](#) ([blt_int8u](#) resource, [blt_int8u](#) *seed)
Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the GET_SEED command.
- [blt_int8u XcpVerifyKeyHook](#) ([blt_int8u](#) resource, [blt_int8u](#) *key, [blt_int8u](#) len)
Called by the UNLOCK command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

7.568.1 Detailed Description

Bootloader callback source file.

7.568.2 Function Documentation

7.568.2.1 BackDoorEntryHook()

```
blt_bool BackDoorEntryHook (  
    void )
```

Checks if a backdoor entry is requested.

Returns

BLT_TRUE if the backdoor entry is requested, BLT_FALSE otherwise.

7.568.2.2 BackDoorInitHook()

```
void BackDoorInitHook (  
    void )
```

Initializes the backdoor entry option.

Returns

none.

7.568.2.3 CopInitHook()

```
void CopInitHook (
    void )
```

Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.

Returns

none.

7.568.2.4 CopServiceHook()

```
void CopServiceHook (
    void )
```

Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.

Returns

none.

7.568.2.5 CpuUserProgramStartHook()

```
blt_bool CpuUserProgramStartHook (
    void )
```

Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.

Returns

BLT_TRUE if it is okay to start the user program, BLT_FALSE to keep keep the bootloader active.

7.568.2.6 NvmDoneHook()

```
blt_bool NvmDoneHook (
    void )
```

Callback that gets called at the end of the NVM programming session.

Returns

BLT_TRUE is successful, BLT_FALSE otherwise.

7.568.2.7 NvmEraseHook()

```
blt_int8u NvmEraseHook (
    blt_addr addr,
    blt_int32u len )
```

Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the memory hasn't been erased yet.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.

Returns

BLT_NVM_OKAY if successful, BLT_NVM_NOT_IN_RANGE if the address is not within the supported memory range, or BLT_NVM_ERROR if the erase operation failed.

7.568.2.8 NvmInitHook()

```
void NvmInitHook (
    void )
```

Callback that gets called at the start of the internal NVM driver initialization routine.

Returns

none.

7.568.2.9 NvmReinitHook()

```
void NvmReinitHook (
    void )
```

Callback that gets called at the start of a firmware update to reinitialize the NVM driver.

Returns

none.

7.568.2.10 NvmWriteHook()

```
blt_int8u NvmWriteHook (
    blt_addr addr,
    blt_int32u len,
    blt_int8u * data )
```

Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the data hasn't been written yet.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.
<i>data</i>	Pointer to the data buffer.

Returns

BLT_NVM_OKAY if successful, BLT_NVM_NOT_IN_RANGE if the address is not within the supported memory range, or BLT_NVM_ERROR if the write operation failed.

7.568.2.11 XcpGetSeedHook()

```
blt_int8u XcpGetSeedHook (
    blt_int8u resource,
    blt_int8u * seed )
```

Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the GET_SEED command.

Parameters

<i>resource</i>	Resource that the seed is requested for (XCP_RES_XXX).
<i>seed</i>	Pointer to byte buffer where the seed will be stored.

Returns

Length of the seed in bytes.

7.568.2.12 XcpVerifyKeyHook()

```
blt_int8u XcpVerifyKeyHook (
    blt_int8u resource,
    blt_int8u * key,
    blt_int8u len )
```

Called by the UNLOCK command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

Parameters

<i>resource</i>	resource to unlock (XCP_RES_XXX).
<i>key</i>	pointer to the byte buffer holding the key.
<i>len</i>	length of the key in bytes.

Returns

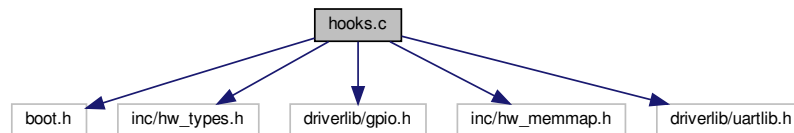
1 if the key was correct, 0 otherwise.

7.569 hooks.c File Reference

Bootloader callback source file.

```
#include "boot.h"
#include "inc/hw_types.h"
#include "driverlib/gpio.h"
#include "inc/hw_memmap.h"
#include "driverlib/uartlib.h"
```

Include dependency graph for ARMC3_LM3S_EK_LM3S6965_GCC/Boot/hooks.c:



Functions

- void [BackDoorInitHook](#) (void)
Initializes the backdoor entry option.
- [blt_bool](#) [BackDoorEntryHook](#) (void)
Checks if a backdoor entry is requested.
- [blt_bool](#) [CpuUserProgramStartHook](#) (void)
Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.
- void [NvmInitHook](#) (void)
Callback that gets called at the start of the internal NVM driver initialization routine.
- void [NvmReinitHook](#) (void)
Callback that gets called at the start of a firmware update to reinitialize the NVM driver.
- [blt_int8u](#) [NvmWriteHook](#) ([blt_addr](#) addr, [blt_int32u](#) len, [blt_int8u](#) *data)
Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the data hasn't been written yet.
- [blt_int8u](#) [NvmEraseHook](#) ([blt_addr](#) addr, [blt_int32u](#) len)
Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the memory hasn't been erased yet.
- [blt_bool](#) [NvmDoneHook](#) (void)
Callback that gets called at the end of the NVM programming session.
- void [CopInitHook](#) (void)
Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.
- void [CopServiceHook](#) (void)

Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.

- [blt_bool FileIsFirmwareUpdateRequestedHook](#) (void)

Callback that gets called to check whether a firmware update from local file storage should be started. This could for example be when a switch is pressed, when a certain file is found on the local file storage, etc.

- const [blt_char](#) * [FileGetFirmwareFilenameHook](#) (void)

Callback to obtain the filename of the firmware file that should be used during the firmware update from the local file storage. This hook function is called at the beginning of the firmware update from local storage sequence.

- void [FileFirmwareUpdateStartedHook](#) (void)

Callback that gets called to inform the application that a firmware update from local storage just started.

- void [FileFirmwareUpdateCompletedHook](#) (void)

Callback that gets called to inform the application that a firmware update was successfully completed.

- void [FileFirmwareUpdateErrorHook](#) ([blt_int8u](#) error_code)

Callback that gets called in case an error occurred during a firmware update. Refer to [<file.h>](#) for a list of available error codes.

- void [FileFirmwareUpdateLogHook](#) ([blt_char](#) *info_string)

Callback that gets called each time new log information becomes available during a firmware update.

- [blt_int8u](#) [XcpGetSeedHook](#) ([blt_int8u](#) resource, [blt_int8u](#) *seed)

Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the GET_SEED command.

- [blt_int8u](#) [XcpVerifyKeyHook](#) ([blt_int8u](#) resource, [blt_int8u](#) *key, [blt_int8u](#) len)

Called by the UNLOCK command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

Variables

- static const [blt_char](#) [firmwareFilename](#) [] = "/demoprogram_ek_lm3s6965.srec"

Firmware filename.

-

```
struct {
    FIL handle
    blt\_bool canUse
} logfile
```

Data structure for grouping log-file related information.

7.569.1 Detailed Description

Bootloader callback source file.

7.569.2 Function Documentation

7.569.2.1 BackDoorEntryHook()

```
blt_bool BackDoorEntryHook (  
    void )
```

Checks if a backdoor entry is requested.

Returns

BLT_TRUE if the backdoor entry is requested, BLT_FALSE otherwise.

7.569.2.2 BackDoorInitHook()

```
void BackDoorInitHook (  
    void )
```

Initializes the backdoor entry option.

Returns

none.

7.569.2.3 CopInitHook()

```
void CopInitHook (  
    void )
```

Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.

Returns

none.

7.569.2.4 CopServiceHook()

```
void CopServiceHook (  
    void )
```

Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.

Returns

none.

7.569.2.5 CpuUserProgramStartHook()

```
blt_bool CpuUserProgramStartHook (
    void )
```

Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.

Returns

BLT_TRUE if it is okay to start the user program, BLT_FALSE to keep keep the bootloader active.

7.569.2.6 FileFirmwareUpdateCompletedHook()

```
void FileFirmwareUpdateCompletedHook (
    void )
```

Callback that gets called to inform the application that a firmware update was successfully completed.

Returns

none.

7.569.2.7 FileFirmwareUpdateErrorHook()

```
void FileFirmwareUpdateErrorHook (
    blt_int8u error_code )
```

Callback that gets called in case an error occurred during a firmware update. Refer to <[file.h](#)> for a list of available error codes.

Returns

none.

7.569.2.8 FileFirmwareUpdateLogHook()

```
void FileFirmwareUpdateLogHook (
    blt_char * info_string )
```

Callback that gets called each time new log information becomes available during a firmware update.

Parameters

<i>info_string</i>	Pointer to a character array with the log entry info.
--------------------	---

Returns

none.

7.569.2.9 FileFirmwareUpdateStartedHook()

```
void FileFirmwareUpdateStartedHook (  
    void )
```

Callback that gets called to inform the application that a firmware update from local storage just started.

Returns

none.

7.569.2.10 FileGetFirmwareFilenameHook()

```
const blt_char* FileGetFirmwareFilenameHook (  
    void )
```

Callback to obtain the filename of the firmware file that should be used during the firmware update from the local file storage. This hook function is called at the beginning of the firmware update from local storage sequence.

Returns

valid firmware filename with full path or BLT_NULL.

7.569.2.11 FileIsFirmwareUpdateRequestedHook()

```
blt_bool FileIsFirmwareUpdateRequestedHook (  
    void )
```

Callback that gets called to check whether a firmware update from local file storage should be started. This could for example be when a switch is pressed, when a certain file is found on the local file storage, etc.

Returns

BLT_TRUE if a firmware update is requested, BLT_FALSE otherwise.

7.569.2.12 NvmDoneHook()

```
blt_bool NvmDoneHook (
    void )
```

Callback that gets called at the end of the NVM programming session.

Returns

BLT_TRUE is successful, BLT_FALSE otherwise.

7.569.2.13 NvmEraseHook()

```
blt_int8u NvmEraseHook (
    blt_addr addr,
    blt_int32u len )
```

Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the memory hasn't been erased yet.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.

Returns

BLT_NVM_OKAY if successful, BLT_NVM_NOT_IN_RANGE if the address is not within the supported memory range, or BLT_NVM_ERROR if the erase operation failed.

7.569.2.14 NvmInitHook()

```
void NvmInitHook (
    void )
```

Callback that gets called at the start of the internal NVM driver initialization routine.

Returns

none.

7.569.2.15 NvmReinitHook()

```
void NvmReinitHook (  
    void )
```

Callback that gets called at the start of a firmware update to reinitialize the NVM driver.

Returns

none.

7.569.2.16 NvmWriteHook()

```
blt_int8u NvmWriteHook (  
    blt_addr addr,  
    blt_int32u len,  
    blt_int8u * data )
```

Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the data hasn't been written yet.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.
<i>data</i>	Pointer to the data buffer.

Returns

BLT_NVM_OKAY if successful, BLT_NVM_NOT_IN_RANGE if the address is not within the supported memory range, or BLT_NVM_ERROR if the write operation failed.

7.569.2.17 XcpGetSeedHook()

```
blt_int8u XcpGetSeedHook (  
    blt_int8u resource,  
    blt_int8u * seed )
```

Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the GET_SEED command.

Parameters

<i>resource</i>	Resource that the seed is requested for (XCP_RES_XXX).
<i>seed</i>	Pointer to byte buffer where the seed will be stored.

Returns

Length of the seed in bytes.

7.569.2.18 XcpVerifyKeyHook()

```
blt_int8u XcpVerifyKeyHook (
    blt_int8u resource,
    blt_int8u * key,
    blt_int8u len )
```

Called by the UNLOCK command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

Parameters

<i>resource</i>	resource to unlock (XCP_RES_XXX).
<i>key</i>	pointer to the byte buffer holding the key.
<i>len</i>	length of the key in bytes.

Returns

1 if the key was correct, 0 otherwise.

7.569.3 Variable Documentation**7.569.3.1 canUse**

```
blt_bool canUse
```

Flag to indicate if the log-file can be used.

7.569.3.2 handle

```
FIL handle
```

FatFS handle to the log-file.

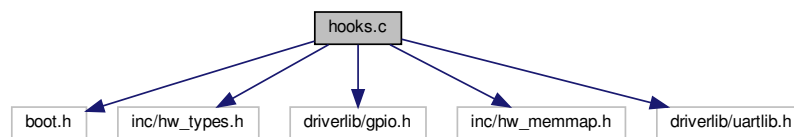
Referenced by UsbFifoMgrRead(), UsbFifoMgrScan(), and UsbFifoMgrWrite().

7.570 hooks.c File Reference

Bootloader callback source file.

```
#include "boot.h"
#include "inc/hw_types.h"
#include "driverlib/gpio.h"
#include "inc/hw_memmap.h"
#include "driverlib/uartlib.h"
```

Include dependency graph for ARMCM3_LM3S_EK_LM3S6965_IAR/Boot/hooks.c:



Functions

- void [BackDoorInitHook](#) (void)
Initializes the backdoor entry option.
- [blt_bool](#) [BackDoorEntryHook](#) (void)
Checks if a backdoor entry is requested.
- [blt_bool](#) [CpuUserProgramStartHook](#) (void)
Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.
- void [NvmInitHook](#) (void)
Callback that gets called at the start of the internal NVM driver initialization routine.
- void [NvmReinitHook](#) (void)
Callback that gets called at the start of a firmware update to reinitialize the NVM driver.
- [blt_int8u](#) [NvmWriteHook](#) ([blt_addr](#) addr, [blt_int32u](#) len, [blt_int8u](#) *data)
Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the data hasn't been written yet.
- [blt_int8u](#) [NvmEraseHook](#) ([blt_addr](#) addr, [blt_int32u](#) len)
Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the memory hasn't been erased yet.
- [blt_bool](#) [NvmDoneHook](#) (void)
Callback that gets called at the end of the NVM programming session.
- void [CopInitHook](#) (void)
Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.
- void [CopServiceHook](#) (void)
Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.
- [blt_bool](#) [FileIsFirmwareUpdateRequestedHook](#) (void)

Callback that gets called to check whether a firmware update from local file storage should be started. This could for example be when a switch is pressed, when a certain file is found on the local file storage, etc.

- const `blt_char * FileGetFirmwareFilenameHook` (void)

Callback to obtain the filename of the firmware file that should be used during the firmware update from the local file storage. This hook function is called at the beginning of the firmware update from local storage sequence.

- void `FileFirmwareUpdateStartedHook` (void)

Callback that gets called to inform the application that a firmware update from local storage just started.

- void `FileFirmwareUpdateCompletedHook` (void)

Callback that gets called to inform the application that a firmware update was successfully completed.

- void `FileFirmwareUpdateErrorHook` (`blt_int8u` error_code)

Callback that gets called in case an error occurred during a firmware update. Refer to [<file.h>](#) for a list of available error codes.

- void `FileFirmwareUpdateLogHook` (`blt_char *info_string`)

Callback that gets called each time new log information becomes available during a firmware update.

- `blt_int8u XcpGetSeedHook` (`blt_int8u` resource, `blt_int8u *seed`)

Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the `GET_SEED` command.

- `blt_int8u XcpVerifyKeyHook` (`blt_int8u` resource, `blt_int8u *key`, `blt_int8u len`)

Called by the `UNLOCK` command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

Variables

- static const `blt_char firmwareFilename` [] = `"/demoprogram_ek_lm3s6965.srec"`

Firmware filename.

-

```
struct {
    FIL handle
    blt_bool canUse
} logfile
```

Data structure for grouping log-file related information.

7.570.1 Detailed Description

Bootloader callback source file.

7.570.2 Function Documentation

7.570.2.1 BackDoorEntryHook()

```
blt_bool BackDoorEntryHook (
    void )
```

Checks if a backdoor entry is requested.

Returns

BLT_TRUE if the backdoor entry is requested, BLT_FALSE otherwise.

7.570.2.2 BackDoorInitHook()

```
void BackDoorInitHook (  
    void )
```

Initializes the backdoor entry option.

Returns

none.

7.570.2.3 CopInitHook()

```
void CopInitHook (  
    void )
```

Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.

Returns

none.

7.570.2.4 CopServiceHook()

```
void CopServiceHook (  
    void )
```

Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.

Returns

none.

7.570.2.5 CpuUserProgramStartHook()

```
blt_bool CpuUserProgramStartHook (  
    void )
```

Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.

Returns

BLT_TRUE if it is okay to start the user program, BLT_FALSE to keep keep the bootloader active.

7.570.2.6 FileFirmwareUpdateCompletedHook()

```
void FileFirmwareUpdateCompletedHook (  
    void )
```

Callback that gets called to inform the application that a firmware update was successfully completed.

Returns

none.

7.570.2.7 FileFirmwareUpdateErrorHook()

```
void FileFirmwareUpdateErrorHook (  
    blt_int8u error_code )
```

Callback that gets called in case an error occurred during a firmware update. Refer to <[file.h](#)> for a list of available error codes.

Returns

none.

7.570.2.8 FileFirmwareUpdateLogHook()

```
void FileFirmwareUpdateLogHook (  
    blt_char * info_string )
```

Callback that gets called each time new log information becomes available during a firmware update.

Parameters

<i>info_string</i>	Pointer to a character array with the log entry info.
--------------------	---

Returns

none.

7.570.2.9 FileFirmwareUpdateStartedHook()

```
void FileFirmwareUpdateStartedHook (  
    void )
```

Callback that gets called to inform the application that a firmware update from local storage just started.

Returns

none.

7.570.2.10 FileGetFirmwareFilenameHook()

```
const blt_char* FileGetFirmwareFilenameHook (  
    void )
```

Callback to obtain the filename of the firmware file that should be used during the firmware update from the local file storage. This hook function is called at the beginning of the firmware update from local storage sequence.

Returns

valid firmware filename with full path or BLT_NULL.

7.570.2.11 FileIsFirmwareUpdateRequestedHook()

```
blt_bool FileIsFirmwareUpdateRequestedHook (  
    void )
```

Callback that gets called to check whether a firmware update from local file storage should be started. This could for example be when a switch is pressed, when a certain file is found on the local file storage, etc.

Returns

BLT_TRUE if a firmware update is requested, BLT_FALSE otherwise.

7.570.2.12 NvmDoneHook()

```
blt_bool NvmDoneHook (  
    void )
```

Callback that gets called at the end of the NVM programming session.

Returns

BLT_TRUE is successful, BLT_FALSE otherwise.

7.570.2.13 NvmEraseHook()

```
blt_int8u NvmEraseHook (  
    blt_addr addr,  
    blt_int32u len )
```

Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the memory hasn't been erased yet.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.

Returns

BLT_NVM_OKAY if successful, BLT_NVM_NOT_IN_RANGE if the address is not within the supported memory range, or BLT_NVM_ERROR if the erase operation failed.

7.570.2.14 NvmInitHook()

```
void NvmInitHook (  
    void )
```

Callback that gets called at the start of the internal NVM driver initialization routine.

Returns

none.

7.570.2.15 NvmReinitHook()

```
void NvmReinitHook (  
    void )
```

Callback that gets called at the start of a firmware update to reinitialize the NVM driver.

Returns

none.

7.570.2.16 NvmWriteHook()

```
blt_int8u NvmWriteHook (  
    blt_addr addr,  
    blt_int32u len,  
    blt_int8u * data )
```

Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the data hasn't been written yet.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.
<i>data</i>	Pointer to the data buffer.

Returns

BLT_NVM_OKAY if successful, BLT_NVM_NOT_IN_RANGE if the address is not within the supported memory range, or BLT_NVM_ERROR if the write operation failed.

7.570.2.17 XcpGetSeedHook()

```
blt_int8u XcpGetSeedHook (
    blt_int8u resource,
    blt_int8u * seed )
```

Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the GET_SEED command.

Parameters

<i>resource</i>	Resource that the seed is requested for (XCP_RES_XXX).
<i>seed</i>	Pointer to byte buffer where the seed will be stored.

Returns

Length of the seed in bytes.

7.570.2.18 XcpVerifyKeyHook()

```
blt_int8u XcpVerifyKeyHook (
    blt_int8u resource,
    blt_int8u * key,
    blt_int8u len )
```

Called by the UNLOCK command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

Parameters

<i>resource</i>	resource to unlock (XCP_RES_XXX).
<i>key</i>	pointer to the byte buffer holding the key.
<i>len</i>	length of the key in bytes.

Returns

1 if the key was correct, 0 otherwise.

7.570.3 Variable Documentation**7.570.3.1 canUse**

`blt_bool canUse`

Flag to indicate if the log-file can be used.

7.570.3.2 handle

`FIL handle`

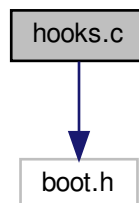
FatFS handle to the log-file.

7.571 hooks.c File Reference

Bootloader callback source file.

```
#include "boot.h"
```

Include dependency graph for ARMCM3_LM3S_EK_LM3S8962_GCC/Boot/hooks.c:



Functions

- void [BackDoorInitHook](#) (void)
Initializes the backdoor entry option.
- [blt_bool BackDoorEntryHook](#) (void)
Checks if a backdoor entry is requested.
- [blt_bool CpuUserProgramStartHook](#) (void)
Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.
- void [NvmInitHook](#) (void)
Callback that gets called at the start of the internal NVM driver initialization routine.
- void [NvmReinitHook](#) (void)
Callback that gets called at the start of a firmware update to reinitialize the NVM driver.
- [blt_int8u NvmWriteHook](#) ([blt_addr](#) addr, [blt_int32u](#) len, [blt_int8u](#) *data)
Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the data hasn't been written yet.
- [blt_int8u NvmEraseHook](#) ([blt_addr](#) addr, [blt_int32u](#) len)
Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the memory hasn't been erased yet.
- [blt_bool NvmDoneHook](#) (void)
Callback that gets called at the end of the NVM programming session.
- void [CopInitHook](#) (void)
Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.
- void [CopServiceHook](#) (void)
Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.
- [blt_int8u XcpGetSeedHook](#) ([blt_int8u](#) resource, [blt_int8u](#) *seed)
Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the GET_SEED command.
- [blt_int8u XcpVerifyKeyHook](#) ([blt_int8u](#) resource, [blt_int8u](#) *key, [blt_int8u](#) len)
Called by the UNLOCK command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

7.571.1 Detailed Description

Bootloader callback source file.

7.571.2 Function Documentation

7.571.2.1 BackDoorEntryHook()

```
blt_bool BackDoorEntryHook (  
    void )
```

Checks if a backdoor entry is requested.

Returns

BLT_TRUE if the backdoor entry is requested, BLT_FALSE otherwise.

7.571.2.2 BackDoorInitHook()

```
void BackDoorInitHook (  
    void )
```

Initializes the backdoor entry option.

Returns

none.

7.571.2.3 CopInitHook()

```
void CopInitHook (  
    void )
```

Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.

Returns

none.

7.571.2.4 CopServiceHook()

```
void CopServiceHook (  
    void )
```

Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.

Returns

none.

7.571.2.5 CpuUserProgramStartHook()

```
blt_bool CpuUserProgramStartHook (
    void )
```

Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.

Returns

BLT_TRUE if it is okay to start the user program, BLT_FALSE to keep keep the bootloader active.

7.571.2.6 NvmDoneHook()

```
blt_bool NvmDoneHook (
    void )
```

Callback that gets called at the end of the NVM programming session.

Returns

BLT_TRUE is successful, BLT_FALSE otherwise.

7.571.2.7 NvmEraseHook()

```
blt_int8u NvmEraseHook (
    blt_addr addr,
    blt_int32u len )
```

Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the memory hasn't been erased yet.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.

Returns

BLT_NVM_OKAY if successful, BLT_NVM_NOT_IN_RANGE if the address is not within the supported memory range, or BLT_NVM_ERROR is the erase operation failed.

7.571.2.8 NvmInitHook()

```
void NvmInitHook (
    void )
```

Callback that gets called at the start of the internal NVM driver initialization routine.

Returns

none.

7.571.2.9 NvmReinitHook()

```
void NvmReinitHook (
    void )
```

Callback that gets called at the start of a firmware update to reinitialize the NVM driver.

Returns

none.

7.571.2.10 NvmWriteHook()

```
blt_int8u NvmWriteHook (
    blt_addr addr,
    blt_int32u len,
    blt_int8u * data )
```

Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the data hasn't been written yet.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.
<i>data</i>	Pointer to the data buffer.

Returns

BLT_NVM_OKAY if successful, BLT_NVM_NOT_IN_RANGE if the address is not within the supported memory range, or BLT_NVM_ERROR if the write operation failed.

7.571.2.11 XcpGetSeedHook()

```
blt_int8u XcpGetSeedHook (
    blt_int8u resource,
    blt_int8u * seed )
```

Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the GET_SEED command.

Parameters

<i>resource</i>	Resource that the seed is requested for (XCP_RES_XXX).
<i>seed</i>	Pointer to byte buffer where the seed will be stored.

Returns

Length of the seed in bytes.

7.571.2.12 XcpVerifyKeyHook()

```
blt_int8u XcpVerifyKeyHook (
    blt_int8u resource,
    blt_int8u * key,
    blt_int8u len )
```

Called by the UNLOCK command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

Parameters

<i>resource</i>	resource to unlock (XCP_RES_XXX).
<i>key</i>	pointer to the byte buffer holding the key.
<i>len</i>	length of the key in bytes.

Returns

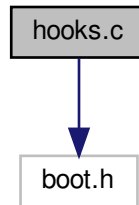
1 if the key was correct, 0 otherwise.

7.572 hooks.c File Reference

Bootloader callback source file.

```
#include "boot.h"
```

Include dependency graph for ARMCM3_LM3S_EK_LM3S8962_IAR/Boot/hooks.c:



Functions

- void [BackDoorInitHook](#) (void)
Initializes the backdoor entry option.
- [blt_bool](#) [BackDoorEntryHook](#) (void)
Checks if a backdoor entry is requested.
- [blt_bool](#) [CpuUserProgramStartHook](#) (void)
Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.
- void [NvmInitHook](#) (void)
Callback that gets called at the start of the internal NVM driver initialization routine.
- void [NvmReinitHook](#) (void)
Callback that gets called at the start of a firmware update to reinitialize the NVM driver.
- [blt_int8u](#) [NvmWriteHook](#) ([blt_addr](#) addr, [blt_int32u](#) len, [blt_int8u](#) *data)
Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the data hasn't been written yet.
- [blt_int8u](#) [NvmEraseHook](#) ([blt_addr](#) addr, [blt_int32u](#) len)
Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the memory hasn't been erased yet.
- [blt_bool](#) [NvmDoneHook](#) (void)
Callback that gets called at the end of the NVM programming session.
- void [CopInitHook](#) (void)
Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.
- void [CopServiceHook](#) (void)
Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.
- [blt_int8u](#) [XcpGetSeedHook](#) ([blt_int8u](#) resource, [blt_int8u](#) *seed)
Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the GET_SEED command.
- [blt_int8u](#) [XcpVerifyKeyHook](#) ([blt_int8u](#) resource, [blt_int8u](#) *key, [blt_int8u](#) len)
Called by the UNLOCK command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

7.572.1 Detailed Description

Bootloader callback source file.

7.572.2 Function Documentation

7.572.2.1 BackDoorEntryHook()

```
blt_bool BackDoorEntryHook (  
    void )
```

Checks if a backdoor entry is requested.

Returns

BLT_TRUE if the backdoor entry is requested, BLT_FALSE otherwise.

7.572.2.2 BackDoorInitHook()

```
void BackDoorInitHook (  
    void )
```

Initializes the backdoor entry option.

Returns

none.

7.572.2.3 CopInitHook()

```
void CopInitHook (  
    void )
```

Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.

Returns

none.

7.572.2.4 CopServiceHook()

```
void CopServiceHook (
    void )
```

Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.

Returns

none.

7.572.2.5 CpuUserProgramStartHook()

```
blt_bool CpuUserProgramStartHook (
    void )
```

Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.

Returns

BLT_TRUE if it is okay to start the user program, BLT_FALSE to keep keep the bootloader active.

7.572.2.6 NvmDoneHook()

```
blt_bool NvmDoneHook (
    void )
```

Callback that gets called at the end of the NVM programming session.

Returns

BLT_TRUE is successful, BLT_FALSE otherwise.

7.572.2.7 NvmEraseHook()

```
blt_int8u NvmEraseHook (
    blt_addr addr,
    blt_int32u len )
```

Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the memory hasn't been erased yet.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.

Returns

BLT_NVM_OKAY if successful, BLT_NVM_NOT_IN_RANGE if the address is not within the supported memory range, or BLT_NVM_ERROR if the erase operation failed.

7.572.2.8 NvmInitHook()

```
void NvmInitHook (  
    void )
```

Callback that gets called at the start of the internal NVM driver initialization routine.

Returns

none.

7.572.2.9 NvmReinitHook()

```
void NvmReinitHook (  
    void )
```

Callback that gets called at the start of a firmware update to reinitialize the NVM driver.

Returns

none.

7.572.2.10 NvmWriteHook()

```
blt_int8u NvmWriteHook (  
    blt_addr addr,  
    blt_int32u len,  
    blt_int8u * data )
```

Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the data hasn't been written yet.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.
<i>data</i>	Pointer to the data buffer.

Returns

BLT_NVM_OKAY if successful, BLT_NVM_NOT_IN_RANGE if the address is not within the supported memory range, or BLT_NVM_ERROR if the write operation failed.

7.572.2.11 XcpGetSeedHook()

```
blt_int8u XcpGetSeedHook (
    blt_int8u resource,
    blt_int8u * seed )
```

Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the GET_SEED command.

Parameters

<i>resource</i>	Resource that the seed is requested for (XCP_RES_XXX).
<i>seed</i>	Pointer to byte buffer where the seed will be stored.

Returns

Length of the seed in bytes.

7.572.2.12 XcpVerifyKeyHook()

```
blt_int8u XcpVerifyKeyHook (
    blt_int8u resource,
    blt_int8u * key,
    blt_int8u len )
```

Called by the UNLOCK command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

Parameters

<i>resource</i>	resource to unlock (XCP_RES_XXX).
<i>key</i>	pointer to the byte buffer holding the key.
<i>len</i>	length of the key in bytes.

Returns

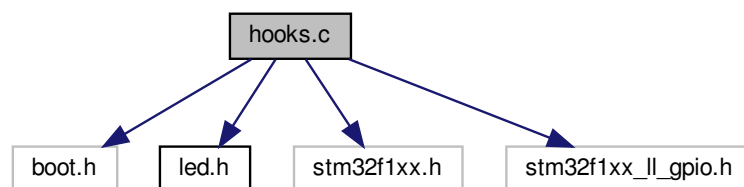
1 if the key was correct, 0 otherwise.

7.573 hooks.c File Reference

Bootloader callback source file.

```
#include "boot.h"
#include "led.h"
#include "stm32f1xx.h"
#include "stm32f1xx_ll_gpio.h"
```

Include dependency graph for ARMCM3_STM32F1_Nucleo_F103RB_CubeIDE/Boot/App/hooks.c:



Functions

- void [BackDoorInitHook](#) (void)
Initializes the backdoor entry option.
- [blt_bool](#) [BackDoorEntryHook](#) (void)
Checks if a backdoor entry is requested.
- [blt_bool](#) [CpuUserProgramStartHook](#) (void)
Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.
- void [CopInitHook](#) (void)
Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.
- void [CopServiceHook](#) (void)
Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.
- void [NvmInitHook](#) (void)
Callback that gets called at the start of the internal NVM driver initialization routine.
- void [NvmReinitHook](#) (void)
Callback that gets called at the start of a firmware update to reinitialize the NVM driver.
- [blt_int8u](#) [NvmWriteHook](#) ([blt_addr](#) addr, [blt_int32u](#) len, [blt_int8u](#) *data)
Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the data hasn't been written yet.
- [blt_int8u](#) [NvmEraseHook](#) ([blt_addr](#) addr, [blt_int32u](#) len)

Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then `BLT_NVM_NOT_IN_RANGE` must be returned to indicate that the memory hasn't been erased yet.

- `blt_bool NvmDoneHook` (void)

Callback that gets called at the end of the NVM programming session.

- `blt_int8u XcpGetSeedHook` (`blt_int8u` resource, `blt_int8u *seed`)

Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the `GET_SEED` command.

- `blt_int8u XcpVerifyKeyHook` (`blt_int8u` resource, `blt_int8u *key`, `blt_int8u len`)

Called by the `UNLOCK` command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

7.573.1 Detailed Description

Bootloader callback source file.

7.573.2 Function Documentation

7.573.2.1 BackDoorEntryHook()

```
blt_bool BackDoorEntryHook (
    void )
```

Checks if a backdoor entry is requested.

Returns

`BLT_TRUE` if the backdoor entry is requested, `BLT_FALSE` otherwise.

7.573.2.2 BackDoorInitHook()

```
void BackDoorInitHook (
    void )
```

Initializes the backdoor entry option.

Returns

none.

7.573.2.3 CopInitHook()

```
void CopInitHook (
    void )
```

Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.

Returns

none.

7.573.2.4 CopServiceHook()

```
void CopServiceHook (
    void )
```

Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.

Returns

none.

7.573.2.5 CpuUserProgramStartHook()

```
blt_bool CpuUserProgramStartHook (
    void )
```

Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.

Returns

BLT_TRUE if it is okay to start the user program, BLT_FALSE to keep keep the bootloader active.

7.573.2.6 NvmDoneHook()

```
blt_bool NvmDoneHook (
    void )
```

Callback that gets called at the end of the NVM programming session.

Returns

BLT_TRUE is successful, BLT_FALSE otherwise.

7.573.2.7 NvmEraseHook()

```
blt_int8u NvmEraseHook (
    blt_addr addr,
    blt_int32u len )
```

Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the memory hasn't been erased yet.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.

Returns

BLT_NVM_OKAY if successful, BLT_NVM_NOT_IN_RANGE if the address is not within the supported memory range, or BLT_NVM_ERROR if the erase operation failed.

7.573.2.8 NvmInitHook()

```
void NvmInitHook (
    void )
```

Callback that gets called at the start of the internal NVM driver initialization routine.

Returns

none.

7.573.2.9 NvmReinitHook()

```
void NvmReinitHook (
    void )
```

Callback that gets called at the start of a firmware update to reinitialize the NVM driver.

Returns

none.

7.573.2.10 NvmWriteHook()

```
blt_int8u NvmWriteHook (
    blt_addr addr,
    blt_int32u len,
    blt_int8u * data )
```

Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the data hasn't been written yet.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.
<i>data</i>	Pointer to the data buffer.

Returns

BLT_NVM_OKAY if successful, BLT_NVM_NOT_IN_RANGE if the address is not within the supported memory range, or BLT_NVM_ERROR if the write operation failed.

7.573.2.11 XcpGetSeedHook()

```
blt_int8u XcpGetSeedHook (
    blt_int8u resource,
    blt_int8u * seed )
```

Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the GET_SEED command.

Parameters

<i>resource</i>	Resource that the seed is requested for (XCP_RES_XXX).
<i>seed</i>	Pointer to byte buffer where the seed will be stored.

Returns

Length of the seed in bytes.

7.573.2.12 XcpVerifyKeyHook()

```
blt_int8u XcpVerifyKeyHook (
    blt_int8u resource,
    blt_int8u * key,
    blt_int8u len )
```

Called by the UNLOCK command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

Parameters

<i>resource</i>	resource to unlock (XCP_RES_XXX).
<i>key</i>	pointer to the byte buffer holding the key.
<i>len</i>	length of the key in bytes.

Returns

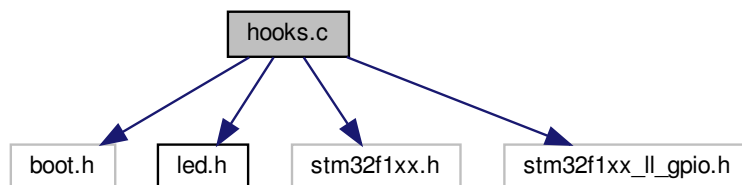
1 if the key was correct, 0 otherwise.

7.574 hooks.c File Reference

Bootloader callback source file.

```
#include "boot.h"
#include "led.h"
#include "stm32f1xx.h"
#include "stm32f1xx_ll_gpio.h"
```

Include dependency graph for ARMCM3_STM32F1_Nucleo_F103RB_GCC/Boot/hooks.c:



Functions

- void [BackDoorInitHook](#) (void)
Initializes the backdoor entry option.
- [blt_bool](#) [BackDoorEntryHook](#) (void)
Checks if a backdoor entry is requested.
- [blt_bool](#) [CpuUserProgramStartHook](#) (void)
Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.
- void [CopInitHook](#) (void)
Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.
- void [CopServiceHook](#) (void)
Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.
- void [NvmInitHook](#) (void)
Callback that gets called at the start of the internal NVM driver initialization routine.
- void [NvmReinitHook](#) (void)
Callback that gets called at the start of a firmware update to reinitialize the NVM driver.
- [blt_int8u](#) [NvmWriteHook](#) ([blt_addr](#) addr, [blt_int32u](#) len, [blt_int8u](#) *data)
Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the data hasn't been written yet.
- [blt_int8u](#) [NvmEraseHook](#) ([blt_addr](#) addr, [blt_int32u](#) len)

Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then `BLT_NVM_NOT_IN_RANGE` must be returned to indicate that the memory hasn't been erased yet.

- `blt_bool NvmDoneHook` (void)

Callback that gets called at the end of the NVM programming session.

- `blt_int8u XcpGetSeedHook` (`blt_int8u` resource, `blt_int8u *seed`)

Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the `GET_SEED` command.

- `blt_int8u XcpVerifyKeyHook` (`blt_int8u` resource, `blt_int8u *key`, `blt_int8u len`)

Called by the `UNLOCK` command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

7.574.1 Detailed Description

Bootloader callback source file.

7.574.2 Function Documentation

7.574.2.1 BackDoorEntryHook()

```
blt_bool BackDoorEntryHook (
    void )
```

Checks if a backdoor entry is requested.

Returns

`BLT_TRUE` if the backdoor entry is requested, `BLT_FALSE` otherwise.

7.574.2.2 BackDoorInitHook()

```
void BackDoorInitHook (
    void )
```

Initializes the backdoor entry option.

Returns

none.

7.574.2.3 CopInitHook()

```
void CopInitHook (
    void )
```

Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.

Returns

none.

7.574.2.4 CopServiceHook()

```
void CopServiceHook (
    void )
```

Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.

Returns

none.

7.574.2.5 CpuUserProgramStartHook()

```
blt_bool CpuUserProgramStartHook (
    void )
```

Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.

Returns

BLT_TRUE if it is okay to start the user program, BLT_FALSE to keep keep the bootloader active.

7.574.2.6 NvmDoneHook()

```
blt_bool NvmDoneHook (
    void )
```

Callback that gets called at the end of the NVM programming session.

Returns

BLT_TRUE is successful, BLT_FALSE otherwise.

7.574.2.7 NvmEraseHook()

```
blt_int8u NvmEraseHook (
    blt_addr addr,
    blt_int32u len )
```

Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the memory hasn't been erased yet.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.

Returns

BLT_NVM_OKAY if successful, BLT_NVM_NOT_IN_RANGE if the address is not within the supported memory range, or BLT_NVM_ERROR if the erase operation failed.

7.574.2.8 NvmInitHook()

```
void NvmInitHook (
    void )
```

Callback that gets called at the start of the internal NVM driver initialization routine.

Returns

none.

7.574.2.9 NvmReinitHook()

```
void NvmReinitHook (
    void )
```

Callback that gets called at the start of a firmware update to reinitialize the NVM driver.

Returns

none.

7.574.2.10 NvmWriteHook()

```
blt_int8u NvmWriteHook (
    blt_addr addr,
    blt_int32u len,
    blt_int8u * data )
```

Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the data hasn't been written yet.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.
<i>data</i>	Pointer to the data buffer.

Returns

BLT_NVM_OKAY if successful, BLT_NVM_NOT_IN_RANGE if the address is not within the supported memory range, or BLT_NVM_ERROR if the write operation failed.

7.574.2.11 XcpGetSeedHook()

```
blt_int8u XcpGetSeedHook (
    blt_int8u resource,
    blt_int8u * seed )
```

Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the GET_SEED command.

Parameters

<i>resource</i>	Resource that the seed is requested for (XCP_RES_XXX).
<i>seed</i>	Pointer to byte buffer where the seed will be stored.

Returns

Length of the seed in bytes.

7.574.2.12 XcpVerifyKeyHook()

```
blt_int8u XcpVerifyKeyHook (
    blt_int8u resource,
    blt_int8u * key,
    blt_int8u len )
```

Called by the UNLOCK command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

Parameters

<i>resource</i>	resource to unlock (XCP_RES_XXX).
<i>key</i>	pointer to the byte buffer holding the key.
<i>len</i>	length of the key in bytes.

Returns

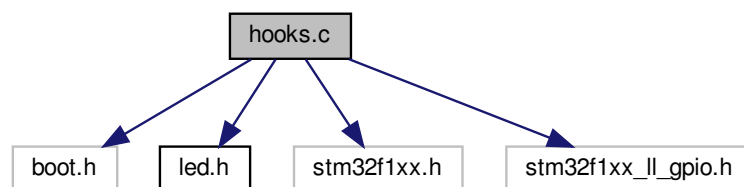
1 if the key was correct, 0 otherwise.

7.575 hooks.c File Reference

Bootloader callback source file.

```
#include "boot.h"
#include "led.h"
#include "stm32f1xx.h"
#include "stm32f1xx_ll_gpio.h"
```

Include dependency graph for ARMCM3_STM32F1_Nucleo_F103RB_IAR/Boot/hooks.c:



Functions

- void [BackDoorInitHook](#) (void)
Initializes the backdoor entry option.
- [blt_bool](#) [BackDoorEntryHook](#) (void)
Checks if a backdoor entry is requested.
- [blt_bool](#) [CpuUserProgramStartHook](#) (void)
Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.
- void [CopInitHook](#) (void)
Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.
- void [CopServiceHook](#) (void)
Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.
- void [NvmInitHook](#) (void)
Callback that gets called at the start of the internal NVM driver initialization routine.
- void [NvmReinitHook](#) (void)
Callback that gets called at the start of a firmware update to reinitialize the NVM driver.
- [blt_int8u](#) [NvmWriteHook](#) ([blt_addr](#) addr, [blt_int32u](#) len, [blt_int8u](#) *data)
Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the data hasn't been written yet.
- [blt_int8u](#) [NvmEraseHook](#) ([blt_addr](#) addr, [blt_int32u](#) len)

Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then `BLT_NVM_NOT_IN_RANGE` must be returned to indicate that the memory hasn't been erased yet.

- `blt_bool NvmDoneHook` (void)

Callback that gets called at the end of the NVM programming session.

- `blt_int8u XcpGetSeedHook` (`blt_int8u` resource, `blt_int8u *seed`)

Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the `GET_SEED` command.

- `blt_int8u XcpVerifyKeyHook` (`blt_int8u` resource, `blt_int8u *key`, `blt_int8u len`)

Called by the `UNLOCK` command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

7.575.1 Detailed Description

Bootloader callback source file.

7.575.2 Function Documentation

7.575.2.1 BackDoorEntryHook()

```
blt_bool BackDoorEntryHook (
    void )
```

Checks if a backdoor entry is requested.

Returns

`BLT_TRUE` if the backdoor entry is requested, `BLT_FALSE` otherwise.

7.575.2.2 BackDoorInitHook()

```
void BackDoorInitHook (
    void )
```

Initializes the backdoor entry option.

Returns

none.

7.575.2.3 CopInitHook()

```
void CopInitHook (
    void )
```

Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.

Returns

none.

7.575.2.4 CopServiceHook()

```
void CopServiceHook (
    void )
```

Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.

Returns

none.

7.575.2.5 CpuUserProgramStartHook()

```
blt_bool CpuUserProgramStartHook (
    void )
```

Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.

Returns

BLT_TRUE if it is okay to start the user program, BLT_FALSE to keep keep the bootloader active.

7.575.2.6 NvmDoneHook()

```
blt_bool NvmDoneHook (
    void )
```

Callback that gets called at the end of the NVM programming session.

Returns

BLT_TRUE is successful, BLT_FALSE otherwise.

7.575.2.7 NvmEraseHook()

```
blt_int8u NvmEraseHook (
    blt_addr addr,
    blt_int32u len )
```

Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the memory hasn't been erased yet.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.

Returns

BLT_NVM_OKAY if successful, BLT_NVM_NOT_IN_RANGE if the address is not within the supported memory range, or BLT_NVM_ERROR if the erase operation failed.

7.575.2.8 NvmInitHook()

```
void NvmInitHook (
    void )
```

Callback that gets called at the start of the internal NVM driver initialization routine.

Returns

none.

7.575.2.9 NvmReinitHook()

```
void NvmReinitHook (
    void )
```

Callback that gets called at the start of a firmware update to reinitialize the NVM driver.

Returns

none.

7.575.2.10 NvmWriteHook()

```
blt_int8u NvmWriteHook (
    blt_addr addr,
    blt_int32u len,
    blt_int8u * data )
```

Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the data hasn't been written yet.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.
<i>data</i>	Pointer to the data buffer.

Returns

BLT_NVM_OKAY if successful, BLT_NVM_NOT_IN_RANGE if the address is not within the supported memory range, or BLT_NVM_ERROR if the write operation failed.

7.575.2.11 XcpGetSeedHook()

```
blt_int8u XcpGetSeedHook (
    blt_int8u resource,
    blt_int8u * seed )
```

Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the GET_SEED command.

Parameters

<i>resource</i>	Resource that the seed is requested for (XCP_RES_XXX).
<i>seed</i>	Pointer to byte buffer where the seed will be stored.

Returns

Length of the seed in bytes.

7.575.2.12 XcpVerifyKeyHook()

```
blt_int8u XcpVerifyKeyHook (
    blt_int8u resource,
    blt_int8u * key,
    blt_int8u len )
```

Called by the UNLOCK command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

Parameters

<i>resource</i>	resource to unlock (XCP_RES_XXX).
<i>key</i>	pointer to the byte buffer holding the key.
<i>len</i>	length of the key in bytes.

Returns

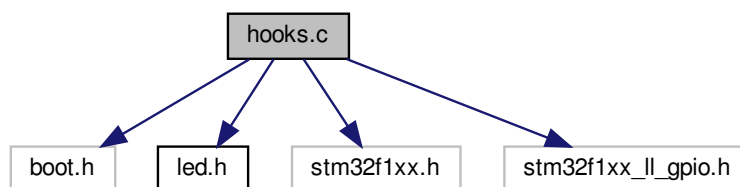
1 if the key was correct, 0 otherwise.

7.576 hooks.c File Reference

Bootloader callback source file.

```
#include "boot.h"
#include "led.h"
#include "stm32f1xx.h"
#include "stm32f1xx_ll_gpio.h"
```

Include dependency graph for ARMCM3_STM32F1_Nucleo_F103RB_Keil/Boot/hooks.c:



Functions

- void [BackDoorInitHook](#) (void)
Initializes the backdoor entry option.
- [blt_bool](#) [BackDoorEntryHook](#) (void)
Checks if a backdoor entry is requested.
- [blt_bool](#) [CpuUserProgramStartHook](#) (void)
Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.
- void [CopInitHook](#) (void)
Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.
- void [CopServiceHook](#) (void)
Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.
- void [NvmInitHook](#) (void)
Callback that gets called at the start of the internal NVM driver initialization routine.
- void [NvmReinitHook](#) (void)
Callback that gets called at the start of a firmware update to reinitialize the NVM driver.
- [blt_int8u](#) [NvmWriteHook](#) ([blt_addr](#) addr, [blt_int32u](#) len, [blt_int8u](#) *data)
Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the data hasn't been written yet.
- [blt_int8u](#) [NvmEraseHook](#) ([blt_addr](#) addr, [blt_int32u](#) len)

Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then `BLT_NVM_NOT_IN_RANGE` must be returned to indicate that the memory hasn't been erased yet.

- `blt_bool NvmDoneHook` (void)

Callback that gets called at the end of the NVM programming session.

- `blt_int8u XcpGetSeedHook` (`blt_int8u` resource, `blt_int8u *seed`)

Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the `GET_SEED` command.

- `blt_int8u XcpVerifyKeyHook` (`blt_int8u` resource, `blt_int8u *key`, `blt_int8u len`)

Called by the `UNLOCK` command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

7.576.1 Detailed Description

Bootloader callback source file.

7.576.2 Function Documentation

7.576.2.1 BackDoorEntryHook()

```
blt_bool BackDoorEntryHook (  
    void )
```

Checks if a backdoor entry is requested.

Returns

`BLT_TRUE` if the backdoor entry is requested, `BLT_FALSE` otherwise.

7.576.2.2 BackDoorInitHook()

```
void BackDoorInitHook (  
    void )
```

Initializes the backdoor entry option.

Returns

none.

7.576.2.3 CopInitHook()

```
void CopInitHook (
    void )
```

Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.

Returns

none.

7.576.2.4 CopServiceHook()

```
void CopServiceHook (
    void )
```

Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.

Returns

none.

7.576.2.5 CpuUserProgramStartHook()

```
blt_bool CpuUserProgramStartHook (
    void )
```

Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.

Returns

BLT_TRUE if it is okay to start the user program, BLT_FALSE to keep keep the bootloader active.

7.576.2.6 NvmDoneHook()

```
blt_bool NvmDoneHook (
    void )
```

Callback that gets called at the end of the NVM programming session.

Returns

BLT_TRUE is successful, BLT_FALSE otherwise.

7.576.2.7 NvmEraseHook()

```
blt_int8u NvmEraseHook (
    blt_addr addr,
    blt_int32u len )
```

Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the memory hasn't been erased yet.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.

Returns

BLT_NVM_OKAY if successful, BLT_NVM_NOT_IN_RANGE if the address is not within the supported memory range, or BLT_NVM_ERROR if the erase operation failed.

7.576.2.8 NvmInitHook()

```
void NvmInitHook (
    void )
```

Callback that gets called at the start of the internal NVM driver initialization routine.

Returns

none.

7.576.2.9 NvmReinitHook()

```
void NvmReinitHook (
    void )
```

Callback that gets called at the start of a firmware update to reinitialize the NVM driver.

Returns

none.

7.576.2.10 NvmWriteHook()

```
blt_int8u NvmWriteHook (
    blt_addr addr,
    blt_int32u len,
    blt_int8u * data )
```

Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the data hasn't been written yet.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.
<i>data</i>	Pointer to the data buffer.

Returns

BLT_NVM_OKAY if successful, BLT_NVM_NOT_IN_RANGE if the address is not within the supported memory range, or BLT_NVM_ERROR if the write operation failed.

7.576.2.11 XcpGetSeedHook()

```
blt_int8u XcpGetSeedHook (
    blt_int8u resource,
    blt_int8u * seed )
```

Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the GET_SEED command.

Parameters

<i>resource</i>	Resource that the seed is requested for (XCP_RES_XXX).
<i>seed</i>	Pointer to byte buffer where the seed will be stored.

Returns

Length of the seed in bytes.

7.576.2.12 XcpVerifyKeyHook()

```
blt_int8u XcpVerifyKeyHook (
    blt_int8u resource,
    blt_int8u * key,
    blt_int8u len )
```

Called by the UNLOCK command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

Parameters

<i>resource</i>	resource to unlock (XCP_RES_XXX).
<i>key</i>	pointer to the byte buffer holding the key.
<i>len</i>	length of the key in bytes.

Returns

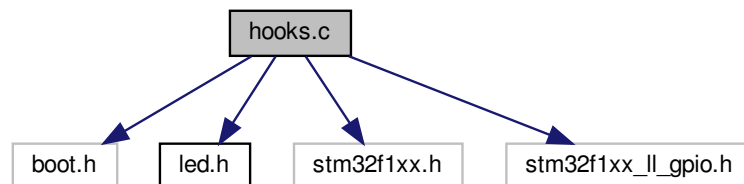
1 if the key was correct, 0 otherwise.

7.577 hooks.c File Reference

Bootloader callback source file.

```
#include "boot.h"
#include "led.h"
#include "stm32f1xx.h"
#include "stm32f1xx_ll_gpio.h"
```

Include dependency graph for ARMC3M3_STM32F1_Olimex_STM32H103_CubeIDE/Boot/App/hooks.c:



Functions

- void [UsbConnectHook](#) ([blt_bool](#) connect)
Callback that gets called whenever the USB device should be connected to the USB bus.
- void [UsbEnterLowPowerModeHook](#) (void)
Callback that gets called whenever the USB host requests the device to enter a low power mode.
- void [UsbLeaveLowPowerModeHook](#) (void)
Callback that gets called whenever the USB host requests the device to exit low power mode.
- void [BackDoorInitHook](#) (void)
Initializes the backdoor entry option.
- [blt_bool](#) [BackDoorEntryHook](#) (void)
Checks if a backdoor entry is requested.
- [blt_bool](#) [CpuUserProgramStartHook](#) (void)
Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.
- void [CopInitHook](#) (void)
Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.
- void [CopServiceHook](#) (void)
Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.
- void [NvmInitHook](#) (void)
Callback that gets called at the start of the internal NVM driver initialization routine.
- void [NvmReinitHook](#) (void)

Callback that gets called at the start of a firmware update to reinitialize the NVM driver.

- `blt_int8u NvmWriteHook (blt_addr addr, blt_int32u len, blt_int8u *data)`

Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then `BLT_NVM_NOT_IN_RANGE` must be returned to indicate that the data hasn't been written yet.

- `blt_int8u NvmEraseHook (blt_addr addr, blt_int32u len)`

Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then `BLT_NVM_NOT_IN_RANGE` must be returned to indicate that the memory hasn't been erased yet.

- `blt_bool NvmDoneHook (void)`

Callback that gets called at the end of the NVM programming session.

- `blt_int8u XcpGetSeedHook (blt_int8u resource, blt_int8u *seed)`

Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the `GET_SEED` command.

- `blt_int8u XcpVerifyKeyHook (blt_int8u resource, blt_int8u *key, blt_int8u len)`

Called by the `UNLOCK` command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

7.577.1 Detailed Description

Bootloader callback source file.

7.577.2 Function Documentation

7.577.2.1 BackDoorEntryHook()

```
blt_bool BackDoorEntryHook (
    void )
```

Checks if a backdoor entry is requested.

Returns

`BLT_TRUE` if the backdoor entry is requested, `BLT_FALSE` otherwise.

7.577.2.2 BackDoorInitHook()

```
void BackDoorInitHook (
    void )
```

Initializes the backdoor entry option.

Returns

none.

7.577.2.3 CopInitHook()

```
void CopInitHook (
    void )
```

Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.

Returns

none.

7.577.2.4 CopServiceHook()

```
void CopServiceHook (
    void )
```

Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.

Returns

none.

7.577.2.5 CpuUserProgramStartHook()

```
blt_bool CpuUserProgramStartHook (
    void )
```

Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.

Returns

BLT_TRUE if it is okay to start the user program, BLT_FALSE to keep keep the bootloader active.

7.577.2.6 NvmDoneHook()

```
blt_bool NvmDoneHook (
    void )
```

Callback that gets called at the end of the NVM programming session.

Returns

BLT_TRUE is successful, BLT_FALSE otherwise.

7.577.2.7 NvmEraseHook()

```
blt_int8u NvmEraseHook (
    blt_addr addr,
    blt_int32u len )
```

Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the memory hasn't been erased yet.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.

Returns

BLT_NVM_OKAY if successful, BLT_NVM_NOT_IN_RANGE if the address is not within the supported memory range, or BLT_NVM_ERROR if the erase operation failed.

7.577.2.8 NvmInitHook()

```
void NvmInitHook (
    void )
```

Callback that gets called at the start of the internal NVM driver initialization routine.

Returns

none.

7.577.2.9 NvmReinitHook()

```
void NvmReinitHook (
    void )
```

Callback that gets called at the start of a firmware update to reinitialize the NVM driver.

Returns

none.

7.577.2.10 NvmWriteHook()

```
blt_int8u NvmWriteHook (
    blt_addr addr,
    blt_int32u len,
    blt_int8u * data )
```

Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the data hasn't been written yet.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.
<i>data</i>	Pointer to the data buffer.

Returns

BLT_NVM_OKAY if successful, BLT_NVM_NOT_IN_RANGE if the address is not within the supported memory range, or BLT_NVM_ERROR if the write operation failed.

7.577.2.11 UsbConnectHook()

```
void UsbConnectHook (
    blt_bool connect )
```

Callback that gets called whenever the USB device should be connected to the USB bus.

Parameters

<i>connect</i>	BLT_TRUE to connect and BLT_FALSE to disconnect.
----------------	--

Returns

none.

7.577.2.12 UsbEnterLowPowerModeHook()

```
void UsbEnterLowPowerModeHook (
    void )
```

Callback that gets called whenever the USB host requests the device to enter a low power mode.

Returns

none.

7.577.2.13 UsbLeaveLowPowerModeHook()

```
void UsbLeaveLowPowerModeHook (
    void )
```

Callback that gets called whenever the USB host requests the device to exit low power mode.

Returns

none.

7.577.2.14 XcpGetSeedHook()

```
blt_int8u XcpGetSeedHook (
    blt_int8u resource,
    blt_int8u * seed )
```

Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the GET_SEED command.

Parameters

<i>resource</i>	Resource that the seed is requested for (XCP_RES_XXX).
<i>seed</i>	Pointer to byte buffer where the seed will be stored.

Returns

Length of the seed in bytes.

7.577.2.15 XcpVerifyKeyHook()

```
blt_int8u XcpVerifyKeyHook (
    blt_int8u resource,
    blt_int8u * key,
    blt_int8u len )
```

Called by the UNLOCK command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

Parameters

<i>resource</i>	resource to unlock (XCP_RES_XXX).
<i>key</i>	pointer to the byte buffer holding the key.
<i>len</i>	length of the key in bytes.

Returns

1 if the key was correct, 0 otherwise.

7.578 hooks.c File Reference

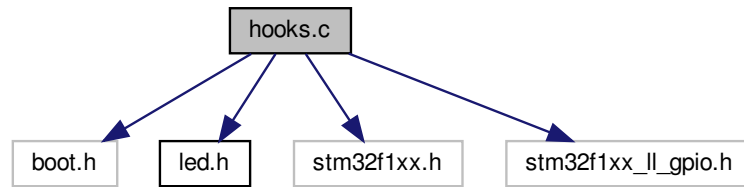
Bootloader callback source file.

```
#include "boot.h"
#include "led.h"
#include "stm32f1xx.h"
```



```
#include "stm32f1xx_ll_gpio.h"
```

Include dependency graph for ARMC3M3_STM32F1_Olimex_STM32H103_GCC/Boot/hooks.c:



Functions

- void [UsbConnectHook](#) ([blt_bool](#) connect)
Callback that gets called whenever the USB device should be connected to the USB bus.
- void [UsbEnterLowPowerModeHook](#) (void)
Callback that gets called whenever the USB host requests the device to enter a low power mode.
- void [UsbLeaveLowPowerModeHook](#) (void)
Callback that gets called whenever the USB host requests the device to exit low power mode.
- void [BackDoorInitHook](#) (void)
Initializes the backdoor entry option.
- [blt_bool](#) [BackDoorEntryHook](#) (void)
Checks if a backdoor entry is requested.
- [blt_bool](#) [CpuUserProgramStartHook](#) (void)
Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.
- void [CopInitHook](#) (void)
Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.
- void [CopServiceHook](#) (void)
Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.
- void [NvmInitHook](#) (void)
Callback that gets called at the start of the internal NVM driver initialization routine.
- void [NvmReinitHook](#) (void)
Callback that gets called at the start of a firmware update to reinitialize the NVM driver.
- [blt_int8u](#) [NvmWriteHook](#) ([blt_addr](#) addr, [blt_int32u](#) len, [blt_int8u](#) *data)
Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the data hasn't been written yet.
- [blt_int8u](#) [NvmEraseHook](#) ([blt_addr](#) addr, [blt_int32u](#) len)
Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the memory hasn't been erased yet.
- [blt_bool](#) [NvmDoneHook](#) (void)
Callback that gets called at the end of the NVM programming session.
- [blt_int8u](#) [XcpGetSeedHook](#) ([blt_int8u](#) resource, [blt_int8u](#) *seed)

Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the GET_SEED command.

- `blt_int8u XcpVerifyKeyHook (blt_int8u resource, blt_int8u *key, blt_int8u len)`

Called by the UNLOCK command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

7.578.1 Detailed Description

Bootloader callback source file.

7.578.2 Function Documentation

7.578.2.1 BackDoorEntryHook()

```
blt_bool BackDoorEntryHook (  
    void )
```

Checks if a backdoor entry is requested.

Returns

BLT_TRUE if the backdoor entry is requested, BLT_FALSE otherwise.

7.578.2.2 BackDoorInitHook()

```
void BackDoorInitHook (  
    void )
```

Initializes the backdoor entry option.

Returns

none.

7.578.2.3 CopInitHook()

```
void CopInitHook (  
    void )
```

Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.

Returns

none.

7.578.2.4 CopServiceHook()

```
void CopServiceHook (
    void )
```

Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.

Returns

none.

7.578.2.5 CpuUserProgramStartHook()

```
blt_bool CpuUserProgramStartHook (
    void )
```

Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.

Returns

BLT_TRUE if it is okay to start the user program, BLT_FALSE to keep keep the bootloader active.

7.578.2.6 NvmDoneHook()

```
blt_bool NvmDoneHook (
    void )
```

Callback that gets called at the end of the NVM programming session.

Returns

BLT_TRUE is successful, BLT_FALSE otherwise.

7.578.2.7 NvmEraseHook()

```
blt_int8u NvmEraseHook (
    blt_addr addr,
    blt_int32u len )
```

Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the memory hasn't been erased yet.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.

Returns

BLT_NVM_OKAY if successful, BLT_NVM_NOT_IN_RANGE if the address is not within the supported memory range, or BLT_NVM_ERROR if the erase operation failed.

7.578.2.8 NvmInitHook()

```
void NvmInitHook (  
    void )
```

Callback that gets called at the start of the internal NVM driver initialization routine.

Returns

none.

7.578.2.9 NvmReinitHook()

```
void NvmReinitHook (  
    void )
```

Callback that gets called at the start of a firmware update to reinitialize the NVM driver.

Returns

none.

7.578.2.10 NvmWriteHook()

```
blt_int8u NvmWriteHook (  
    blt_addr addr,  
    blt_int32u len,  
    blt_int8u * data )
```

Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the data hasn't been written yet.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.
<i>data</i>	Pointer to the data buffer.

Returns

BLT_NVM_OKAY if successful, BLT_NVM_NOT_IN_RANGE if the address is not within the supported memory range, or BLT_NVM_ERROR if the write operation failed.

7.578.2.11 UsbConnectHook()

```
void UsbConnectHook (
    blt_bool connect )
```

Callback that gets called whenever the USB device should be connected to the USB bus.

Parameters

<i>connect</i>	BLT_TRUE to connect and BLT_FALSE to disconnect.
----------------	--

Returns

none.

7.578.2.12 UsbEnterLowPowerModeHook()

```
void UsbEnterLowPowerModeHook (
    void )
```

Callback that gets called whenever the USB host requests the device to enter a low power mode.

Returns

none.

7.578.2.13 UsbLeaveLowPowerModeHook()

```
void UsbLeaveLowPowerModeHook (
    void )
```

Callback that gets called whenever the USB host requests the device to exit low power mode.

Returns

none.

7.578.2.14 XcpGetSeedHook()

```
blt_int8u XcpGetSeedHook (
    blt_int8u resource,
    blt_int8u * seed )
```

Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the GET_SEED command.

Parameters

<i>resource</i>	Resource that the seed is requested for (XCP_RES_XXX).
<i>seed</i>	Pointer to byte buffer where the seed will be stored.

Returns

Length of the seed in bytes.

7.578.2.15 XcpVerifyKeyHook()

```
blt_int8u XcpVerifyKeyHook (
    blt_int8u resource,
    blt_int8u * key,
    blt_int8u len )
```

Called by the UNLOCK command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

Parameters

<i>resource</i>	resource to unlock (XCP_RES_XXX).
<i>key</i>	pointer to the byte buffer holding the key.
<i>len</i>	length of the key in bytes.

Returns

1 if the key was correct, 0 otherwise.

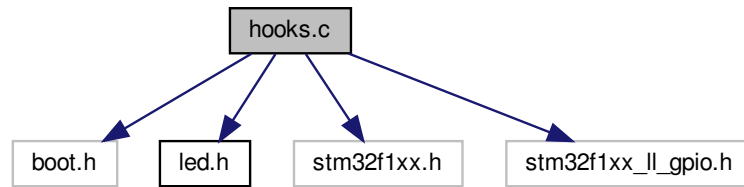
7.579 hooks.c File Reference

Bootloader callback source file.

```
#include "boot.h"
#include "led.h"
#include "stm32f1xx.h"
```

```
#include "stm32f1xx_ll_gpio.h"
```

Include dependency graph for ARMC3M3_STM32F1_Olimex_STM32H103_IAR/Boot/hooks.c:



Functions

- void [UsbConnectHook](#) ([blt_bool](#) connect)
Callback that gets called whenever the USB device should be connected to the USB bus.
- void [UsbEnterLowPowerModeHook](#) (void)
Callback that gets called whenever the USB host requests the device to enter a low power mode.
- void [UsbLeaveLowPowerModeHook](#) (void)
Callback that gets called whenever the USB host requests the device to exit low power mode.
- void [BackDoorInitHook](#) (void)
Initializes the backdoor entry option.
- [blt_bool](#) [BackDoorEntryHook](#) (void)
Checks if a backdoor entry is requested.
- [blt_bool](#) [CpuUserProgramStartHook](#) (void)
Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.
- void [CopInitHook](#) (void)
Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.
- void [CopServiceHook](#) (void)
Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.
- void [NvmInitHook](#) (void)
Callback that gets called at the start of the internal NVM driver initialization routine.
- void [NvmReinitHook](#) (void)
Callback that gets called at the start of a firmware update to reinitialize the NVM driver.
- [blt_int8u](#) [NvmWriteHook](#) ([blt_addr](#) addr, [blt_int32u](#) len, [blt_int8u](#) *data)
Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the data hasn't been written yet.
- [blt_int8u](#) [NvmEraseHook](#) ([blt_addr](#) addr, [blt_int32u](#) len)
Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the memory hasn't been erased yet.
- [blt_bool](#) [NvmDoneHook](#) (void)
Callback that gets called at the end of the NVM programming session.
- [blt_int8u](#) [XcpGetSeedHook](#) ([blt_int8u](#) resource, [blt_int8u](#) *seed)

Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the GET_SEED command.

- `blt_int8u XcpVerifyKeyHook (blt_int8u resource, blt_int8u *key, blt_int8u len)`

Called by the UNLOCK command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

7.579.1 Detailed Description

Bootloader callback source file.

7.579.2 Function Documentation

7.579.2.1 BackDoorEntryHook()

```
blt_bool BackDoorEntryHook (  
    void )
```

Checks if a backdoor entry is requested.

Returns

BLT_TRUE if the backdoor entry is requested, BLT_FALSE otherwise.

7.579.2.2 BackDoorInitHook()

```
void BackDoorInitHook (  
    void )
```

Initializes the backdoor entry option.

Returns

none.

7.579.2.3 CopInitHook()

```
void CopInitHook (  
    void )
```

Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.

Returns

none.

7.579.2.4 CopServiceHook()

```
void CopServiceHook (
    void )
```

Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.

Returns

none.

7.579.2.5 CpuUserProgramStartHook()

```
blt_bool CpuUserProgramStartHook (
    void )
```

Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.

Returns

BLT_TRUE if it is okay to start the user program, BLT_FALSE to keep keep the bootloader active.

7.579.2.6 NvmDoneHook()

```
blt_bool NvmDoneHook (
    void )
```

Callback that gets called at the end of the NVM programming session.

Returns

BLT_TRUE is successful, BLT_FALSE otherwise.

7.579.2.7 NvmEraseHook()

```
blt_int8u NvmEraseHook (
    blt_addr addr,
    blt_int32u len )
```

Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the memory hasn't been erased yet.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.

Returns

BLT_NVM_OKAY if successful, BLT_NVM_NOT_IN_RANGE if the address is not within the supported memory range, or BLT_NVM_ERROR if the erase operation failed.

7.579.2.8 NvmInitHook()

```
void NvmInitHook (  
    void )
```

Callback that gets called at the start of the internal NVM driver initialization routine.

Returns

none.

7.579.2.9 NvmReinitHook()

```
void NvmReinitHook (  
    void )
```

Callback that gets called at the start of a firmware update to reinitialize the NVM driver.

Returns

none.

7.579.2.10 NvmWriteHook()

```
blt_int8u NvmWriteHook (  
    blt_addr addr,  
    blt_int32u len,  
    blt_int8u * data )
```

Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the data hasn't been written yet.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.
<i>data</i>	Pointer to the data buffer.

Returns

BLT_NVM_OKAY if successful, BLT_NVM_NOT_IN_RANGE if the address is not within the supported memory range, or BLT_NVM_ERROR if the write operation failed.

7.579.2.11 UsbConnectHook()

```
void UsbConnectHook (
    blt_bool connect )
```

Callback that gets called whenever the USB device should be connected to the USB bus.

Parameters

<i>connect</i>	BLT_TRUE to connect and BLT_FALSE to disconnect.
----------------	--

Returns

none.

7.579.2.12 UsbEnterLowPowerModeHook()

```
void UsbEnterLowPowerModeHook (
    void )
```

Callback that gets called whenever the USB host requests the device to enter a low power mode.

Returns

none.

7.579.2.13 UsbLeaveLowPowerModeHook()

```
void UsbLeaveLowPowerModeHook (
    void )
```

Callback that gets called whenever the USB host requests the device to exit low power mode.

Returns

none.

7.579.2.14 XcpGetSeedHook()

```
blt_int8u XcpGetSeedHook (
    blt_int8u resource,
    blt_int8u * seed )
```

Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the GET_SEED command.

Parameters

<i>resource</i>	Resource that the seed is requested for (XCP_RES_XXX).
<i>seed</i>	Pointer to byte buffer where the seed will be stored.

Returns

Length of the seed in bytes.

7.579.2.15 XcpVerifyKeyHook()

```
blt_int8u XcpVerifyKeyHook (
    blt_int8u resource,
    blt_int8u * key,
    blt_int8u len )
```

Called by the UNLOCK command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

Parameters

<i>resource</i>	resource to unlock (XCP_RES_XXX).
<i>key</i>	pointer to the byte buffer holding the key.
<i>len</i>	length of the key in bytes.

Returns

1 if the key was correct, 0 otherwise.

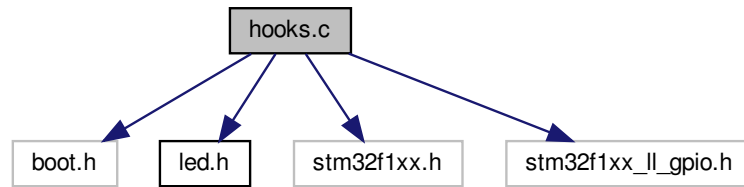
7.580 hooks.c File Reference

Bootloader callback source file.

```
#include "boot.h"
#include "led.h"
#include "stm32f1xx.h"
```

```
#include "stm32f1xx_ll_gpio.h"
```

Include dependency graph for ARMC32F1_Olimex_STM32H103_Keil/Boot/hooks.c:



Functions

- void [UsbConnectHook](#) ([blt_bool](#) connect)
Callback that gets called whenever the USB device should be connected to the USB bus.
- void [UsbEnterLowPowerModeHook](#) (void)
Callback that gets called whenever the USB host requests the device to enter a low power mode.
- void [UsbLeaveLowPowerModeHook](#) (void)
Callback that gets called whenever the USB host requests the device to exit low power mode.
- void [BackDoorInitHook](#) (void)
Initializes the backdoor entry option.
- [blt_bool](#) [BackDoorEntryHook](#) (void)
Checks if a backdoor entry is requested.
- [blt_bool](#) [CpuUserProgramStartHook](#) (void)
Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.
- void [CopInitHook](#) (void)
Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.
- void [CopServiceHook](#) (void)
Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.
- void [NvmInitHook](#) (void)
Callback that gets called at the start of the internal NVM driver initialization routine.
- void [NvmReinitHook](#) (void)
Callback that gets called at the start of a firmware update to reinitialize the NVM driver.
- [blt_int8u](#) [NvmWriteHook](#) ([blt_addr](#) addr, [blt_int32u](#) len, [blt_int8u](#) *data)
Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the data hasn't been written yet.
- [blt_int8u](#) [NvmEraseHook](#) ([blt_addr](#) addr, [blt_int32u](#) len)
Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the memory hasn't been erased yet.
- [blt_bool](#) [NvmDoneHook](#) (void)
Callback that gets called at the end of the NVM programming session.
- [blt_int8u](#) [XcpGetSeedHook](#) ([blt_int8u](#) resource, [blt_int8u](#) *seed)

Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the GET_SEED command.

- `blt_int8u XcpVerifyKeyHook (blt_int8u resource, blt_int8u *key, blt_int8u len)`

Called by the UNLOCK command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

7.580.1 Detailed Description

Bootloader callback source file.

7.580.2 Function Documentation

7.580.2.1 BackDoorEntryHook()

```
blt_bool BackDoorEntryHook (  
    void )
```

Checks if a backdoor entry is requested.

Returns

BLT_TRUE if the backdoor entry is requested, BLT_FALSE otherwise.

7.580.2.2 BackDoorInitHook()

```
void BackDoorInitHook (  
    void )
```

Initializes the backdoor entry option.

Returns

none.

7.580.2.3 CopInitHook()

```
void CopInitHook (  
    void )
```

Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.

Returns

none.

7.580.2.4 CopServiceHook()

```
void CopServiceHook (
    void )
```

Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.

Returns

none.

7.580.2.5 CpuUserProgramStartHook()

```
blt_bool CpuUserProgramStartHook (
    void )
```

Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.

Returns

BLT_TRUE if it is okay to start the user program, BLT_FALSE to keep keep the bootloader active.

7.580.2.6 NvmDoneHook()

```
blt_bool NvmDoneHook (
    void )
```

Callback that gets called at the end of the NVM programming session.

Returns

BLT_TRUE is successful, BLT_FALSE otherwise.

7.580.2.7 NvmEraseHook()

```
blt_int8u NvmEraseHook (
    blt_addr addr,
    blt_int32u len )
```

Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the memory hasn't been erased yet.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.

Returns

BLT_NVM_OKAY if successful, BLT_NVM_NOT_IN_RANGE if the address is not within the supported memory range, or BLT_NVM_ERROR if the erase operation failed.

7.580.2.8 NvmInitHook()

```
void NvmInitHook (  
    void )
```

Callback that gets called at the start of the internal NVM driver initialization routine.

Returns

none.

7.580.2.9 NvmReinitHook()

```
void NvmReinitHook (  
    void )
```

Callback that gets called at the start of a firmware update to reinitialize the NVM driver.

Returns

none.

7.580.2.10 NvmWriteHook()

```
blt_int8u NvmWriteHook (  
    blt_addr addr,  
    blt_int32u len,  
    blt_int8u * data )
```

Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the data hasn't been written yet.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.
<i>data</i>	Pointer to the data buffer.

Returns

BLT_NVM_OKAY if successful, BLT_NVM_NOT_IN_RANGE if the address is not within the supported memory range, or BLT_NVM_ERROR if the write operation failed.

7.580.2.11 UsbConnectHook()

```
void UsbConnectHook (
    blt_bool connect )
```

Callback that gets called whenever the USB device should be connected to the USB bus.

Parameters

<i>connect</i>	BLT_TRUE to connect and BLT_FALSE to disconnect.
----------------	--

Returns

none.

7.580.2.12 UsbEnterLowPowerModeHook()

```
void UsbEnterLowPowerModeHook (
    void )
```

Callback that gets called whenever the USB host requests the device to enter a low power mode.

Returns

none.

7.580.2.13 UsbLeaveLowPowerModeHook()

```
void UsbLeaveLowPowerModeHook (
    void )
```

Callback that gets called whenever the USB host requests the device to exit low power mode.

Returns

none.

7.580.2.14 XcpGetSeedHook()

```
blt_int8u XcpGetSeedHook (
    blt_int8u resource,
    blt_int8u * seed )
```

Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the GET_SEED command.

Parameters

<i>resource</i>	Resource that the seed is requested for (XCP_RES_XXX).
<i>seed</i>	Pointer to byte buffer where the seed will be stored.

Returns

Length of the seed in bytes.

7.580.2.15 XcpVerifyKeyHook()

```
blt_int8u XcpVerifyKeyHook (
    blt_int8u resource,
    blt_int8u * key,
    blt_int8u len )
```

Called by the UNLOCK command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

Parameters

<i>resource</i>	resource to unlock (XCP_RES_XXX).
<i>key</i>	pointer to the byte buffer holding the key.
<i>len</i>	length of the key in bytes.

Returns

1 if the key was correct, 0 otherwise.

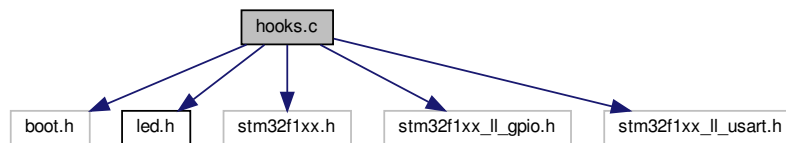
7.581 hooks.c File Reference

Bootloader callback source file.

```
#include "boot.h"
#include "led.h"
#include "stm32f1xx.h"
#include "stm32f1xx_ll_gpio.h"
```

```
#include "stm32f1xx_ll_usart.h"
```

Include dependency graph for ARMC32F1_Olimex_STM32P103_CubeIDE/Boot/App/hooks.c:



Functions

- void [BackDoorInitHook](#) (void)
Initializes the backdoor entry option.
- [blt_bool BackDoorEntryHook](#) (void)
Checks if a backdoor entry is requested.
- [blt_bool CpuUserProgramStartHook](#) (void)
Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.
- void [CopInitHook](#) (void)
Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.
- void [CopServiceHook](#) (void)
Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.
- void [NvmInitHook](#) (void)
Callback that gets called at the start of the internal NVM driver initialization routine.
- void [NvmReinitHook](#) (void)
Callback that gets called at the start of a firmware update to reinitialize the NVM driver.
- [blt_int8u NvmWriteHook](#) ([blt_addr](#) addr, [blt_int32u](#) len, [blt_int8u](#) *data)
Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the data hasn't been written yet.
- [blt_int8u NvmEraseHook](#) ([blt_addr](#) addr, [blt_int32u](#) len)
Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the memory hasn't been erased yet.
- [blt_bool NvmDoneHook](#) (void)
Callback that gets called at the end of the NVM programming session.
- [blt_bool FileIsFirmwareUpdateRequestedHook](#) (void)
Callback that gets called to check whether a firmware update from local file storage should be started. This could for example be when a switch is pressed, when a certain file is found on the local file storage, etc.
- const [blt_char](#) * [FileGetFirmwareFilenameHook](#) (void)
Callback to obtain the filename of the firmware file that should be used during the firmware update from the local file storage. This hook function is called at the beginning of the firmware update from local storage sequence.
- void [FileFirmwareUpdateStartedHook](#) (void)
Callback that gets called to inform the application that a firmware update from local storage just started.
- void [FileFirmwareUpdateCompletedHook](#) (void)
Callback that gets called to inform the application that a firmware update was successfully completed.

- void `FileFirmwareUpdateErrorHook` (`blt_int8u` error_code)
Callback that gets called in case an error occurred during a firmware update. Refer to <file.h> for a list of available error codes.
- void `FileFirmwareUpdateLogHook` (`blt_char` *info_string)
Callback that gets called each time new log information becomes available during a firmware update.
- `blt_int8u` `XcpGetSeedHook` (`blt_int8u` resource, `blt_int8u` *seed)
Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the GET_SEED command.
- `blt_int8u` `XcpVerifyKeyHook` (`blt_int8u` resource, `blt_int8u` *key, `blt_int8u` len)
Called by the UNLOCK command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

Variables

- static const `blt_char` `firmwareFilename` [] = "/demoprogram_olimex_stm32p103.srec"
Firmware filename.
- ```

struct {
 FIL handle
 blt_bool canUse
} logfile

```

*Data structure for grouping log-file related information.*

## 7.581.1 Detailed Description

Bootloader callback source file.

## 7.581.2 Function Documentation

### 7.581.2.1 BackDoorEntryHook()

```

blt_bool BackDoorEntryHook (
 void)

```

Checks if a backdoor entry is requested.

#### Returns

BLT\_TRUE if the backdoor entry is requested, BLT\_FALSE otherwise.

### 7.581.2.2 BackDoorInitHook()

```
void BackDoorInitHook (
 void)
```

Initializes the backdoor entry option.

#### Returns

none.

### 7.581.2.3 CopInitHook()

```
void CopInitHook (
 void)
```

Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.

#### Returns

none.

### 7.581.2.4 CopServiceHook()

```
void CopServiceHook (
 void)
```

Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.

#### Returns

none.

### 7.581.2.5 CpuUserProgramStartHook()

```
blt_bool CpuUserProgramStartHook (
 void)
```

Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.

#### Returns

BLT\_TRUE if it is okay to start the user program, BLT\_FALSE to keep keep the bootloader active.

#### 7.581.2.6 FileFirmwareUpdateCompletedHook()

```
void FileFirmwareUpdateCompletedHook (
 void)
```

Callback that gets called to inform the application that a firmware update was successfully completed.

##### Returns

none.

#### 7.581.2.7 FileFirmwareUpdateErrorHook()

```
void FileFirmwareUpdateErrorHook (
 blt_int8u error_code)
```

Callback that gets called in case an error occurred during a firmware update. Refer to <[file.h](#)> for a list of available error codes.

##### Returns

none.

#### 7.581.2.8 FileFirmwareUpdateLogHook()

```
void FileFirmwareUpdateLogHook (
 blt_char * info_string)
```

Callback that gets called each time new log information becomes available during a firmware update.

##### Parameters

|                    |                                                       |
|--------------------|-------------------------------------------------------|
| <i>info_string</i> | Pointer to a character array with the log entry info. |
|--------------------|-------------------------------------------------------|

##### Returns

none.

#### 7.581.2.9 FileFirmwareUpdateStartedHook()

```
void FileFirmwareUpdateStartedHook (
 void)
```

Callback that gets called to inform the application that a firmware update from local storage just started.

**Returns**

none.

**7.581.2.10 FileGetFirmwareFilenameHook()**

```
const blt_char* FileGetFirmwareFilenameHook (
 void)
```

Callback to obtain the filename of the firmware file that should be used during the firmware update from the local file storage. This hook function is called at the beginning of the firmware update from local storage sequence.

**Returns**

valid firmware filename with full path or BLT\_NULL.

**7.581.2.11 FileIsFirmwareUpdateRequestedHook()**

```
blt_bool FileIsFirmwareUpdateRequestedHook (
 void)
```

Callback that gets called to check whether a firmware update from local file storage should be started. This could for example be when a switch is pressed, when a certain file is found on the local file storage, etc.

**Returns**

BLT\_TRUE if a firmware update is requested, BLT\_FALSE otherwise.

**7.581.2.12 NvmDoneHook()**

```
blt_bool NvmDoneHook (
 void)
```

Callback that gets called at the end of the NVM programming session.

**Returns**

BLT\_TRUE is successful, BLT\_FALSE otherwise.

**7.581.2.13 NvmEraseHook()**

```
blt_int8u NvmEraseHook (
 blt_addr addr,
 blt_int32u len)
```

Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the memory hasn't been erased yet.

**Parameters**

|             |                  |
|-------------|------------------|
| <i>addr</i> | Start address.   |
| <i>len</i>  | Length in bytes. |

**Returns**

BLT\_NVM\_OKAY if successful, BLT\_NVM\_NOT\_IN\_RANGE if the address is not within the supported memory range, or BLT\_NVM\_ERROR if the erase operation failed.

**7.581.2.14 NvmInitHook()**

```
void NvmInitHook (
 void)
```

Callback that gets called at the start of the internal NVM driver initialization routine.

**Returns**

none.

**7.581.2.15 NvmReinitHook()**

```
void NvmReinitHook (
 void)
```

Callback that gets called at the start of a firmware update to reinitialize the NVM driver.

**Returns**

none.

**7.581.2.16 NvmWriteHook()**

```
blt_int8u NvmWriteHook (
 blt_addr addr,
 blt_int32u len,
 blt_int8u * data)
```

Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the data hasn't been written yet.



## Parameters

|             |                             |
|-------------|-----------------------------|
| <i>addr</i> | Start address.              |
| <i>len</i>  | Length in bytes.            |
| <i>data</i> | Pointer to the data buffer. |

## Returns

BLT\_NVM\_OKAY if successful, BLT\_NVM\_NOT\_IN\_RANGE if the address is not within the supported memory range, or BLT\_NVM\_ERROR if the write operation failed.

## 7.581.2.17 XcpGetSeedHook()

```
blt_int8u XcpGetSeedHook (
 blt_int8u resource,
 blt_int8u * seed)
```

Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the GET\_SEED command.

## Parameters

|                 |                                                        |
|-----------------|--------------------------------------------------------|
| <i>resource</i> | Resource that the seed is requested for (XCP_RES_XXX). |
| <i>seed</i>     | Pointer to byte buffer where the seed will be stored.  |

## Returns

Length of the seed in bytes.

## 7.581.2.18 XcpVerifyKeyHook()

```
blt_int8u XcpVerifyKeyHook (
 blt_int8u resource,
 blt_int8u * key,
 blt_int8u len)
```

Called by the UNLOCK command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

## Parameters

|                 |                                             |
|-----------------|---------------------------------------------|
| <i>resource</i> | resource to unlock (XCP_RES_XXX).           |
| <i>key</i>      | pointer to the byte buffer holding the key. |
| <i>len</i>      | length of the key in bytes.                 |

**Returns**

1 if the key was correct, 0 otherwise.

**7.581.3 Variable Documentation****7.581.3.1 canUse**

`blt_bool` canUse

Flag to indicate if the log-file can be used.

**7.581.3.2 handle**

`FIL` handle

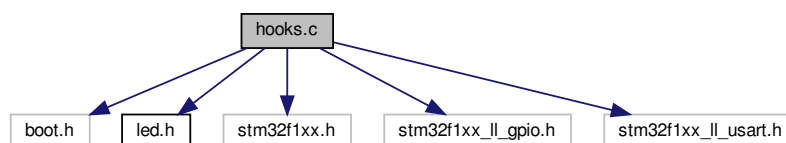
FatFS handle to the log-file.

**7.582 hooks.c File Reference**

Bootloader callback source file.

```
#include "boot.h"
#include "led.h"
#include "stm32f1xx.h"
#include "stm32f1xx_ll_gpio.h"
#include "stm32f1xx_ll_usart.h"
```

Include dependency graph for ARMCM3\_STM32F1\_Olimex\_STM32P103\_GCC/Boot/hooks.c:



## Functions

- void [BackDoorInitHook](#) (void)  
*Initializes the backdoor entry option.*
- [blt\\_bool BackDoorEntryHook](#) (void)  
*Checks if a backdoor entry is requested.*
- [blt\\_bool CpuUserProgramStartHook](#) (void)  
*Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.*
- void [CopInitHook](#) (void)  
*Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.*
- void [CopServiceHook](#) (void)  
*Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.*
- void [NvmInitHook](#) (void)  
*Callback that gets called at the start of the internal NVM driver initialization routine.*
- void [NvmReinitHook](#) (void)  
*Callback that gets called at the start of a firmware update to reinitialize the NVM driver.*
- [blt\\_int8u NvmWriteHook](#) ([blt\\_addr](#) addr, [blt\\_int32u](#) len, [blt\\_int8u](#) \*data)  
*Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the data hasn't been written yet.*
- [blt\\_int8u NvmEraseHook](#) ([blt\\_addr](#) addr, [blt\\_int32u](#) len)  
*Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the memory hasn't been erased yet.*
- [blt\\_bool NvmDoneHook](#) (void)  
*Callback that gets called at the end of the NVM programming session.*
- [blt\\_bool FileIsFirmwareUpdateRequestedHook](#) (void)  
*Callback that gets called to check whether a firmware update from local file storage should be started. This could for example be when a switch is pressed, when a certain file is found on the local file storage, etc.*
- const [blt\\_char](#) \* [FileGetFirmwareFilenameHook](#) (void)  
*Callback to obtain the filename of the firmware file that should be used during the firmware update from the local file storage. This hook function is called at the beginning of the firmware update from local storage sequence.*
- void [FileFirmwareUpdateStartedHook](#) (void)  
*Callback that gets called to inform the application that a firmware update from local storage just started.*
- void [FileFirmwareUpdateCompletedHook](#) (void)  
*Callback that gets called to inform the application that a firmware update was successfully completed.*
- void [FileFirmwareUpdateErrorHook](#) ([blt\\_int8u](#) error\_code)  
*Callback that gets called in case an error occurred during a firmware update. Refer to <file.h> for a list of available error codes.*
- void [FileFirmwareUpdateLogHook](#) ([blt\\_char](#) \*info\_string)  
*Callback that gets called each time new log information becomes available during a firmware update.*
- [blt\\_int8u XcpGetSeedHook](#) ([blt\\_int8u](#) resource, [blt\\_int8u](#) \*seed)  
*Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the GET\_SEED command.*
- [blt\\_int8u XcpVerifyKeyHook](#) ([blt\\_int8u](#) resource, [blt\\_int8u](#) \*key, [blt\\_int8u](#) len)  
*Called by the UNLOCK command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.*

## Variables

- static const `blt_char firmwareFilename []` = `"/demoprogram_olimex_stm32p103.srec"`  
*Firmware filename.*
- ```
struct {  
    FIL handle  
    blt_bool canUse  
} logfile
```

Data structure for grouping log-file related information.

7.582.1 Detailed Description

Bootloader callback source file.

7.582.2 Function Documentation

7.582.2.1 BackDoorEntryHook()

```
blt_bool BackDoorEntryHook (  
    void )
```

Checks if a backdoor entry is requested.

Returns

BLT_TRUE if the backdoor entry is requested, BLT_FALSE otherwise.

7.582.2.2 BackDoorInitHook()

```
void BackDoorInitHook (  
    void )
```

Initializes the backdoor entry option.

Returns

none.

7.582.2.3 CopInitHook()

```
void CopInitHook (
    void )
```

Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.

Returns

none.

7.582.2.4 CopServiceHook()

```
void CopServiceHook (
    void )
```

Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.

Returns

none.

7.582.2.5 CpuUserProgramStartHook()

```
blt_bool CpuUserProgramStartHook (
    void )
```

Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.

Returns

BLT_TRUE if it is okay to start the user program, BLT_FALSE to keep keep the bootloader active.

7.582.2.6 FileFirmwareUpdateCompletedHook()

```
void FileFirmwareUpdateCompletedHook (
    void )
```

Callback that gets called to inform the application that a firmware update was successfully completed.

Returns

none.

7.582.2.7 FileFirmwareUpdateErrorHook()

```
void FileFirmwareUpdateErrorHook (
    blt_int8u error_code )
```

Callback that gets called in case an error occurred during a firmware update. Refer to <[file.h](#)> for a list of available error codes.

Returns

none.

7.582.2.8 FileFirmwareUpdateLogHook()

```
void FileFirmwareUpdateLogHook (
    blt_char * info_string )
```

Callback that gets called each time new log information becomes available during a firmware update.

Parameters

<i>info_string</i>	Pointer to a character array with the log entry info.
--------------------	---

Returns

none.

7.582.2.9 FileFirmwareUpdateStartedHook()

```
void FileFirmwareUpdateStartedHook (
    void )
```

Callback that gets called to inform the application that a firmware update from local storage just started.

Returns

none.

7.582.2.10 FileGetFirmwareFilenameHook()

```
const blt_char* FileGetFirmwareFilenameHook (
    void )
```

Callback to obtain the filename of the firmware file that should be used during the firmware update from the local file storage. This hook function is called at the beginning of the firmware update from local storage sequence.

Returns

valid firmware filename with full path or BLT_NULL.

7.582.2.11 FileIsFirmwareUpdateRequestedHook()

```
blt_bool FileIsFirmwareUpdateRequestedHook (
    void )
```

Callback that gets called to check whether a firmware update from local file storage should be started. This could for example be when a switch is pressed, when a certain file is found on the local file storage, etc.

Returns

BLT_TRUE if a firmware update is requested, BLT_FALSE otherwise.

7.582.2.12 NvmDoneHook()

```
blt_bool NvmDoneHook (
    void )
```

Callback that gets called at the end of the NVM programming session.

Returns

BLT_TRUE is successful, BLT_FALSE otherwise.

7.582.2.13 NvmEraseHook()

```
blt_int8u NvmEraseHook (
    blt_addr addr,
    blt_int32u len )
```

Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the memory hasn't been erased yet.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.

Returns

BLT_NVM_OKAY if successful, BLT_NVM_NOT_IN_RANGE if the address is not within the supported memory range, or BLT_NVM_ERROR if the erase operation failed.

7.582.2.14 NvmInitHook()

```
void NvmInitHook (
    void )
```

Callback that gets called at the start of the internal NVM driver initialization routine.

Returns

none.

7.582.2.15 NvmReinitHook()

```
void NvmReinitHook (
    void )
```

Callback that gets called at the start of a firmware update to reinitialize the NVM driver.

Returns

none.

7.582.2.16 NvmWriteHook()

```
blt_int8u NvmWriteHook (
    blt_addr addr,
    blt_int32u len,
    blt_int8u * data )
```

Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the data hasn't been written yet.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.
<i>data</i>	Pointer to the data buffer.

Returns

BLT_NVM_OKAY if successful, BLT_NVM_NOT_IN_RANGE if the address is not within the supported memory range, or BLT_NVM_ERROR if the write operation failed.

7.582.2.17 XcpGetSeedHook()

```
blt_int8u XcpGetSeedHook (
    blt_int8u resource,
    blt_int8u * seed )
```

Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the GET_SEED command.

Parameters

<i>resource</i>	Resource that the seed is requested for (XCP_RES_XXX).
<i>seed</i>	Pointer to byte buffer where the seed will be stored.

Returns

Length of the seed in bytes.

7.582.2.18 XcpVerifyKeyHook()

```
blt_int8u XcpVerifyKeyHook (
    blt_int8u resource,
    blt_int8u * key,
    blt_int8u len )
```

Called by the UNLOCK command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

Parameters

<i>resource</i>	resource to unlock (XCP_RES_XXX).
<i>key</i>	pointer to the byte buffer holding the key.
<i>len</i>	length of the key in bytes.

Returns

1 if the key was correct, 0 otherwise.

7.582.3 Variable Documentation

7.582.3.1 canUse

```
blt_bool canUse
```

Flag to indicate if the log-file can be used.

7.582.3.2 handle

FIL handle

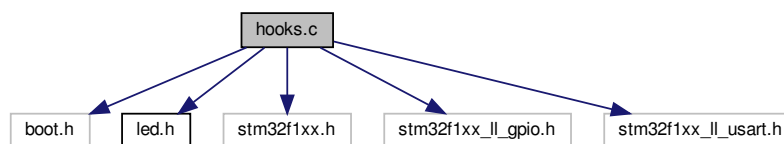
FatFS handle to the log-file.

7.583 hooks.c File Reference

Bootloader callback source file.

```
#include "boot.h"
#include "led.h"
#include "stm32f1xx.h"
#include "stm32f1xx_ll_gpio.h"
#include "stm32f1xx_ll_usart.h"
```

Include dependency graph for ARMCM3_STM32F1_Olimex_STM32P103_IAR/Boot/hooks.c:



Functions

- void [BackDoorInitHook](#) (void)
Initializes the backdoor entry option.
- [blt_bool](#) [BackDoorEntryHook](#) (void)
Checks if a backdoor entry is requested.
- [blt_bool](#) [CpuUserProgramStartHook](#) (void)
Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.
- void [CopInitHook](#) (void)
Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.
- void [CopServiceHook](#) (void)
Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.
- void [NvmInitHook](#) (void)
Callback that gets called at the start of the internal NVM driver initialization routine.
- void [NvmReinitHook](#) (void)
Callback that gets called at the start of a firmware update to reinitialize the NVM driver.
- [blt_int8u](#) [NvmWriteHook](#) ([blt_addr](#) addr, [blt_int32u](#) len, [blt_int8u](#) *data)
Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the data hasn't been written yet.

- [blt_int8u NvmEraseHook](#) ([blt_addr](#) addr, [blt_int32u](#) len)
Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then `BLT_NVM_NOT_IN_RANGE` must be returned to indicate that the memory hasn't been erased yet.
- [blt_bool NvmDoneHook](#) (void)
Callback that gets called at the end of the NVM programming session.
- [blt_bool FileIsFirmwareUpdateRequestedHook](#) (void)
Callback that gets called to check whether a firmware update from local file storage should be started. This could for example be when a switch is pressed, when a certain file is found on the local file storage, etc.
- const [blt_char](#) * [FileGetFirmwareFilenameHook](#) (void)
Callback to obtain the filename of the firmware file that should be used during the firmware update from the local file storage. This hook function is called at the beginning of the firmware update from local storage sequence.
- void [FileFirmwareUpdateStartedHook](#) (void)
Callback that gets called to inform the application that a firmware update from local storage just started.
- void [FileFirmwareUpdateCompletedHook](#) (void)
Callback that gets called to inform the application that a firmware update was successfully completed.
- void [FileFirmwareUpdateErrorHook](#) ([blt_int8u](#) error_code)
Callback that gets called in case an error occurred during a firmware update. Refer to [<file.h>](#) for a list of available error codes.
- void [FileFirmwareUpdateLogHook](#) ([blt_char](#) *info_string)
Callback that gets called each time new log information becomes available during a firmware update.
- [blt_int8u XcpGetSeedHook](#) ([blt_int8u](#) resource, [blt_int8u](#) *seed)
Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the `GET_SEED` command.
- [blt_int8u XcpVerifyKeyHook](#) ([blt_int8u](#) resource, [blt_int8u](#) *key, [blt_int8u](#) len)
Called by the `UNLOCK` command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

Variables

- static const [blt_char](#) [firmwareFilename](#) [] = "/demoprogram_olimex_stm32p103.srec"
Firmware filename.
- ```

struct {
 FIL handle
 blt_bool canUse
} logfile

```

*Data structure for grouping log-file related information.*

## 7.583.1 Detailed Description

Bootloader callback source file.

## 7.583.2 Function Documentation

#### 7.583.2.1 BackDoorEntryHook()

```
blt_bool BackDoorEntryHook (
 void)
```

Checks if a backdoor entry is requested.

##### Returns

BLT\_TRUE if the backdoor entry is requested, BLT\_FALSE otherwise.

#### 7.583.2.2 BackDoorInitHook()

```
void BackDoorInitHook (
 void)
```

Initializes the backdoor entry option.

##### Returns

none.

#### 7.583.2.3 CopInitHook()

```
void CopInitHook (
 void)
```

Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.

##### Returns

none.

#### 7.583.2.4 CopServiceHook()

```
void CopServiceHook (
 void)
```

Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.

##### Returns

none.

#### 7.583.2.5 CpuUserProgramStartHook()

```
blt_bool CpuUserProgramStartHook (
 void)
```

Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.

##### Returns

BLT\_TRUE if it is okay to start the user program, BLT\_FALSE to keep keep the bootloader active.

#### 7.583.2.6 FileFirmwareUpdateCompletedHook()

```
void FileFirmwareUpdateCompletedHook (
 void)
```

Callback that gets called to inform the application that a firmware update was successfully completed.

##### Returns

none.

#### 7.583.2.7 FileFirmwareUpdateErrorHook()

```
void FileFirmwareUpdateErrorHook (
 blt_int8u error_code)
```

Callback that gets called in case an error occurred during a firmware update. Refer to <[file.h](#)> for a list of available error codes.

##### Returns

none.

#### 7.583.2.8 FileFirmwareUpdateLogHook()

```
void FileFirmwareUpdateLogHook (
 blt_char * info_string)
```

Callback that gets called each time new log information becomes available during a firmware update.

**Parameters**

|                    |                                                       |
|--------------------|-------------------------------------------------------|
| <i>info_string</i> | Pointer to a character array with the log entry info. |
|--------------------|-------------------------------------------------------|

**Returns**

none.

**7.583.2.9 FileFirmwareUpdateStartedHook()**

```
void FileFirmwareUpdateStartedHook (
 void)
```

Callback that gets called to inform the application that a firmware update from local storage just started.

**Returns**

none.

**7.583.2.10 FileGetFirmwareFilenameHook()**

```
const blt_char* FileGetFirmwareFilenameHook (
 void)
```

Callback to obtain the filename of the firmware file that should be used during the firmware update from the local file storage. This hook function is called at the beginning of the firmware update from local storage sequence.

**Returns**

valid firmware filename with full path or BLT\_NULL.

**7.583.2.11 FileIsFirmwareUpdateRequestedHook()**

```
blt_bool FileIsFirmwareUpdateRequestedHook (
 void)
```

Callback that gets called to check whether a firmware update from local file storage should be started. This could for example be when a switch is pressed, when a certain file is found on the local file storage, etc.

**Returns**

BLT\_TRUE if a firmware update is requested, BLT\_FALSE otherwise.

### 7.583.2.12 NvmDoneHook()

```
blt_bool NvmDoneHook (
 void)
```

Callback that gets called at the end of the NVM programming session.

#### Returns

BLT\_TRUE is successful, BLT\_FALSE otherwise.

### 7.583.2.13 NvmEraseHook()

```
blt_int8u NvmEraseHook (
 blt_addr addr,
 blt_int32u len)
```

Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the memory hasn't been erased yet.

#### Parameters

|             |                  |
|-------------|------------------|
| <i>addr</i> | Start address.   |
| <i>len</i>  | Length in bytes. |

#### Returns

BLT\_NVM\_OKAY if successful, BLT\_NVM\_NOT\_IN\_RANGE if the address is not within the supported memory range, or BLT\_NVM\_ERROR is the erase operation failed.

### 7.583.2.14 NvmInitHook()

```
void NvmInitHook (
 void)
```

Callback that gets called at the start of the internal NVM driver initialization routine.

#### Returns

none.

### 7.583.2.15 NvmReinitHook()

```
void NvmReinitHook (
 void)
```

Callback that gets called at the start of a firmware update to reinitialize the NVM driver.

#### Returns

none.

### 7.583.2.16 NvmWriteHook()

```
blt_int8u NvmWriteHook (
 blt_addr addr,
 blt_int32u len,
 blt_int8u * data)
```

Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the data hasn't been written yet.

#### Parameters

|             |                             |
|-------------|-----------------------------|
| <i>addr</i> | Start address.              |
| <i>len</i>  | Length in bytes.            |
| <i>data</i> | Pointer to the data buffer. |

#### Returns

BLT\_NVM\_OKAY if successful, BLT\_NVM\_NOT\_IN\_RANGE if the address is not within the supported memory range, or BLT\_NVM\_ERROR if the write operation failed.

### 7.583.2.17 XcpGetSeedHook()

```
blt_int8u XcpGetSeedHook (
 blt_int8u resource,
 blt_int8u * seed)
```

Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the GET\_SEED command.

#### Parameters

|                 |                                                        |
|-----------------|--------------------------------------------------------|
| <i>resource</i> | Resource that the seed is requested for (XCP_RES_XXX). |
| <i>seed</i>     | Pointer to byte buffer where the seed will be stored.  |



**Returns**

Length of the seed in bytes.

**7.583.2.18 XcpVerifyKeyHook()**

```
blt_int8u XcpVerifyKeyHook (
 blt_int8u resource,
 blt_int8u * key,
 blt_int8u len)
```

Called by the UNLOCK command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

**Parameters**

|                 |                                             |
|-----------------|---------------------------------------------|
| <i>resource</i> | resource to unlock (XCP_RES_XXX).           |
| <i>key</i>      | pointer to the byte buffer holding the key. |
| <i>len</i>      | length of the key in bytes.                 |

**Returns**

1 if the key was correct, 0 otherwise.

**7.583.3 Variable Documentation****7.583.3.1 canUse**

```
blt_bool canUse
```

Flag to indicate if the log-file can be used.

**7.583.3.2 handle**

```
FIL handle
```

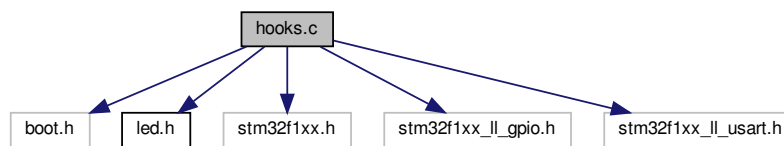
FatFS handle to the log-file.

## 7.584 hooks.c File Reference

Bootloader callback source file.

```
#include "boot.h"
#include "led.h"
#include "stm32f1xx.h"
#include "stm32f1xx_ll_gpio.h"
#include "stm32f1xx_ll_usart.h"
```

Include dependency graph for ARMCM3\_STM32F1\_Olimex\_STM32P103\_Keil/Boot/hooks.c:



## Functions

- void [BackDoorInitHook](#) (void)  
*Initializes the backdoor entry option.*
- [blt\\_bool BackDoorEntryHook](#) (void)  
*Checks if a backdoor entry is requested.*
- [blt\\_bool CpuUserProgramStartHook](#) (void)  
*Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.*
- void [CopInitHook](#) (void)  
*Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.*
- void [CopServiceHook](#) (void)  
*Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.*
- void [NvmInitHook](#) (void)  
*Callback that gets called at the start of the internal NVM driver initialization routine.*
- void [NvmReinitHook](#) (void)  
*Callback that gets called at the start of a firmware update to reinitialize the NVM driver.*
- [blt\\_int8u NvmWriteHook](#) ([blt\\_addr](#) addr, [blt\\_int32u](#) len, [blt\\_int8u](#) \*data)  
*Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the data hasn't been written yet.*
- [blt\\_int8u NvmEraseHook](#) ([blt\\_addr](#) addr, [blt\\_int32u](#) len)  
*Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the memory hasn't been erased yet.*
- [blt\\_bool NvmDoneHook](#) (void)  
*Callback that gets called at the end of the NVM programming session.*
- [blt\\_bool FileIsFirmwareUpdateRequestedHook](#) (void)

Callback that gets called to check whether a firmware update from local file storage should be started. This could for example be when a switch is pressed, when a certain file is found on the local file storage, etc.

- const `blt_char * FileGetFirmwareFilenameHook` (void)

Callback to obtain the filename of the firmware file that should be used during the firmware update from the local file storage. This hook function is called at the beginning of the firmware update from local storage sequence.

- void `FileFirmwareUpdateStartedHook` (void)

Callback that gets called to inform the application that a firmware update from local storage just started.

- void `FileFirmwareUpdateCompletedHook` (void)

Callback that gets called to inform the application that a firmware update was successfully completed.

- void `FileFirmwareUpdateErrorHook` (blt\_int8u error\_code)

Callback that gets called in case an error occurred during a firmware update. Refer to [<file.h>](#) for a list of available error codes.

- void `FileFirmwareUpdateLogHook` (blt\_char \*info\_string)

Callback that gets called each time new log information becomes available during a firmware update.

- blt\_int8u `XcpGetSeedHook` (blt\_int8u resource, blt\_int8u \*seed)

Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the `GET_SEED` command.

- blt\_int8u `XcpVerifyKeyHook` (blt\_int8u resource, blt\_int8u \*key, blt\_int8u len)

Called by the `UNLOCK` command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

## Variables

- static const `blt_char firmwareFilename` [] = `"/demoprogram_olimex_stm32p103.srec"`

Firmware filename.

- 

```
struct {
 FIL handle
 blt_bool canUse
} logfile
```

Data structure for grouping log-file related information.

## 7.584.1 Detailed Description

Bootloader callback source file.

## 7.584.2 Function Documentation

### 7.584.2.1 BackDoorEntryHook()

```
blt_bool BackDoorEntryHook (
 void)
```

Checks if a backdoor entry is requested.

#### Returns

BLT\_TRUE if the backdoor entry is requested, BLT\_FALSE otherwise.

#### 7.584.2.2 BackDoorInitHook()

```
void BackDoorInitHook (
 void)
```

Initializes the backdoor entry option.

##### Returns

none.

#### 7.584.2.3 CopInitHook()

```
void CopInitHook (
 void)
```

Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.

##### Returns

none.

#### 7.584.2.4 CopServiceHook()

```
void CopServiceHook (
 void)
```

Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.

##### Returns

none.

#### 7.584.2.5 CpuUserProgramStartHook()

```
blt_bool CpuUserProgramStartHook (
 void)
```

Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.

##### Returns

BLT\_TRUE if it is okay to start the user program, BLT\_FALSE to keep keep the bootloader active.

#### 7.584.2.6 FileFirmwareUpdateCompletedHook()

```
void FileFirmwareUpdateCompletedHook (
 void)
```

Callback that gets called to inform the application that a firmware update was successfully completed.

##### Returns

none.

#### 7.584.2.7 FileFirmwareUpdateErrorHook()

```
void FileFirmwareUpdateErrorHook (
 blt_int8u error_code)
```

Callback that gets called in case an error occurred during a firmware update. Refer to <[file.h](#)> for a list of available error codes.

##### Returns

none.

#### 7.584.2.8 FileFirmwareUpdateLogHook()

```
void FileFirmwareUpdateLogHook (
 blt_char * info_string)
```

Callback that gets called each time new log information becomes available during a firmware update.

##### Parameters

|                    |                                                       |
|--------------------|-------------------------------------------------------|
| <i>info_string</i> | Pointer to a character array with the log entry info. |
|--------------------|-------------------------------------------------------|

##### Returns

none.

#### 7.584.2.9 FileFirmwareUpdateStartedHook()

```
void FileFirmwareUpdateStartedHook (
 void)
```

Callback that gets called to inform the application that a firmware update from local storage just started.

**Returns**

none.

**7.584.2.10 FileGetFirmwareFilenameHook()**

```
const blt_char* FileGetFirmwareFilenameHook (
 void)
```

Callback to obtain the filename of the firmware file that should be used during the firmware update from the local file storage. This hook function is called at the beginning of the firmware update from local storage sequence.

**Returns**

valid firmware filename with full path or BLT\_NULL.

**7.584.2.11 FileIsFirmwareUpdateRequestedHook()**

```
blt_bool FileIsFirmwareUpdateRequestedHook (
 void)
```

Callback that gets called to check whether a firmware update from local file storage should be started. This could for example be when a switch is pressed, when a certain file is found on the local file storage, etc.

**Returns**

BLT\_TRUE if a firmware update is requested, BLT\_FALSE otherwise.

**7.584.2.12 NvmDoneHook()**

```
blt_bool NvmDoneHook (
 void)
```

Callback that gets called at the end of the NVM programming session.

**Returns**

BLT\_TRUE is successful, BLT\_FALSE otherwise.

**7.584.2.13 NvmEraseHook()**

```
blt_int8u NvmEraseHook (
 blt_addr addr,
 blt_int32u len)
```

Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the memory hasn't been erased yet.

## Parameters

|             |                  |
|-------------|------------------|
| <i>addr</i> | Start address.   |
| <i>len</i>  | Length in bytes. |

## Returns

BLT\_NVM\_OKAY if successful, BLT\_NVM\_NOT\_IN\_RANGE if the address is not within the supported memory range, or BLT\_NVM\_ERROR if the erase operation failed.

## 7.584.2.14 NvmInitHook()

```
void NvmInitHook (
 void)
```

Callback that gets called at the start of the internal NVM driver initialization routine.

## Returns

none.

## 7.584.2.15 NvmReinitHook()

```
void NvmReinitHook (
 void)
```

Callback that gets called at the start of a firmware update to reinitialize the NVM driver.

## Returns

none.

## 7.584.2.16 NvmWriteHook()

```
blt_int8u NvmWriteHook (
 blt_addr addr,
 blt_int32u len,
 blt_int8u * data)
```

Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the data hasn't been written yet.

**Parameters**

|             |                             |
|-------------|-----------------------------|
| <i>addr</i> | Start address.              |
| <i>len</i>  | Length in bytes.            |
| <i>data</i> | Pointer to the data buffer. |

**Returns**

BLT\_NVM\_OKAY if successful, BLT\_NVM\_NOT\_IN\_RANGE if the address is not within the supported memory range, or BLT\_NVM\_ERROR if the write operation failed.

**7.584.2.17 XcpGetSeedHook()**

```
blt_int8u XcpGetSeedHook (
 blt_int8u resource,
 blt_int8u * seed)
```

Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the GET\_SEED command.

**Parameters**

|                 |                                                        |
|-----------------|--------------------------------------------------------|
| <i>resource</i> | Resource that the seed is requested for (XCP_RES_XXX). |
| <i>seed</i>     | Pointer to byte buffer where the seed will be stored.  |

**Returns**

Length of the seed in bytes.

**7.584.2.18 XcpVerifyKeyHook()**

```
blt_int8u XcpVerifyKeyHook (
 blt_int8u resource,
 blt_int8u * key,
 blt_int8u len)
```

Called by the UNLOCK command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

**Parameters**

|                 |                                             |
|-----------------|---------------------------------------------|
| <i>resource</i> | resource to unlock (XCP_RES_XXX).           |
| <i>key</i>      | pointer to the byte buffer holding the key. |
| <i>len</i>      | length of the key in bytes.                 |



### Returns

1 if the key was correct, 0 otherwise.

## 7.584.3 Variable Documentation

### 7.584.3.1 canUse

`blt_bool` canUse

Flag to indicate if the log-file can be used.

### 7.584.3.2 handle

FIL handle

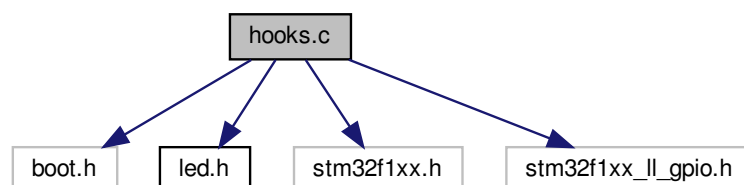
FatFS handle to the log-file.

## 7.585 hooks.c File Reference

Bootloader callback source file.

```
#include "boot.h"
#include "led.h"
#include "stm32f1xx.h"
#include "stm32f1xx_ll_gpio.h"
```

Include dependency graph for ARMCM3\_STM32F1\_Olimexino\_STM32\_CubeIDE/Boot/App/hooks.c:



## Functions

- void [UsbConnectHook](#) ([blt\\_bool](#) connect)  
*Callback that gets called whenever the USB device should be connected to the USB bus.*
- void [UsbEnterLowPowerModeHook](#) (void)  
*Callback that gets called whenever the USB host requests the device to enter a low power mode.*
- void [UsbLeaveLowPowerModeHook](#) (void)  
*Callback that gets called whenever the USB host requests the device to exit low power mode.*
- void [BackDoorInitHook](#) (void)  
*Initializes the backdoor entry option.*
- [blt\\_bool](#) [BackDoorEntryHook](#) (void)  
*Checks if a backdoor entry is requested.*
- [blt\\_bool](#) [CpuUserProgramStartHook](#) (void)  
*Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.*
- void [CopInitHook](#) (void)  
*Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.*
- void [CopServiceHook](#) (void)  
*Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.*
- void [NvmInitHook](#) (void)  
*Callback that gets called at the start of the internal NVM driver initialization routine.*
- void [NvmReinitHook](#) (void)  
*Callback that gets called at the start of a firmware update to reinitialize the NVM driver.*
- [blt\\_int8u](#) [NvmWriteHook](#) ([blt\\_addr](#) addr, [blt\\_int32u](#) len, [blt\\_int8u](#) \*data)  
*Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the data hasn't been written yet.*
- [blt\\_int8u](#) [NvmEraseHook](#) ([blt\\_addr](#) addr, [blt\\_int32u](#) len)  
*Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the memory hasn't been erased yet.*
- [blt\\_bool](#) [NvmDoneHook](#) (void)  
*Callback that gets called at the end of the NVM programming session.*
- [blt\\_bool](#) [FileIsFirmwareUpdateRequestedHook](#) (void)  
*Callback that gets called to check whether a firmware update from local file storage should be started. This could for example be when a switch is pressed, when a certain file is found on the local file storage, etc.*
- const [blt\\_char](#) \* [FileGetFirmwareFilenameHook](#) (void)  
*Callback to obtain the filename of the firmware file that should be used during the firmware update from the local file storage. This hook function is called at the beginning of the firmware update from local storage sequence.*
- void [FileFirmwareUpdateStartedHook](#) (void)  
*Callback that gets called to inform the application that a firmware update from local storage just started.*
- void [FileFirmwareUpdateCompletedHook](#) (void)  
*Callback that gets called to inform the application that a firmware update was successfully completed.*
- void [FileFirmwareUpdateErrorHook](#) ([blt\\_int8u](#) error\_code)  
*Callback that gets called in case an error occurred during a firmware update. Refer to [<file.h>](#) for a list of available error codes.*
- void [FileFirmwareUpdateLogHook](#) ([blt\\_char](#) \*info\_string)  
*Callback that gets called each time new log information becomes available during a firmware update.*
- [blt\\_int8u](#) [XcpGetSeedHook](#) ([blt\\_int8u](#) resource, [blt\\_int8u](#) \*seed)  
*Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the GET\_SEED command.*

- `blt_int8u XcpVerifyKeyHook (blt_int8u resource, blt_int8u *key, blt_int8u len)`

*Called by the UNLOCK command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.*

## Variables

- static const `blt_char firmwareFilename [] = "/demoprogram_olimexino_stm32.srec"`

*Firmware filename.*

- 

```
struct {
 FIL handle
 blt_bool canUse
} logfile
```

*Data structure for grouping log-file related information.*

## 7.585.1 Detailed Description

Bootloader callback source file.

## 7.585.2 Function Documentation

### 7.585.2.1 BackDoorEntryHook()

```
blt_bool BackDoorEntryHook (
 void)
```

Checks if a backdoor entry is requested.

#### Returns

BLT\_TRUE if the backdoor entry is requested, BLT\_FALSE otherwise.

### 7.585.2.2 BackDoorInitHook()

```
void BackDoorInitHook (
 void)
```

Initializes the backdoor entry option.

#### Returns

none.

### 7.585.2.3 CopInitHook()

```
void CopInitHook (
 void)
```

Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.

#### Returns

none.

### 7.585.2.4 CopServiceHook()

```
void CopServiceHook (
 void)
```

Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.

#### Returns

none.

### 7.585.2.5 CpuUserProgramStartHook()

```
blt_bool CpuUserProgramStartHook (
 void)
```

Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.

#### Returns

BLT\_TRUE if it is okay to start the user program, BLT\_FALSE to keep keep the bootloader active.

### 7.585.2.6 FileFirmwareUpdateCompletedHook()

```
void FileFirmwareUpdateCompletedHook (
 void)
```

Callback that gets called to inform the application that a firmware update was successfully completed.

#### Returns

none.

#### 7.585.2.7 FileFirmwareUpdateErrorHook()

```
void FileFirmwareUpdateErrorHook (
 blt_int8u error_code)
```

Callback that gets called in case an error occurred during a firmware update. Refer to [<file.h>](#) for a list of available error codes.

##### Returns

none.

#### 7.585.2.8 FileFirmwareUpdateLogHook()

```
void FileFirmwareUpdateLogHook (
 blt_char * info_string)
```

Callback that gets called each time new log information becomes available during a firmware update.

##### Parameters

|                    |                                                       |
|--------------------|-------------------------------------------------------|
| <i>info_string</i> | Pointer to a character array with the log entry info. |
|--------------------|-------------------------------------------------------|

##### Returns

none.

#### 7.585.2.9 FileFirmwareUpdateStartedHook()

```
void FileFirmwareUpdateStartedHook (
 void)
```

Callback that gets called to inform the application that a firmware update from local storage just started.

##### Returns

none.

#### 7.585.2.10 FileGetFirmwareFilenameHook()

```
const blt_char* FileGetFirmwareFilenameHook (
 void)
```

Callback to obtain the filename of the firmware file that should be used during the firmware update from the local file storage. This hook function is called at the beginning of the firmware update from local storage sequence.

##### Returns

valid firmware filename with full path or BLT\_NULL.

### 7.585.2.11 FileIsFirmwareUpdateRequestedHook()

```
blt_bool FileIsFirmwareUpdateRequestedHook (
 void)
```

Callback that gets called to check whether a firmware update from local file storage should be started. This could for example be when a switch is pressed, when a certain file is found on the local file storage, etc.

#### Returns

BLT\_TRUE if a firmware update is requested, BLT\_FALSE otherwise.

### 7.585.2.12 NvmDoneHook()

```
blt_bool NvmDoneHook (
 void)
```

Callback that gets called at the end of the NVM programming session.

#### Returns

BLT\_TRUE is successful, BLT\_FALSE otherwise.

### 7.585.2.13 NvmEraseHook()

```
blt_int8u NvmEraseHook (
 blt_addr addr,
 blt_int32u len)
```

Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the memory hasn't been erased yet.

#### Parameters

|             |                  |
|-------------|------------------|
| <i>addr</i> | Start address.   |
| <i>len</i>  | Length in bytes. |

#### Returns

BLT\_NVM\_OKAY if successful, BLT\_NVM\_NOT\_IN\_RANGE if the address is not within the supported memory range, or BLT\_NVM\_ERROR if the erase operation failed.

#### 7.585.2.14 NvmInitHook()

```
void NvmInitHook (
 void)
```

Callback that gets called at the start of the internal NVM driver initialization routine.

##### Returns

none.

#### 7.585.2.15 NvmReinitHook()

```
void NvmReinitHook (
 void)
```

Callback that gets called at the start of a firmware update to reinitialize the NVM driver.

##### Returns

none.

#### 7.585.2.16 NvmWriteHook()

```
blt_int8u NvmWriteHook (
 blt_addr addr,
 blt_int32u len,
 blt_int8u * data)
```

Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the data hasn't been written yet.

##### Parameters

|             |                             |
|-------------|-----------------------------|
| <i>addr</i> | Start address.              |
| <i>len</i>  | Length in bytes.            |
| <i>data</i> | Pointer to the data buffer. |

##### Returns

BLT\_NVM\_OKAY if successful, BLT\_NVM\_NOT\_IN\_RANGE if the address is not within the supported memory range, or BLT\_NVM\_ERROR if the write operation failed.

#### 7.585.2.17 UsbConnectHook()

```
void UsbConnectHook (
 blt_bool connect)
```

Callback that gets called whenever the USB device should be connected to the USB bus.

##### Parameters

|                |                                                  |
|----------------|--------------------------------------------------|
| <i>connect</i> | BLT_TRUE to connect and BLT_FALSE to disconnect. |
|----------------|--------------------------------------------------|

##### Returns

none.

#### 7.585.2.18 UsbEnterLowPowerModeHook()

```
void UsbEnterLowPowerModeHook (
 void)
```

Callback that gets called whenever the USB host requests the device to enter a low power mode.

##### Returns

none.

#### 7.585.2.19 UsbLeaveLowPowerModeHook()

```
void UsbLeaveLowPowerModeHook (
 void)
```

Callback that gets called whenever the USB host requests the device to exit low power mode.

##### Returns

none.

#### 7.585.2.20 XcpGetSeedHook()

```
blt_int8u XcpGetSeedHook (
 blt_int8u resource,
 blt_int8u * seed)
```

Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the GET\_SEED command.



## Parameters

|                 |                                                        |
|-----------------|--------------------------------------------------------|
| <i>resource</i> | Resource that the seed is requested for (XCP_RES_XXX). |
| <i>seed</i>     | Pointer to byte buffer where the seed will be stored.  |

## Returns

Length of the seed in bytes.

## 7.585.2.21 XcpVerifyKeyHook()

```
blt_int8u XcpVerifyKeyHook (
 blt_int8u resource,
 blt_int8u * key,
 blt_int8u len)
```

Called by the UNLOCK command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

## Parameters

|                 |                                             |
|-----------------|---------------------------------------------|
| <i>resource</i> | resource to unlock (XCP_RES_XXX).           |
| <i>key</i>      | pointer to the byte buffer holding the key. |
| <i>len</i>      | length of the key in bytes.                 |

## Returns

1 if the key was correct, 0 otherwise.

## 7.585.3 Variable Documentation

## 7.585.3.1 canUse

```
blt_bool canUse
```

Flag to indicate if the log-file can be used.

## 7.585.3.2 handle

```
FIL handle
```

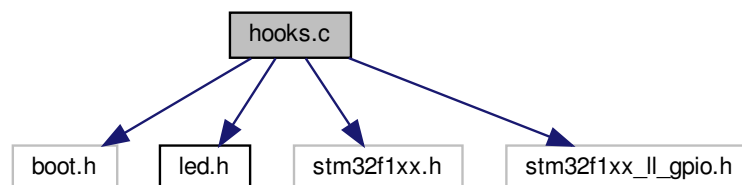
FatFS handle to the log-file.

## 7.586 hooks.c File Reference

Bootloader callback source file.

```
#include "boot.h"
#include "led.h"
#include "stm32f1xx.h"
#include "stm32f1xx_ll_gpio.h"
```

Include dependency graph for ARMC32F1\_Olimexino\_STM32\_GCC/Boot/hooks.c:



### Functions

- void [UsbConnectHook](#) ([blt\\_bool](#) connect)  
*Callback that gets called whenever the USB device should be connected to the USB bus.*
- void [UsbEnterLowPowerModeHook](#) (void)  
*Callback that gets called whenever the USB host requests the device to enter a low power mode.*
- void [UsbLeaveLowPowerModeHook](#) (void)  
*Callback that gets called whenever the USB host requests the device to exit low power mode.*
- void [BackDoorInitHook](#) (void)  
*Initializes the backdoor entry option.*
- [blt\\_bool](#) [BackDoorEntryHook](#) (void)  
*Checks if a backdoor entry is requested.*
- [blt\\_bool](#) [CpuUserProgramStartHook](#) (void)  
*Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.*
- void [CopInitHook](#) (void)  
*Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.*
- void [CopServiceHook](#) (void)  
*Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.*
- void [NvmInitHook](#) (void)  
*Callback that gets called at the start of the internal NVM driver initialization routine.*
- void [NvmReinitHook](#) (void)  
*Callback that gets called at the start of a firmware update to reinitialize the NVM driver.*
- [blt\\_int8u](#) [NvmWriteHook](#) ([blt\\_addr](#) addr, [blt\\_int32u](#) len, [blt\\_int8u](#) \*data)  
*Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the data hasn't been written yet.*

- [blt\\_int8u NvmEraseHook](#) ([blt\\_addr](#) addr, [blt\\_int32u](#) len)  
*Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then `BLT_NVM_NOT_IN_RANGE` must be returned to indicate that the memory hasn't been erased yet.*
- [blt\\_bool NvmDoneHook](#) (void)  
*Callback that gets called at the end of the NVM programming session.*
- [blt\\_bool FileIsFirmwareUpdateRequestedHook](#) (void)  
*Callback that gets called to check whether a firmware update from local file storage should be started. This could for example be when a switch is pressed, when a certain file is found on the local file storage, etc.*
- const [blt\\_char](#) \* [FileGetFirmwareFilenameHook](#) (void)  
*Callback to obtain the filename of the firmware file that should be used during the firmware update from the local file storage. This hook function is called at the beginning of the firmware update from local storage sequence.*
- void [FileFirmwareUpdateStartedHook](#) (void)  
*Callback that gets called to inform the application that a firmware update from local storage just started.*
- void [FileFirmwareUpdateCompletedHook](#) (void)  
*Callback that gets called to inform the application that a firmware update was successfully completed.*
- void [FileFirmwareUpdateErrorHook](#) ([blt\\_int8u](#) error\_code)  
*Callback that gets called in case an error occurred during a firmware update. Refer to [<file.h>](#) for a list of available error codes.*
- void [FileFirmwareUpdateLogHook](#) ([blt\\_char](#) \*info\_string)  
*Callback that gets called each time new log information becomes available during a firmware update.*
- [blt\\_int8u XcpGetSeedHook](#) ([blt\\_int8u](#) resource, [blt\\_int8u](#) \*seed)  
*Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the `GET_SEED` command.*
- [blt\\_int8u XcpVerifyKeyHook](#) ([blt\\_int8u](#) resource, [blt\\_int8u](#) \*key, [blt\\_int8u](#) len)  
*Called by the `UNLOCK` command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.*

## Variables

- static const [blt\\_char](#) [firmwareFilename](#) [] = "/demoprogram\_olimexino\_stm32.srec"  
*Firmware filename.*
- ```

struct {
    FIL handle
    blt\_bool canUse
} logfile

```

Data structure for grouping log-file related information.

7.586.1 Detailed Description

Bootloader callback source file.

7.586.2 Function Documentation

7.586.2.1 BackDoorEntryHook()

```
blt_bool BackDoorEntryHook (  
    void )
```

Checks if a backdoor entry is requested.

Returns

BLT_TRUE if the backdoor entry is requested, BLT_FALSE otherwise.

7.586.2.2 BackDoorInitHook()

```
void BackDoorInitHook (  
    void )
```

Initializes the backdoor entry option.

Returns

none.

7.586.2.3 CopInitHook()

```
void CopInitHook (  
    void )
```

Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.

Returns

none.

7.586.2.4 CopServiceHook()

```
void CopServiceHook (  
    void )
```

Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.

Returns

none.

7.586.2.5 CpuUserProgramStartHook()

```
blt_bool CpuUserProgramStartHook (  
    void )
```

Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.

Returns

BLT_TRUE if it is okay to start the user program, BLT_FALSE to keep keep the bootloader active.

7.586.2.6 FileFirmwareUpdateCompletedHook()

```
void FileFirmwareUpdateCompletedHook (  
    void )
```

Callback that gets called to inform the application that a firmware update was successfully completed.

Returns

none.

7.586.2.7 FileFirmwareUpdateErrorHook()

```
void FileFirmwareUpdateErrorHook (  
    blt_int8u error_code )
```

Callback that gets called in case an error occurred during a firmware update. Refer to <[file.h](#)> for a list of available error codes.

Returns

none.

7.586.2.8 FileFirmwareUpdateLogHook()

```
void FileFirmwareUpdateLogHook (  
    blt_char * info_string )
```

Callback that gets called each time new log information becomes available during a firmware update.

Parameters

<i>info_string</i>	Pointer to a character array with the log entry info.
--------------------	---

Returns

none.

7.586.2.9 FileFirmwareUpdateStartedHook()

```
void FileFirmwareUpdateStartedHook (  
    void )
```

Callback that gets called to inform the application that a firmware update from local storage just started.

Returns

none.

7.586.2.10 FileGetFirmwareFilenameHook()

```
const blt_char* FileGetFirmwareFilenameHook (  
    void )
```

Callback to obtain the filename of the firmware file that should be used during the firmware update from the local file storage. This hook function is called at the beginning of the firmware update from local storage sequence.

Returns

valid firmware filename with full path or BLT_NULL.

7.586.2.11 FileIsFirmwareUpdateRequestedHook()

```
blt_bool FileIsFirmwareUpdateRequestedHook (  
    void )
```

Callback that gets called to check whether a firmware update from local file storage should be started. This could for example be when a switch is pressed, when a certain file is found on the local file storage, etc.

Returns

BLT_TRUE if a firmware update is requested, BLT_FALSE otherwise.

7.586.2.12 NvmDoneHook()

```
blt_bool NvmDoneHook (
    void )
```

Callback that gets called at the end of the NVM programming session.

Returns

BLT_TRUE is successful, BLT_FALSE otherwise.

7.586.2.13 NvmEraseHook()

```
blt_int8u NvmEraseHook (
    blt_addr addr,
    blt_int32u len )
```

Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the memory hasn't been erased yet.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.

Returns

BLT_NVM_OKAY if successful, BLT_NVM_NOT_IN_RANGE if the address is not within the supported memory range, or BLT_NVM_ERROR if the erase operation failed.

7.586.2.14 NvmInitHook()

```
void NvmInitHook (
    void )
```

Callback that gets called at the start of the internal NVM driver initialization routine.

Returns

none.

7.586.2.15 NvmReinitHook()

```
void NvmReinitHook (  
    void )
```

Callback that gets called at the start of a firmware update to reinitialize the NVM driver.

Returns

none.

7.586.2.16 NvmWriteHook()

```
blt_int8u NvmWriteHook (  
    blt_addr addr,  
    blt_int32u len,  
    blt_int8u * data )
```

Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the data hasn't been written yet.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.
<i>data</i>	Pointer to the data buffer.

Returns

BLT_NVM_OKAY if successful, BLT_NVM_NOT_IN_RANGE if the address is not within the supported memory range, or BLT_NVM_ERROR if the write operation failed.

7.586.2.17 UsbConnectHook()

```
void UsbConnectHook (  
    blt_bool connect )
```

Callback that gets called whenever the USB device should be connected to the USB bus.

Parameters

<i>connect</i>	BLT_TRUE to connect and BLT_FALSE to disconnect.
----------------	--

Returns

none.

7.586.2.18 UsbEnterLowPowerModeHook()

```
void UsbEnterLowPowerModeHook (  
    void )
```

Callback that gets called whenever the USB host requests the device to enter a low power mode.

Returns

none.

7.586.2.19 UsbLeaveLowPowerModeHook()

```
void UsbLeaveLowPowerModeHook (  
    void )
```

Callback that gets called whenever the USB host requests the device to exit low power mode.

Returns

none.

7.586.2.20 XcpGetSeedHook()

```
blt_int8u XcpGetSeedHook (  
    blt_int8u resource,  
    blt_int8u * seed )
```

Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the GET_SEED command.

Parameters

<i>resource</i>	Resource that the seed is requested for (XCP_RES_XXX).
<i>seed</i>	Pointer to byte buffer where the seed will be stored.

Returns

Length of the seed in bytes.

7.586.2.21 XcpVerifyKeyHook()

```
blt_int8u XcpVerifyKeyHook (
    blt_int8u resource,
    blt_int8u * key,
    blt_int8u len )
```

Called by the UNLOCK command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

Parameters

<i>resource</i>	resource to unlock (XCP_RES_XXX).
<i>key</i>	pointer to the byte buffer holding the key.
<i>len</i>	length of the key in bytes.

Returns

1 if the key was correct, 0 otherwise.

7.586.3 Variable Documentation**7.586.3.1 canUse**

```
blt_bool canUse
```

Flag to indicate if the log-file can be used.

7.586.3.2 handle

```
FIL handle
```

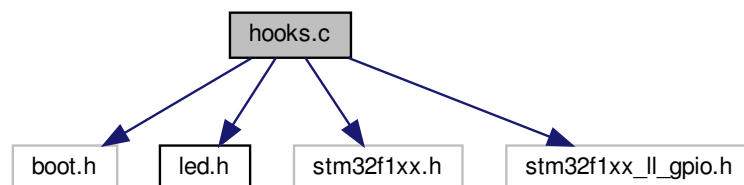
FatFS handle to the log-file.

7.587 hooks.c File Reference

Bootloader callback source file.

```
#include "boot.h"
#include "led.h"
#include "stm32f1xx.h"
#include "stm32f1xx_ll_gpio.h"
```

Include dependency graph for ARMC32F1_Olimexino_STM32_IAR/Boot/hooks.c:



Functions

- void [UsbConnectHook](#) ([blt_bool](#) connect)
Callback that gets called whenever the USB device should be connected to the USB bus.
- void [UsbEnterLowPowerModeHook](#) (void)
Callback that gets called whenever the USB host requests the device to enter a low power mode.
- void [UsbLeaveLowPowerModeHook](#) (void)
Callback that gets called whenever the USB host requests the device to exit low power mode.
- void [BackDoorInitHook](#) (void)
Initializes the backdoor entry option.
- [blt_bool](#) [BackDoorEntryHook](#) (void)
Checks if a backdoor entry is requested.
- [blt_bool](#) [CpuUserProgramStartHook](#) (void)
Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.
- void [CopInitHook](#) (void)
Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.
- void [CopServiceHook](#) (void)
Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.
- void [NvmInitHook](#) (void)
Callback that gets called at the start of the internal NVM driver initialization routine.
- void [NvmReinitHook](#) (void)
Callback that gets called at the start of a firmware update to reinitialize the NVM driver.
- [blt_int8u](#) [NvmWriteHook](#) ([blt_addr](#) addr, [blt_int32u](#) len, [blt_int8u](#) *data)
Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the data hasn't been written yet.

- [blt_int8u NvmEraseHook](#) ([blt_addr](#) addr, [blt_int32u](#) len)
Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then `BLT_NVM_NOT_IN_RANGE` must be returned to indicate that the memory hasn't been erased yet.
- [blt_bool NvmDoneHook](#) (void)
Callback that gets called at the end of the NVM programming session.
- [blt_bool FileIsFirmwareUpdateRequestedHook](#) (void)
Callback that gets called to check whether a firmware update from local file storage should be started. This could for example be when a switch is pressed, when a certain file is found on the local file storage, etc.
- const [blt_char](#) * [FileGetFirmwareFilenameHook](#) (void)
Callback to obtain the filename of the firmware file that should be used during the firmware update from the local file storage. This hook function is called at the beginning of the firmware update from local storage sequence.
- void [FileFirmwareUpdateStartedHook](#) (void)
Callback that gets called to inform the application that a firmware update from local storage just started.
- void [FileFirmwareUpdateCompletedHook](#) (void)
Callback that gets called to inform the application that a firmware update was successfully completed.
- void [FileFirmwareUpdateErrorHook](#) ([blt_int8u](#) error_code)
Callback that gets called in case an error occurred during a firmware update. Refer to [<file.h>](#) for a list of available error codes.
- void [FileFirmwareUpdateLogHook](#) ([blt_char](#) *info_string)
Callback that gets called each time new log information becomes available during a firmware update.
- [blt_int8u XcpGetSeedHook](#) ([blt_int8u](#) resource, [blt_int8u](#) *seed)
Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the `GET_SEED` command.
- [blt_int8u XcpVerifyKeyHook](#) ([blt_int8u](#) resource, [blt_int8u](#) *key, [blt_int8u](#) len)
Called by the `UNLOCK` command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

Variables

- static const [blt_char](#) [firmwareFilename](#) [] = "/demoprogram_olimexino_stm32.srec"
Firmware filename.
- ```

struct {
 FIL handle
 blt_bool canUse
} logfile

```

*Data structure for grouping log-file related information.*

### 7.587.1 Detailed Description

Bootloader callback source file.

### 7.587.2 Function Documentation

### 7.587.2.1 BackDoorEntryHook()

```
blt_bool BackDoorEntryHook (
 void)
```

Checks if a backdoor entry is requested.

#### Returns

BLT\_TRUE if the backdoor entry is requested, BLT\_FALSE otherwise.

### 7.587.2.2 BackDoorInitHook()

```
void BackDoorInitHook (
 void)
```

Initializes the backdoor entry option.

#### Returns

none.

### 7.587.2.3 CopInitHook()

```
void CopInitHook (
 void)
```

Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.

#### Returns

none.

### 7.587.2.4 CopServiceHook()

```
void CopServiceHook (
 void)
```

Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.

#### Returns

none.

#### 7.587.2.5 CpuUserProgramStartHook()

```
blt_bool CpuUserProgramStartHook (
 void)
```

Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.

##### Returns

BLT\_TRUE if it is okay to start the user program, BLT\_FALSE to keep keep the bootloader active.

#### 7.587.2.6 FileFirmwareUpdateCompletedHook()

```
void FileFirmwareUpdateCompletedHook (
 void)
```

Callback that gets called to inform the application that a firmware update was successfully completed.

##### Returns

none.

#### 7.587.2.7 FileFirmwareUpdateErrorHook()

```
void FileFirmwareUpdateErrorHook (
 blt_int8u error_code)
```

Callback that gets called in case an error occurred during a firmware update. Refer to <[file.h](#)> for a list of available error codes.

##### Returns

none.

#### 7.587.2.8 FileFirmwareUpdateLogHook()

```
void FileFirmwareUpdateLogHook (
 blt_char * info_string)
```

Callback that gets called each time new log information becomes available during a firmware update.

**Parameters**

|                    |                                                       |
|--------------------|-------------------------------------------------------|
| <i>info_string</i> | Pointer to a character array with the log entry info. |
|--------------------|-------------------------------------------------------|

**Returns**

none.

**7.587.2.9 FileFirmwareUpdateStartedHook()**

```
void FileFirmwareUpdateStartedHook (
 void)
```

Callback that gets called to inform the application that a firmware update from local storage just started.

**Returns**

none.

**7.587.2.10 FileGetFirmwareFilenameHook()**

```
const blt_char* FileGetFirmwareFilenameHook (
 void)
```

Callback to obtain the filename of the firmware file that should be used during the firmware update from the local file storage. This hook function is called at the beginning of the firmware update from local storage sequence.

**Returns**

valid firmware filename with full path or BLT\_NULL.

**7.587.2.11 FileIsFirmwareUpdateRequestedHook()**

```
blt_bool FileIsFirmwareUpdateRequestedHook (
 void)
```

Callback that gets called to check whether a firmware update from local file storage should be started. This could for example be when a switch is pressed, when a certain file is found on the local file storage, etc.

**Returns**

BLT\_TRUE if a firmware update is requested, BLT\_FALSE otherwise.

#### 7.587.2.12 NvmDoneHook()

```
blt_bool NvmDoneHook (
 void)
```

Callback that gets called at the end of the NVM programming session.

##### Returns

BLT\_TRUE is successful, BLT\_FALSE otherwise.

#### 7.587.2.13 NvmEraseHook()

```
blt_int8u NvmEraseHook (
 blt_addr addr,
 blt_int32u len)
```

Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the memory hasn't been erased yet.

##### Parameters

|             |                  |
|-------------|------------------|
| <i>addr</i> | Start address.   |
| <i>len</i>  | Length in bytes. |

##### Returns

BLT\_NVM\_OKAY if successful, BLT\_NVM\_NOT\_IN\_RANGE if the address is not within the supported memory range, or BLT\_NVM\_ERROR if the erase operation failed.

#### 7.587.2.14 NvmInitHook()

```
void NvmInitHook (
 void)
```

Callback that gets called at the start of the internal NVM driver initialization routine.

##### Returns

none.



### 7.587.2.15 NvmReinitHook()

```
void NvmReinitHook (
 void)
```

Callback that gets called at the start of a firmware update to reinitialize the NVM driver.

#### Returns

none.

### 7.587.2.16 NvmWriteHook()

```
blt_int8u NvmWriteHook (
 blt_addr addr,
 blt_int32u len,
 blt_int8u * data)
```

Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the data hasn't been written yet.

#### Parameters

|             |                             |
|-------------|-----------------------------|
| <i>addr</i> | Start address.              |
| <i>len</i>  | Length in bytes.            |
| <i>data</i> | Pointer to the data buffer. |

#### Returns

BLT\_NVM\_OKAY if successful, BLT\_NVM\_NOT\_IN\_RANGE if the address is not within the supported memory range, or BLT\_NVM\_ERROR if the write operation failed.

### 7.587.2.17 UsbConnectHook()

```
void UsbConnectHook (
 blt_bool connect)
```

Callback that gets called whenever the USB device should be connected to the USB bus.

#### Parameters

|                |                                                  |
|----------------|--------------------------------------------------|
| <i>connect</i> | BLT_TRUE to connect and BLT_FALSE to disconnect. |
|----------------|--------------------------------------------------|

**Returns**

none.

**7.587.2.18 UsbEnterLowPowerModeHook()**

```
void UsbEnterLowPowerModeHook (
 void)
```

Callback that gets called whenever the USB host requests the device to enter a low power mode.

**Returns**

none.

**7.587.2.19 UsbLeaveLowPowerModeHook()**

```
void UsbLeaveLowPowerModeHook (
 void)
```

Callback that gets called whenever the USB host requests the device to exit low power mode.

**Returns**

none.

**7.587.2.20 XcpGetSeedHook()**

```
blt_int8u XcpGetSeedHook (
 blt_int8u resource,
 blt_int8u * seed)
```

Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the GET\_SEED command.

**Parameters**

|                 |                                                        |
|-----------------|--------------------------------------------------------|
| <i>resource</i> | Resource that the seed is requested for (XCP_RES_XXX). |
| <i>seed</i>     | Pointer to byte buffer where the seed will be stored.  |

**Returns**

Length of the seed in bytes.

**7.587.2.21 XcpVerifyKeyHook()**

```
blt_int8u XcpVerifyKeyHook (
 blt_int8u resource,
 blt_int8u * key,
 blt_int8u len)
```

Called by the UNLOCK command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

**Parameters**

|                 |                                             |
|-----------------|---------------------------------------------|
| <i>resource</i> | resource to unlock (XCP_RES_XXX).           |
| <i>key</i>      | pointer to the byte buffer holding the key. |
| <i>len</i>      | length of the key in bytes.                 |

**Returns**

1 if the key was correct, 0 otherwise.

**7.587.3 Variable Documentation****7.587.3.1 canUse**

```
blt_bool canUse
```

Flag to indicate if the log-file can be used.

**7.587.3.2 handle**

```
FIL handle
```

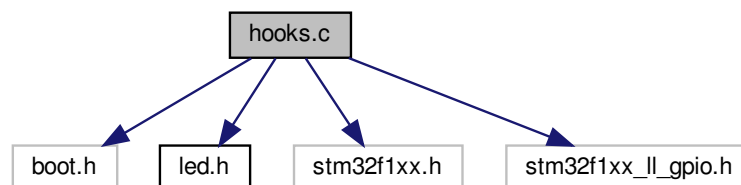
FatFS handle to the log-file.

## 7.588 hooks.c File Reference

Bootloader callback source file.

```
#include "boot.h"
#include "led.h"
#include "stm32f1xx.h"
#include "stm32f1xx_ll_gpio.h"
```

Include dependency graph for ARMC32F1\_Olimexino\_STM32\_Keil/Boot/hooks.c:



### Functions

- void [UsbConnectHook](#) ([blt\\_bool](#) connect)
 

*Callback that gets called whenever the USB device should be connected to the USB bus.*
- void [UsbEnterLowPowerModeHook](#) (void)
 

*Callback that gets called whenever the USB host requests the device to enter a low power mode.*
- void [UsbLeaveLowPowerModeHook](#) (void)
 

*Callback that gets called whenever the USB host requests the device to exit low power mode.*
- void [BackDoorInitHook](#) (void)
 

*Initializes the backdoor entry option.*
- [blt\\_bool](#) [BackDoorEntryHook](#) (void)
 

*Checks if a backdoor entry is requested.*
- [blt\\_bool](#) [CpuUserProgramStartHook](#) (void)
 

*Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.*
- void [CopInitHook](#) (void)
 

*Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.*
- void [CopServiceHook](#) (void)
 

*Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.*
- void [NvmInitHook](#) (void)
 

*Callback that gets called at the start of the internal NVM driver initialization routine.*
- void [NvmReinitHook](#) (void)
 

*Callback that gets called at the start of a firmware update to reinitialize the NVM driver.*
- [blt\\_int8u](#) [NvmWriteHook](#) ([blt\\_addr](#) addr, [blt\\_int32u](#) len, [blt\\_int8u](#) \*data)
 

*Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the data hasn't been written yet.*

- [blt\\_int8u NvmEraseHook](#) ([blt\\_addr](#) addr, [blt\\_int32u](#) len)  
*Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then `BLT_NVM_NOT_IN_RANGE` must be returned to indicate that the memory hasn't been erased yet.*
- [blt\\_bool NvmDoneHook](#) (void)  
*Callback that gets called at the end of the NVM programming session.*
- [blt\\_bool FileIsFirmwareUpdateRequestedHook](#) (void)  
*Callback that gets called to check whether a firmware update from local file storage should be started. This could for example be when a switch is pressed, when a certain file is found on the local file storage, etc.*
- const [blt\\_char](#) \* [FileGetFirmwareFilenameHook](#) (void)  
*Callback to obtain the filename of the firmware file that should be used during the firmware update from the local file storage. This hook function is called at the beginning of the firmware update from local storage sequence.*
- void [FileFirmwareUpdateStartedHook](#) (void)  
*Callback that gets called to inform the application that a firmware update from local storage just started.*
- void [FileFirmwareUpdateCompletedHook](#) (void)  
*Callback that gets called to inform the application that a firmware update was successfully completed.*
- void [FileFirmwareUpdateErrorHook](#) ([blt\\_int8u](#) error\_code)  
*Callback that gets called in case an error occurred during a firmware update. Refer to [<file.h>](#) for a list of available error codes.*
- void [FileFirmwareUpdateLogHook](#) ([blt\\_char](#) \*info\_string)  
*Callback that gets called each time new log information becomes available during a firmware update.*
- [blt\\_int8u XcpGetSeedHook](#) ([blt\\_int8u](#) resource, [blt\\_int8u](#) \*seed)  
*Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the `GET_SEED` command.*
- [blt\\_int8u XcpVerifyKeyHook](#) ([blt\\_int8u](#) resource, [blt\\_int8u](#) \*key, [blt\\_int8u](#) len)  
*Called by the `UNLOCK` command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.*

## Variables

- static const [blt\\_char](#) [firmwareFilename](#) [] = "/demoprogram\_olimexino\_stm32.srec"  
*Firmware filename.*
- ```

struct {
    FIL handle
    blt\_bool canUse
} logfile

```

Data structure for grouping log-file related information.

7.588.1 Detailed Description

Bootloader callback source file.

7.588.2 Function Documentation

7.588.2.1 BackDoorEntryHook()

```
blt_bool BackDoorEntryHook (  
    void )
```

Checks if a backdoor entry is requested.

Returns

BLT_TRUE if the backdoor entry is requested, BLT_FALSE otherwise.

7.588.2.2 BackDoorInitHook()

```
void BackDoorInitHook (  
    void )
```

Initializes the backdoor entry option.

Returns

none.

7.588.2.3 CopInitHook()

```
void CopInitHook (  
    void )
```

Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.

Returns

none.

7.588.2.4 CopServiceHook()

```
void CopServiceHook (  
    void )
```

Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.

Returns

none.

7.588.2.5 CpuUserProgramStartHook()

```
blt_bool CpuUserProgramStartHook (
    void )
```

Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.

Returns

BLT_TRUE if it is okay to start the user program, BLT_FALSE to keep keep the bootloader active.

7.588.2.6 FileFirmwareUpdateCompletedHook()

```
void FileFirmwareUpdateCompletedHook (
    void )
```

Callback that gets called to inform the application that a firmware update was successfully completed.

Returns

none.

7.588.2.7 FileFirmwareUpdateErrorHook()

```
void FileFirmwareUpdateErrorHook (
    blt_int8u error_code )
```

Callback that gets called in case an error occurred during a firmware update. Refer to <[file.h](#)> for a list of available error codes.

Returns

none.

7.588.2.8 FileFirmwareUpdateLogHook()

```
void FileFirmwareUpdateLogHook (
    blt_char * info_string )
```

Callback that gets called each time new log information becomes available during a firmware update.

Parameters

<i>info_string</i>	Pointer to a character array with the log entry info.
--------------------	---

Returns

none.

7.588.2.9 FileFirmwareUpdateStartedHook()

```
void FileFirmwareUpdateStartedHook (  
    void )
```

Callback that gets called to inform the application that a firmware update from local storage just started.

Returns

none.

7.588.2.10 FileGetFirmwareFilenameHook()

```
const blt_char* FileGetFirmwareFilenameHook (  
    void )
```

Callback to obtain the filename of the firmware file that should be used during the firmware update from the local file storage. This hook function is called at the beginning of the firmware update from local storage sequence.

Returns

valid firmware filename with full path or BLT_NULL.

7.588.2.11 FileIsFirmwareUpdateRequestedHook()

```
blt_bool FileIsFirmwareUpdateRequestedHook (  
    void )
```

Callback that gets called to check whether a firmware update from local file storage should be started. This could for example be when a switch is pressed, when a certain file is found on the local file storage, etc.

Returns

BLT_TRUE if a firmware update is requested, BLT_FALSE otherwise.

7.588.2.12 NvmDoneHook()

```
blt_bool NvmDoneHook (
    void )
```

Callback that gets called at the end of the NVM programming session.

Returns

BLT_TRUE is successful, BLT_FALSE otherwise.

7.588.2.13 NvmEraseHook()

```
blt_int8u NvmEraseHook (
    blt_addr addr,
    blt_int32u len )
```

Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the memory hasn't been erased yet.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.

Returns

BLT_NVM_OKAY if successful, BLT_NVM_NOT_IN_RANGE if the address is not within the supported memory range, or BLT_NVM_ERROR if the erase operation failed.

7.588.2.14 NvmInitHook()

```
void NvmInitHook (
    void )
```

Callback that gets called at the start of the internal NVM driver initialization routine.

Returns

none.

7.588.2.15 NvmReinitHook()

```
void NvmReinitHook (  
    void )
```

Callback that gets called at the start of a firmware update to reinitialize the NVM driver.

Returns

none.

7.588.2.16 NvmWriteHook()

```
blt_int8u NvmWriteHook (  
    blt_addr addr,  
    blt_int32u len,  
    blt_int8u * data )
```

Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the data hasn't been written yet.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.
<i>data</i>	Pointer to the data buffer.

Returns

BLT_NVM_OKAY if successful, BLT_NVM_NOT_IN_RANGE if the address is not within the supported memory range, or BLT_NVM_ERROR if the write operation failed.

7.588.2.17 UsbConnectHook()

```
void UsbConnectHook (  
    blt_bool connect )
```

Callback that gets called whenever the USB device should be connected to the USB bus.

Parameters

<i>connect</i>	BLT_TRUE to connect and BLT_FALSE to disconnect.
----------------	--

Returns

none.

7.588.2.18 UsbEnterLowPowerModeHook()

```
void UsbEnterLowPowerModeHook (  
    void )
```

Callback that gets called whenever the USB host requests the device to enter a low power mode.

Returns

none.

7.588.2.19 UsbLeaveLowPowerModeHook()

```
void UsbLeaveLowPowerModeHook (  
    void )
```

Callback that gets called whenever the USB host requests the device to exit low power mode.

Returns

none.

7.588.2.20 XcpGetSeedHook()

```
blt_int8u XcpGetSeedHook (  
    blt_int8u resource,  
    blt_int8u * seed )
```

Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the GET_SEED command.

Parameters

<i>resource</i>	Resource that the seed is requested for (XCP_RES_XXX).
<i>seed</i>	Pointer to byte buffer where the seed will be stored.

Returns

Length of the seed in bytes.

7.588.2.21 XcpVerifyKeyHook()

```
blt_int8u XcpVerifyKeyHook (
    blt_int8u resource,
    blt_int8u * key,
    blt_int8u len )
```

Called by the UNLOCK command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

Parameters

<i>resource</i>	resource to unlock (XCP_RES_XXX).
<i>key</i>	pointer to the byte buffer holding the key.
<i>len</i>	length of the key in bytes.

Returns

1 if the key was correct, 0 otherwise.

7.588.3 Variable Documentation**7.588.3.1 canUse**

```
blt_bool canUse
```

Flag to indicate if the log-file can be used.

7.588.3.2 handle

```
FIL handle
```

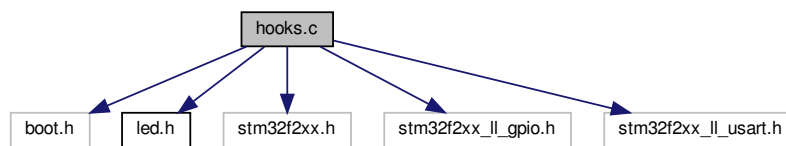
FatFS handle to the log-file.

7.589 hooks.c File Reference

Bootloader callback source file.

```
#include "boot.h"
#include "led.h"
#include "stm32f2xx.h"
#include "stm32f2xx_ll_gpio.h"
#include "stm32f2xx_ll_usart.h"
```

Include dependency graph for ARMCM3_STM32F2_Olimex_STM32P207_CubeIDE/Boot/App/hooks.c:



Functions

- void [BackDoorInitHook](#) (void)
Initializes the backdoor entry option.
- [blt_bool](#) [BackDoorEntryHook](#) (void)
Checks if a backdoor entry is requested.
- [blt_bool](#) [CpuUserProgramStartHook](#) (void)
Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.
- void [CopInitHook](#) (void)
Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.
- void [CopServiceHook](#) (void)
Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.
- void [NvmInitHook](#) (void)
Callback that gets called at the start of the internal NVM driver initialization routine.
- void [NvmReinitHook](#) (void)
Callback that gets called at the start of a firmware update to reinitialize the NVM driver.
- [blt_int8u](#) [NvmWriteHook](#) ([blt_addr](#) addr, [blt_int32u](#) len, [blt_int8u](#) *data)
Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the data hasn't been written yet.
- [blt_int8u](#) [NvmEraseHook](#) ([blt_addr](#) addr, [blt_int32u](#) len)
Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the memory hasn't been erased yet.
- [blt_bool](#) [NvmDoneHook](#) (void)
Callback that gets called at the end of the NVM programming session.
- [blt_bool](#) [FileIsFirmwareUpdateRequestedHook](#) (void)

Callback that gets called to check whether a firmware update from local file storage should be started. This could for example be when a switch is pressed, when a certain file is found on the local file storage, etc.

- const `blt_char * FileGetFirmwareFilenameHook` (void)

Callback to obtain the filename of the firmware file that should be used during the firmware update from the local file storage. This hook function is called at the beginning of the firmware update from local storage sequence.

- void `FileFirmwareUpdateStartedHook` (void)

Callback that gets called to inform the application that a firmware update from local storage just started.

- void `FileFirmwareUpdateCompletedHook` (void)

Callback that gets called to inform the application that a firmware update was successfully completed.

- void `FileFirmwareUpdateErrorHook` (blt_int8u error_code)

Callback that gets called in case an error occurred during a firmware update. Refer to [<file.h>](#) for a list of available error codes.

- void `FileFirmwareUpdateLogHook` (blt_char *info_string)

Callback that gets called each time new log information becomes available during a firmware update.

- blt_int8u `XcpGetSeedHook` (blt_int8u resource, blt_int8u *seed)

Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the `GET_SEED` command.

- blt_int8u `XcpVerifyKeyHook` (blt_int8u resource, blt_int8u *key, blt_int8u len)

Called by the `UNLOCK` command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

Variables

- static const `blt_char firmwareFilename` [] = `"/demoprogram_stm32f207.srec"`

Firmware filename.

-

```
struct {
    FIL handle
    blt_bool canUse
} logfile
```

Data structure for grouping log-file related information.

7.589.1 Detailed Description

Bootloader callback source file.

7.589.2 Function Documentation

7.589.2.1 BackDoorEntryHook()

```
blt_bool BackDoorEntryHook (
    void )
```

Checks if a backdoor entry is requested.

Returns

BLT_TRUE if the backdoor entry is requested, BLT_FALSE otherwise.

7.589.2.2 BackDoorInitHook()

```
void BackDoorInitHook (  
    void )
```

Initializes the backdoor entry option.

Returns

none.

7.589.2.3 CopInitHook()

```
void CopInitHook (  
    void )
```

Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.

Returns

none.

7.589.2.4 CopServiceHook()

```
void CopServiceHook (  
    void )
```

Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.

Returns

none.

7.589.2.5 CpuUserProgramStartHook()

```
blt_bool CpuUserProgramStartHook (  
    void )
```

Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.

Returns

BLT_TRUE if it is okay to start the user program, BLT_FALSE to keep keep the bootloader active.

7.589.2.6 FileFirmwareUpdateCompletedHook()

```
void FileFirmwareUpdateCompletedHook (
    void )
```

Callback that gets called to inform the application that a firmware update was successfully completed.

Returns

none.

7.589.2.7 FileFirmwareUpdateErrorHook()

```
void FileFirmwareUpdateErrorHook (
    blt_int8u error_code )
```

Callback that gets called in case an error occurred during a firmware update. Refer to <[file.h](#)> for a list of available error codes.

Returns

none.

7.589.2.8 FileFirmwareUpdateLogHook()

```
void FileFirmwareUpdateLogHook (
    blt_char * info_string )
```

Callback that gets called each time new log information becomes available during a firmware update.

Parameters

<i>info_string</i>	Pointer to a character array with the log entry info.
--------------------	---

Returns

none.

7.589.2.9 FileFirmwareUpdateStartedHook()

```
void FileFirmwareUpdateStartedHook (
    void )
```

Callback that gets called to inform the application that a firmware update from local storage just started.

Returns

none.

7.589.2.10 FileGetFirmwareFilenameHook()

```
const blt_char* FileGetFirmwareFilenameHook (  
    void )
```

Callback to obtain the filename of the firmware file that should be used during the firmware update from the local file storage. This hook function is called at the beginning of the firmware update from local storage sequence.

Returns

valid firmware filename with full path or BLT_NULL.

7.589.2.11 FileIsFirmwareUpdateRequestedHook()

```
blt_bool FileIsFirmwareUpdateRequestedHook (  
    void )
```

Callback that gets called to check whether a firmware update from local file storage should be started. This could for example be when a switch is pressed, when a certain file is found on the local file storage, etc.

Returns

BLT_TRUE if a firmware update is requested, BLT_FALSE otherwise.

7.589.2.12 NvmDoneHook()

```
blt_bool NvmDoneHook (  
    void )
```

Callback that gets called at the end of the NVM programming session.

Returns

BLT_TRUE is successful, BLT_FALSE otherwise.

7.589.2.13 NvmEraseHook()

```
blt_int8u NvmEraseHook (  
    blt_addr addr,  
    blt_int32u len )
```

Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the memory hasn't been erased yet.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.

Returns

BLT_NVM_OKAY if successful, BLT_NVM_NOT_IN_RANGE if the address is not within the supported memory range, or BLT_NVM_ERROR if the erase operation failed.

7.589.2.14 NvmInitHook()

```
void NvmInitHook (  
    void )
```

Callback that gets called at the start of the internal NVM driver initialization routine.

Returns

none.

7.589.2.15 NvmReinitHook()

```
void NvmReinitHook (  
    void )
```

Callback that gets called at the start of a firmware update to reinitialize the NVM driver.

Returns

none.

7.589.2.16 NvmWriteHook()

```
blt_int8u NvmWriteHook (  
    blt_addr addr,  
    blt_int32u len,  
    blt_int8u * data )
```

Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the data hasn't been written yet.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.
<i>data</i>	Pointer to the data buffer.

Returns

BLT_NVM_OKAY if successful, BLT_NVM_NOT_IN_RANGE if the address is not within the supported memory range, or BLT_NVM_ERROR if the write operation failed.

7.589.2.17 XcpGetSeedHook()

```
blt_int8u XcpGetSeedHook (
    blt_int8u resource,
    blt_int8u * seed )
```

Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the GET_SEED command.

Parameters

<i>resource</i>	Resource that the seed is requested for (XCP_RES_XXX).
<i>seed</i>	Pointer to byte buffer where the seed will be stored.

Returns

Length of the seed in bytes.

7.589.2.18 XcpVerifyKeyHook()

```
blt_int8u XcpVerifyKeyHook (
    blt_int8u resource,
    blt_int8u * key,
    blt_int8u len )
```

Called by the UNLOCK command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

Parameters

<i>resource</i>	resource to unlock (XCP_RES_XXX).
<i>key</i>	pointer to the byte buffer holding the key.
<i>len</i>	length of the key in bytes.

Returns

1 if the key was correct, 0 otherwise.

7.589.3 Variable Documentation**7.589.3.1 canUse**

`blt_bool` canUse

Flag to indicate if the log-file can be used.

7.589.3.2 handle

`FIL` handle

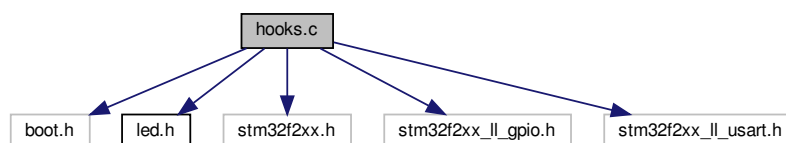
FatFS handle to the log-file.

7.590 hooks.c File Reference

Bootloader callback source file.

```
#include "boot.h"
#include "led.h"
#include "stm32f2xx.h"
#include "stm32f2xx_ll_gpio.h"
#include "stm32f2xx_ll_usart.h"
```

Include dependency graph for ARMC3_STM32F2_Olimex_STM32P207_GCC/Boot/hooks.c:



Functions

- void [BackDoorInitHook](#) (void)
Initializes the backdoor entry option.
- [blt_bool BackDoorEntryHook](#) (void)
Checks if a backdoor entry is requested.
- [blt_bool CpuUserProgramStartHook](#) (void)
Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.
- void [CopInitHook](#) (void)
Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.
- void [CopServiceHook](#) (void)
Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.
- void [NvmInitHook](#) (void)
Callback that gets called at the start of the internal NVM driver initialization routine.
- void [NvmReinitHook](#) (void)
Callback that gets called at the start of a firmware update to reinitialize the NVM driver.
- [blt_int8u NvmWriteHook](#) ([blt_addr](#) addr, [blt_int32u](#) len, [blt_int8u](#) *data)
Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the data hasn't been written yet.
- [blt_int8u NvmEraseHook](#) ([blt_addr](#) addr, [blt_int32u](#) len)
Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the memory hasn't been erased yet.
- [blt_bool NvmDoneHook](#) (void)
Callback that gets called at the end of the NVM programming session.
- [blt_bool FileIsFirmwareUpdateRequestedHook](#) (void)
Callback that gets called to check whether a firmware update from local file storage should be started. This could for example be when a switch is pressed, when a certain file is found on the local file storage, etc.
- const [blt_char](#) * [FileGetFirmwareFilenameHook](#) (void)
Callback to obtain the filename of the firmware file that should be used during the firmware update from the local file storage. This hook function is called at the beginning of the firmware update from local storage sequence.
- void [FileFirmwareUpdateStartedHook](#) (void)
Callback that gets called to inform the application that a firmware update from local storage just started.
- void [FileFirmwareUpdateCompletedHook](#) (void)
Callback that gets called to inform the application that a firmware update was successfully completed.
- void [FileFirmwareUpdateErrorHook](#) ([blt_int8u](#) error_code)
Callback that gets called in case an error occurred during a firmware update. Refer to <file.h> for a list of available error codes.
- void [FileFirmwareUpdateLogHook](#) ([blt_char](#) *info_string)
Callback that gets called each time new log information becomes available during a firmware update.
- [blt_int8u XcpGetSeedHook](#) ([blt_int8u](#) resource, [blt_int8u](#) *seed)
Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the GET_SEED command.
- [blt_int8u XcpVerifyKeyHook](#) ([blt_int8u](#) resource, [blt_int8u](#) *key, [blt_int8u](#) len)
Called by the UNLOCK command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

Variables

- static const `blt_char firmwareFilename []` = `"/demoprogram_stm32f207.srec"`
Firmware filename.
- ```
struct {
 FIL handle
 blt_bool canUse
} logfile
```

*Data structure for grouping log-file related information.*

### 7.590.1 Detailed Description

Bootloader callback source file.

### 7.590.2 Function Documentation

#### 7.590.2.1 BackDoorEntryHook()

```
blt_bool BackDoorEntryHook (
 void)
```

Checks if a backdoor entry is requested.

#### Returns

BLT\_TRUE if the backdoor entry is requested, BLT\_FALSE otherwise.

#### 7.590.2.2 BackDoorInitHook()

```
void BackDoorInitHook (
 void)
```

Initializes the backdoor entry option.

#### Returns

none.

### 7.590.2.3 CopInitHook()

```
void CopInitHook (
 void)
```

Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.

#### Returns

none.

### 7.590.2.4 CopServiceHook()

```
void CopServiceHook (
 void)
```

Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.

#### Returns

none.

### 7.590.2.5 CpuUserProgramStartHook()

```
blt_bool CpuUserProgramStartHook (
 void)
```

Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.

#### Returns

BLT\_TRUE if it is okay to start the user program, BLT\_FALSE to keep keep the bootloader active.

### 7.590.2.6 FileFirmwareUpdateCompletedHook()

```
void FileFirmwareUpdateCompletedHook (
 void)
```

Callback that gets called to inform the application that a firmware update was successfully completed.

#### Returns

none.

#### 7.590.2.7 FileFirmwareUpdateErrorHook()

```
void FileFirmwareUpdateErrorHook (
 blt_int8u error_code)
```

Callback that gets called in case an error occurred during a firmware update. Refer to <[file.h](#)> for a list of available error codes.

##### Returns

none.

#### 7.590.2.8 FileFirmwareUpdateLogHook()

```
void FileFirmwareUpdateLogHook (
 blt_char * info_string)
```

Callback that gets called each time new log information becomes available during a firmware update.

##### Parameters

|                    |                                                       |
|--------------------|-------------------------------------------------------|
| <i>info_string</i> | Pointer to a character array with the log entry info. |
|--------------------|-------------------------------------------------------|

##### Returns

none.

#### 7.590.2.9 FileFirmwareUpdateStartedHook()

```
void FileFirmwareUpdateStartedHook (
 void)
```

Callback that gets called to inform the application that a firmware update from local storage just started.

##### Returns

none.

#### 7.590.2.10 FileGetFirmwareFilenameHook()

```
const blt_char* FileGetFirmwareFilenameHook (
 void)
```

Callback to obtain the filename of the firmware file that should be used during the firmware update from the local file storage. This hook function is called at the beginning of the firmware update from local storage sequence.

##### Returns

valid firmware filename with full path or BLT\_NULL.



#### 7.590.2.11 FileIsFirmwareUpdateRequestedHook()

```
blt_bool FileIsFirmwareUpdateRequestedHook (
 void)
```

Callback that gets called to check whether a firmware update from local file storage should be started. This could for example be when a switch is pressed, when a certain file is found on the local file storage, etc.

##### Returns

BLT\_TRUE if a firmware update is requested, BLT\_FALSE otherwise.

#### 7.590.2.12 NvmDoneHook()

```
blt_bool NvmDoneHook (
 void)
```

Callback that gets called at the end of the NVM programming session.

##### Returns

BLT\_TRUE is successful, BLT\_FALSE otherwise.

#### 7.590.2.13 NvmEraseHook()

```
blt_int8u NvmEraseHook (
 blt_addr addr,
 blt_int32u len)
```

Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the memory hasn't been erased yet.

##### Parameters

|             |                  |
|-------------|------------------|
| <i>addr</i> | Start address.   |
| <i>len</i>  | Length in bytes. |

##### Returns

BLT\_NVM\_OKAY if successful, BLT\_NVM\_NOT\_IN\_RANGE if the address is not within the supported memory range, or BLT\_NVM\_ERROR if the erase operation failed.

#### 7.590.2.14 NvmInitHook()

```
void NvmInitHook (
 void)
```

Callback that gets called at the start of the internal NVM driver initialization routine.

##### Returns

none.

#### 7.590.2.15 NvmReinitHook()

```
void NvmReinitHook (
 void)
```

Callback that gets called at the start of a firmware update to reinitialize the NVM driver.

##### Returns

none.

#### 7.590.2.16 NvmWriteHook()

```
blt_int8u NvmWriteHook (
 blt_addr addr,
 blt_int32u len,
 blt_int8u * data)
```

Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the data hasn't been written yet.

##### Parameters

|             |                             |
|-------------|-----------------------------|
| <i>addr</i> | Start address.              |
| <i>len</i>  | Length in bytes.            |
| <i>data</i> | Pointer to the data buffer. |

##### Returns

BLT\_NVM\_OKAY if successful, BLT\_NVM\_NOT\_IN\_RANGE if the address is not within the supported memory range, or BLT\_NVM\_ERROR if the write operation failed.

### 7.590.2.17 XcpGetSeedHook()

```
blt_int8u XcpGetSeedHook (
 blt_int8u resource,
 blt_int8u * seed)
```

Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the GET\_SEED command.

#### Parameters

|                 |                                                        |
|-----------------|--------------------------------------------------------|
| <i>resource</i> | Resource that the seed is requested for (XCP_RES_XXX). |
| <i>seed</i>     | Pointer to byte buffer where the seed will be stored.  |

#### Returns

Length of the seed in bytes.

### 7.590.2.18 XcpVerifyKeyHook()

```
blt_int8u XcpVerifyKeyHook (
 blt_int8u resource,
 blt_int8u * key,
 blt_int8u len)
```

Called by the UNLOCK command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

#### Parameters

|                 |                                             |
|-----------------|---------------------------------------------|
| <i>resource</i> | resource to unlock (XCP_RES_XXX).           |
| <i>key</i>      | pointer to the byte buffer holding the key. |
| <i>len</i>      | length of the key in bytes.                 |

#### Returns

1 if the key was correct, 0 otherwise.

## 7.590.3 Variable Documentation

### 7.590.3.1 canUse

```
blt_bool canUse
```

Flag to indicate if the log-file can be used.

### 7.590.3.2 handle

FIL handle

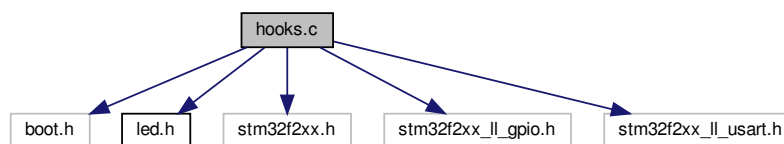
FatFS handle to the log-file.

## 7.591 hooks.c File Reference

Bootloader callback source file.

```
#include "boot.h"
#include "led.h"
#include "stm32f2xx.h"
#include "stm32f2xx_ll_gpio.h"
#include "stm32f2xx_ll_usart.h"
```

Include dependency graph for ARMCM3\_STM32F2\_Olimex\_STM32P207\_IAR/Boot/hooks.c:



## Functions

- void [BackDoorInitHook](#) (void)  
*Initializes the backdoor entry option.*
- [blt\\_bool](#) [BackDoorEntryHook](#) (void)  
*Checks if a backdoor entry is requested.*
- [blt\\_bool](#) [CpuUserProgramStartHook](#) (void)  
*Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.*
- void [CopInitHook](#) (void)  
*Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.*
- void [CopServiceHook](#) (void)  
*Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.*
- void [NvmInitHook](#) (void)  
*Callback that gets called at the start of the internal NVM driver initialization routine.*
- void [NvmReinitHook](#) (void)  
*Callback that gets called at the start of a firmware update to reinitialize the NVM driver.*
- [blt\\_int8u](#) [NvmWriteHook](#) ([blt\\_addr](#) addr, [blt\\_int32u](#) len, [blt\\_int8u](#) \*data)  
*Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the data hasn't been written yet.*

- [blt\\_int8u NvmEraseHook](#) ([blt\\_addr](#) addr, [blt\\_int32u](#) len)  
*Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then `BLT_NVM_NOT_IN_RANGE` must be returned to indicate that the memory hasn't been erased yet.*
- [blt\\_bool NvmDoneHook](#) (void)  
*Callback that gets called at the end of the NVM programming session.*
- [blt\\_bool FileIsFirmwareUpdateRequestedHook](#) (void)  
*Callback that gets called to check whether a firmware update from local file storage should be started. This could for example be when a switch is pressed, when a certain file is found on the local file storage, etc.*
- const [blt\\_char](#) \* [FileGetFirmwareFilenameHook](#) (void)  
*Callback to obtain the filename of the firmware file that should be used during the firmware update from the local file storage. This hook function is called at the beginning of the firmware update from local storage sequence.*
- void [FileFirmwareUpdateStartedHook](#) (void)  
*Callback that gets called to inform the application that a firmware update from local storage just started.*
- void [FileFirmwareUpdateCompletedHook](#) (void)  
*Callback that gets called to inform the application that a firmware update was successfully completed.*
- void [FileFirmwareUpdateErrorHook](#) ([blt\\_int8u](#) error\_code)  
*Callback that gets called in case an error occurred during a firmware update. Refer to [<file.h>](#) for a list of available error codes.*
- void [FileFirmwareUpdateLogHook](#) ([blt\\_char](#) \*info\_string)  
*Callback that gets called each time new log information becomes available during a firmware update.*
- [blt\\_int8u XcpGetSeedHook](#) ([blt\\_int8u](#) resource, [blt\\_int8u](#) \*seed)  
*Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the `GET_SEED` command.*
- [blt\\_int8u XcpVerifyKeyHook](#) ([blt\\_int8u](#) resource, [blt\\_int8u](#) \*key, [blt\\_int8u](#) len)  
*Called by the `UNLOCK` command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.*

## Variables

- static const [blt\\_char](#) [firmwareFilename](#) [] = `"/demoprogram_stm32f207.srec"`  
*Firmware filename.*
- ```

struct {
    FIL handle
    blt\_bool canUse
} logfile

```

Data structure for grouping log-file related information.

7.591.1 Detailed Description

Bootloader callback source file.

7.591.2 Function Documentation

7.591.2.1 BackDoorEntryHook()

```
blt_bool BackDoorEntryHook (  
    void )
```

Checks if a backdoor entry is requested.

Returns

BLT_TRUE if the backdoor entry is requested, BLT_FALSE otherwise.

7.591.2.2 BackDoorInitHook()

```
void BackDoorInitHook (  
    void )
```

Initializes the backdoor entry option.

Returns

none.

7.591.2.3 CopInitHook()

```
void CopInitHook (  
    void )
```

Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.

Returns

none.

7.591.2.4 CopServiceHook()

```
void CopServiceHook (  
    void )
```

Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.

Returns

none.

7.591.2.5 CpuUserProgramStartHook()

```
blt_bool CpuUserProgramStartHook (
    void )
```

Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.

Returns

BLT_TRUE if it is okay to start the user program, BLT_FALSE to keep keep the bootloader active.

7.591.2.6 FileFirmwareUpdateCompletedHook()

```
void FileFirmwareUpdateCompletedHook (
    void )
```

Callback that gets called to inform the application that a firmware update was successfully completed.

Returns

none.

7.591.2.7 FileFirmwareUpdateErrorHook()

```
void FileFirmwareUpdateErrorHook (
    blt_int8u error_code )
```

Callback that gets called in case an error occurred during a firmware update. Refer to <[file.h](#)> for a list of available error codes.

Returns

none.

7.591.2.8 FileFirmwareUpdateLogHook()

```
void FileFirmwareUpdateLogHook (
    blt_char * info_string )
```

Callback that gets called each time new log information becomes available during a firmware update.

Parameters

<i>info_string</i>	Pointer to a character array with the log entry info.
--------------------	---

Returns

none.

7.591.2.9 FileFirmwareUpdateStartedHook()

```
void FileFirmwareUpdateStartedHook (  
    void )
```

Callback that gets called to inform the application that a firmware update from local storage just started.

Returns

none.

7.591.2.10 FileGetFirmwareFilenameHook()

```
const blt_char* FileGetFirmwareFilenameHook (  
    void )
```

Callback to obtain the filename of the firmware file that should be used during the firmware update from the local file storage. This hook function is called at the beginning of the firmware update from local storage sequence.

Returns

valid firmware filename with full path or BLT_NULL.

7.591.2.11 FileIsFirmwareUpdateRequestedHook()

```
blt_bool FileIsFirmwareUpdateRequestedHook (  
    void )
```

Callback that gets called to check whether a firmware update from local file storage should be started. This could for example be when a switch is pressed, when a certain file is found on the local file storage, etc.

Returns

BLT_TRUE if a firmware update is requested, BLT_FALSE otherwise.

7.591.2.12 NvmDoneHook()

```
blt_bool NvmDoneHook (
    void )
```

Callback that gets called at the end of the NVM programming session.

Returns

BLT_TRUE is successful, BLT_FALSE otherwise.

7.591.2.13 NvmEraseHook()

```
blt_int8u NvmEraseHook (
    blt_addr addr,
    blt_int32u len )
```

Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the memory hasn't been erased yet.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.

Returns

BLT_NVM_OKAY if successful, BLT_NVM_NOT_IN_RANGE if the address is not within the supported memory range, or BLT_NVM_ERROR if the erase operation failed.

7.591.2.14 NvmInitHook()

```
void NvmInitHook (
    void )
```

Callback that gets called at the start of the internal NVM driver initialization routine.

Returns

none.

7.591.2.15 NvmReinitHook()

```
void NvmReinitHook (  
    void )
```

Callback that gets called at the start of a firmware update to reinitialize the NVM driver.

Returns

none.

7.591.2.16 NvmWriteHook()

```
blt_int8u NvmWriteHook (  
    blt_addr addr,  
    blt_int32u len,  
    blt_int8u * data )
```

Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the data hasn't been written yet.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.
<i>data</i>	Pointer to the data buffer.

Returns

BLT_NVM_OKAY if successful, BLT_NVM_NOT_IN_RANGE if the address is not within the supported memory range, or BLT_NVM_ERROR if the write operation failed.

7.591.2.17 XcpGetSeedHook()

```
blt_int8u XcpGetSeedHook (  
    blt_int8u resource,  
    blt_int8u * seed )
```

Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the GET_SEED command.

Parameters

<i>resource</i>	Resource that the seed is requested for (XCP_RES_XXX).
<i>seed</i>	Pointer to byte buffer where the seed will be stored.

Returns

Length of the seed in bytes.

7.591.2.18 XcpVerifyKeyHook()

```
blt_int8u XcpVerifyKeyHook (
    blt_int8u resource,
    blt_int8u * key,
    blt_int8u len )
```

Called by the UNLOCK command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

Parameters

<i>resource</i>	resource to unlock (XCP_RES_XXX).
<i>key</i>	pointer to the byte buffer holding the key.
<i>len</i>	length of the key in bytes.

Returns

1 if the key was correct, 0 otherwise.

7.591.3 Variable Documentation**7.591.3.1 canUse**

```
blt_bool canUse
```

Flag to indicate if the log-file can be used.

7.591.3.2 handle

```
FIL handle
```

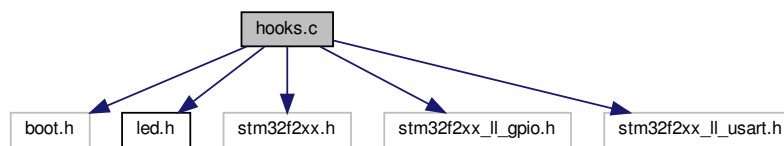
FatFS handle to the log-file.

7.592 hooks.c File Reference

Bootloader callback source file.

```
#include "boot.h"
#include "led.h"
#include "stm32f2xx.h"
#include "stm32f2xx_ll_gpio.h"
#include "stm32f2xx_ll_usart.h"
```

Include dependency graph for ARMCM3_STM32F2_Olimex_STM32P207_Keil/Boot/hooks.c:



Functions

- void [BackDoorInitHook](#) (void)
Initializes the backdoor entry option.
- [blt_bool](#) [BackDoorEntryHook](#) (void)
Checks if a backdoor entry is requested.
- [blt_bool](#) [CpuUserProgramStartHook](#) (void)
Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.
- void [CopInitHook](#) (void)
Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.
- void [CopServiceHook](#) (void)
Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.
- void [NvmInitHook](#) (void)
Callback that gets called at the start of the internal NVM driver initialization routine.
- void [NvmReinitHook](#) (void)
Callback that gets called at the start of a firmware update to reinitialize the NVM driver.
- [blt_int8u](#) [NvmWriteHook](#) ([blt_addr](#) addr, [blt_int32u](#) len, [blt_int8u](#) *data)
Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the data hasn't been written yet.
- [blt_int8u](#) [NvmEraseHook](#) ([blt_addr](#) addr, [blt_int32u](#) len)
Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the memory hasn't been erased yet.
- [blt_bool](#) [NvmDoneHook](#) (void)
Callback that gets called at the end of the NVM programming session.
- [blt_bool](#) [FileIsFirmwareUpdateRequestedHook](#) (void)

Callback that gets called to check whether a firmware update from local file storage should be started. This could for example be when a switch is pressed, when a certain file is found on the local file storage, etc.

- const `blt_char * FileGetFirmwareFilenameHook` (void)

Callback to obtain the filename of the firmware file that should be used during the firmware update from the local file storage. This hook function is called at the beginning of the firmware update from local storage sequence.

- void `FileFirmwareUpdateStartedHook` (void)

Callback that gets called to inform the application that a firmware update from local storage just started.

- void `FileFirmwareUpdateCompletedHook` (void)

Callback that gets called to inform the application that a firmware update was successfully completed.

- void `FileFirmwareUpdateErrorHook` (blt_int8u error_code)

Callback that gets called in case an error occurred during a firmware update. Refer to [<file.h>](#) for a list of available error codes.

- void `FileFirmwareUpdateLogHook` (blt_char *info_string)

Callback that gets called each time new log information becomes available during a firmware update.

- blt_int8u `XcpGetSeedHook` (blt_int8u resource, blt_int8u *seed)

Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the `GET_SEED` command.

- blt_int8u `XcpVerifyKeyHook` (blt_int8u resource, blt_int8u *key, blt_int8u len)

Called by the `UNLOCK` command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

Variables

- static const `blt_char firmwareFilename` [] = `"/demoprogram_stm32f207.srec"`

Firmware filename.

-

```
struct {
    FIL handle
    blt_bool canUse
} logfile
```

Data structure for grouping log-file related information.

7.592.1 Detailed Description

Bootloader callback source file.

7.592.2 Function Documentation

7.592.2.1 BackDoorEntryHook()

```
blt_bool BackDoorEntryHook (
    void )
```

Checks if a backdoor entry is requested.

Returns

BLT_TRUE if the backdoor entry is requested, BLT_FALSE otherwise.

7.592.2.2 BackDoorInitHook()

```
void BackDoorInitHook (  
    void )
```

Initializes the backdoor entry option.

Returns

none.

7.592.2.3 CopInitHook()

```
void CopInitHook (  
    void )
```

Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.

Returns

none.

7.592.2.4 CopServiceHook()

```
void CopServiceHook (  
    void )
```

Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.

Returns

none.

7.592.2.5 CpuUserProgramStartHook()

```
blt_bool CpuUserProgramStartHook (  
    void )
```

Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.

Returns

BLT_TRUE if it is okay to start the user program, BLT_FALSE to keep keep the bootloader active.

7.592.2.6 FileFirmwareUpdateCompletedHook()

```
void FileFirmwareUpdateCompletedHook (
    void )
```

Callback that gets called to inform the application that a firmware update was successfully completed.

Returns

none.

7.592.2.7 FileFirmwareUpdateErrorHook()

```
void FileFirmwareUpdateErrorHook (
    blt_int8u error_code )
```

Callback that gets called in case an error occurred during a firmware update. Refer to <[file.h](#)> for a list of available error codes.

Returns

none.

7.592.2.8 FileFirmwareUpdateLogHook()

```
void FileFirmwareUpdateLogHook (
    blt_char * info_string )
```

Callback that gets called each time new log information becomes available during a firmware update.

Parameters

<i>info_string</i>	Pointer to a character array with the log entry info.
--------------------	---

Returns

none.

7.592.2.9 FileFirmwareUpdateStartedHook()

```
void FileFirmwareUpdateStartedHook (
    void )
```

Callback that gets called to inform the application that a firmware update from local storage just started.

Returns

none.

7.592.2.10 FileGetFirmwareFilenameHook()

```
const blt_char* FileGetFirmwareFilenameHook (  
    void )
```

Callback to obtain the filename of the firmware file that should be used during the firmware update from the local file storage. This hook function is called at the beginning of the firmware update from local storage sequence.

Returns

valid firmware filename with full path or BLT_NULL.

7.592.2.11 FileIsFirmwareUpdateRequestedHook()

```
blt_bool FileIsFirmwareUpdateRequestedHook (  
    void )
```

Callback that gets called to check whether a firmware update from local file storage should be started. This could for example be when a switch is pressed, when a certain file is found on the local file storage, etc.

Returns

BLT_TRUE if a firmware update is requested, BLT_FALSE otherwise.

7.592.2.12 NvmDoneHook()

```
blt_bool NvmDoneHook (  
    void )
```

Callback that gets called at the end of the NVM programming session.

Returns

BLT_TRUE is successful, BLT_FALSE otherwise.

7.592.2.13 NvmEraseHook()

```
blt_int8u NvmEraseHook (  
    blt_addr addr,  
    blt_int32u len )
```

Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the memory hasn't been erased yet.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.

Returns

BLT_NVM_OKAY if successful, BLT_NVM_NOT_IN_RANGE if the address is not within the supported memory range, or BLT_NVM_ERROR if the erase operation failed.

7.592.2.14 NvmInitHook()

```
void NvmInitHook (  
    void )
```

Callback that gets called at the start of the internal NVM driver initialization routine.

Returns

none.

7.592.2.15 NvmReinitHook()

```
void NvmReinitHook (  
    void )
```

Callback that gets called at the start of a firmware update to reinitialize the NVM driver.

Returns

none.

7.592.2.16 NvmWriteHook()

```
blt_int8u NvmWriteHook (  
    blt_addr addr,  
    blt_int32u len,  
    blt_int8u * data )
```

Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the data hasn't been written yet.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.
<i>data</i>	Pointer to the data buffer.

Returns

BLT_NVM_OKAY if successful, BLT_NVM_NOT_IN_RANGE if the address is not within the supported memory range, or BLT_NVM_ERROR if the write operation failed.

7.592.2.17 XcpGetSeedHook()

```
blt_int8u XcpGetSeedHook (
    blt_int8u resource,
    blt_int8u * seed )
```

Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the GET_SEED command.

Parameters

<i>resource</i>	Resource that the seed is requested for (XCP_RES_XXX).
<i>seed</i>	Pointer to byte buffer where the seed will be stored.

Returns

Length of the seed in bytes.

7.592.2.18 XcpVerifyKeyHook()

```
blt_int8u XcpVerifyKeyHook (
    blt_int8u resource,
    blt_int8u * key,
    blt_int8u len )
```

Called by the UNLOCK command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

Parameters

<i>resource</i>	resource to unlock (XCP_RES_XXX).
<i>key</i>	pointer to the byte buffer holding the key.
<i>len</i>	length of the key in bytes.

Returns

1 if the key was correct, 0 otherwise.

7.592.3 Variable Documentation

7.592.3.1 canUse

```
blt_bool canUse
```

Flag to indicate if the log-file can be used.

7.592.3.2 handle

```
FIL handle
```

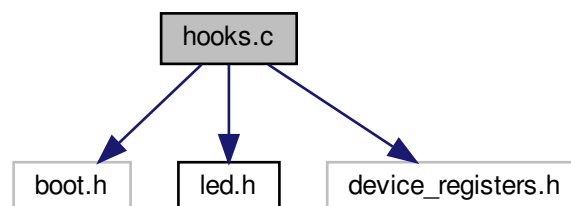
FatFS handle to the log-file.

7.593 hooks.c File Reference

Bootloader callback source file.

```
#include "boot.h"
#include "led.h"
#include "device_registers.h"
```

Include dependency graph for ARMCM4_S32K14_S32K144EVB_GCC/Boot/hooks.c:



Functions

- void [BackDoorInitHook](#) (void)
Initializes the backdoor entry option.
- [blt_bool BackDoorEntryHook](#) (void)
Checks if a backdoor entry is requested.
- [blt_bool CpuUserProgramStartHook](#) (void)
Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.
- void [CopInitHook](#) (void)
Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.
- void [CopServiceHook](#) (void)
Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.
- void [NvmInitHook](#) (void)
Callback that gets called at the start of the internal NVM driver initialization routine.
- void [NvmReinitHook](#) (void)
Callback that gets called at the start of a firmware update to reinitialize the NVM driver.
- [blt_int8u NvmWriteHook](#) ([blt_addr](#) addr, [blt_int32u](#) len, [blt_int8u](#) *data)
Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the data hasn't been written yet.
- [blt_int8u NvmEraseHook](#) ([blt_addr](#) addr, [blt_int32u](#) len)
Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the memory hasn't been erased yet.
- [blt_bool NvmDoneHook](#) (void)
Callback that gets called at the end of the NVM programming session.
- [blt_int8u XcpGetSeedHook](#) ([blt_int8u](#) resource, [blt_int8u](#) *seed)
Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the GET_SEED command.
- [blt_int8u XcpVerifyKeyHook](#) ([blt_int8u](#) resource, [blt_int8u](#) *key, [blt_int8u](#) len)
Called by the UNLOCK command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

7.593.1 Detailed Description

Bootloader callback source file.

7.593.2 Function Documentation

7.593.2.1 BackDoorEntryHook()

```
blt_bool BackDoorEntryHook (  
    void )
```

Checks if a backdoor entry is requested.

Returns

BLT_TRUE if the backdoor entry is requested, BLT_FALSE otherwise.

7.593.2.2 BackDoorInitHook()

```
void BackDoorInitHook (  
    void )
```

Initializes the backdoor entry option.

Returns

none.

7.593.2.3 CopInitHook()

```
void CopInitHook (  
    void )
```

Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.

Returns

none.

7.593.2.4 CopServiceHook()

```
void CopServiceHook (  
    void )
```

Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.

Returns

none.

7.593.2.5 CpuUserProgramStartHook()

```
blt_bool CpuUserProgramStartHook (
    void )
```

Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.

Returns

BLT_TRUE if it is okay to start the user program, BLT_FALSE to keep keep the bootloader active.

7.593.2.6 NvmDoneHook()

```
blt_bool NvmDoneHook (
    void )
```

Callback that gets called at the end of the NVM programming session.

Returns

BLT_TRUE is successful, BLT_FALSE otherwise.

7.593.2.7 NvmEraseHook()

```
blt_int8u NvmEraseHook (
    blt_addr addr,
    blt_int32u len )
```

Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the memory hasn't been erased yet.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.

Returns

BLT_NVM_OKAY if successful, BLT_NVM_NOT_IN_RANGE if the address is not within the supported memory range, or BLT_NVM_ERROR is the erase operation failed.

7.593.2.8 NvmInitHook()

```
void NvmInitHook (
    void )
```

Callback that gets called at the start of the internal NVM driver initialization routine.

Returns

none.

7.593.2.9 NvmReinitHook()

```
void NvmReinitHook (
    void )
```

Callback that gets called at the start of a firmware update to reinitialize the NVM driver.

Returns

none.

7.593.2.10 NvmWriteHook()

```
blt_int8u NvmWriteHook (
    blt_addr addr,
    blt_int32u len,
    blt_int8u * data )
```

Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the data hasn't been written yet.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.
<i>data</i>	Pointer to the data buffer.

Returns

BLT_NVM_OKAY if successful, BLT_NVM_NOT_IN_RANGE if the address is not within the supported memory range, or BLT_NVM_ERROR if the write operation failed.

7.593.2.11 XcpGetSeedHook()

```
blt_int8u XcpGetSeedHook (
    blt_int8u resource,
    blt_int8u * seed )
```

Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the GET_SEED command.

Parameters

<i>resource</i>	Resource that the seed is requested for (XCP_RES_XXX).
<i>seed</i>	Pointer to byte buffer where the seed will be stored.

Returns

Length of the seed in bytes.

7.593.2.12 XcpVerifyKeyHook()

```
blt_int8u XcpVerifyKeyHook (
    blt_int8u resource,
    blt_int8u * key,
    blt_int8u len )
```

Called by the UNLOCK command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

Parameters

<i>resource</i>	resource to unlock (XCP_RES_XXX).
<i>key</i>	pointer to the byte buffer holding the key.
<i>len</i>	length of the key in bytes.

Returns

1 if the key was correct, 0 otherwise.

7.594 hooks.c File Reference

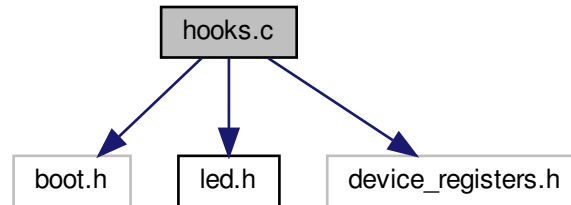
Bootloader callback source file.

```
#include "boot.h"
#include "led.h"
```



```
#include "device_registers.h"
```

Include dependency graph for ARMCM4_S32K14_S32K144EVB_IAR/Boot/hooks.c:



Functions

- void [BackDoorInitHook](#) (void)
Initializes the backdoor entry option.
- [blt_bool BackDoorEntryHook](#) (void)
Checks if a backdoor entry is requested.
- [blt_bool CpuUserProgramStartHook](#) (void)
Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.
- void [CopInitHook](#) (void)
Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.
- void [CopServiceHook](#) (void)
Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.
- void [NvmInitHook](#) (void)
Callback that gets called at the start of the internal NVM driver initialization routine.
- void [NvmReinitHook](#) (void)
Callback that gets called at the start of a firmware update to reinitialize the NVM driver.
- [blt_int8u NvmWriteHook](#) ([blt_addr](#) addr, [blt_int32u](#) len, [blt_int8u](#) *data)
Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the data hasn't been written yet.
- [blt_int8u NvmEraseHook](#) ([blt_addr](#) addr, [blt_int32u](#) len)
Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the memory hasn't been erased yet.
- [blt_bool NvmDoneHook](#) (void)
Callback that gets called at the end of the NVM programming session.
- [blt_int8u XcpGetSeedHook](#) ([blt_int8u](#) resource, [blt_int8u](#) *seed)
Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the GET_SEED command.
- [blt_int8u XcpVerifyKeyHook](#) ([blt_int8u](#) resource, [blt_int8u](#) *key, [blt_int8u](#) len)
Called by the UNLOCK command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

7.594.1 Detailed Description

Bootloader callback source file.

7.594.2 Function Documentation

7.594.2.1 BackDoorEntryHook()

```
blt_bool BackDoorEntryHook (  
    void )
```

Checks if a backdoor entry is requested.

Returns

BLT_TRUE if the backdoor entry is requested, BLT_FALSE otherwise.

7.594.2.2 BackDoorInitHook()

```
void BackDoorInitHook (  
    void )
```

Initializes the backdoor entry option.

Returns

none.

7.594.2.3 CopInitHook()

```
void CopInitHook (  
    void )
```

Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.

Returns

none.

7.594.2.4 CopServiceHook()

```
void CopServiceHook (
    void )
```

Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.

Returns

none.

7.594.2.5 CpuUserProgramStartHook()

```
blt_bool CpuUserProgramStartHook (
    void )
```

Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.

Returns

BLT_TRUE if it is okay to start the user program, BLT_FALSE to keep keep the bootloader active.

7.594.2.6 NvmDoneHook()

```
blt_bool NvmDoneHook (
    void )
```

Callback that gets called at the end of the NVM programming session.

Returns

BLT_TRUE is successful, BLT_FALSE otherwise.

7.594.2.7 NvmEraseHook()

```
blt_int8u NvmEraseHook (
    blt_addr addr,
    blt_int32u len )
```

Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the memory hasn't been erased yet.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.

Returns

BLT_NVM_OKAY if successful, BLT_NVM_NOT_IN_RANGE if the address is not within the supported memory range, or BLT_NVM_ERROR if the erase operation failed.

7.594.2.8 NvmInitHook()

```
void NvmInitHook (  
    void )
```

Callback that gets called at the start of the internal NVM driver initialization routine.

Returns

none.

7.594.2.9 NvmReinitHook()

```
void NvmReinitHook (  
    void )
```

Callback that gets called at the start of a firmware update to reinitialize the NVM driver.

Returns

none.

7.594.2.10 NvmWriteHook()

```
blt_int8u NvmWriteHook (  
    blt_addr addr,  
    blt_int32u len,  
    blt_int8u * data )
```

Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the data hasn't been written yet.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.
<i>data</i>	Pointer to the data buffer.

Returns

BLT_NVM_OKAY if successful, BLT_NVM_NOT_IN_RANGE if the address is not within the supported memory range, or BLT_NVM_ERROR if the write operation failed.

7.594.2.11 XcpGetSeedHook()

```
blt_int8u XcpGetSeedHook (
    blt_int8u resource,
    blt_int8u * seed )
```

Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the GET_SEED command.

Parameters

<i>resource</i>	Resource that the seed is requested for (XCP_RES_XXX).
<i>seed</i>	Pointer to byte buffer where the seed will be stored.

Returns

Length of the seed in bytes.

7.594.2.12 XcpVerifyKeyHook()

```
blt_int8u XcpVerifyKeyHook (
    blt_int8u resource,
    blt_int8u * key,
    blt_int8u len )
```

Called by the UNLOCK command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

Parameters

<i>resource</i>	resource to unlock (XCP_RES_XXX).
<i>key</i>	pointer to the byte buffer holding the key.
<i>len</i>	length of the key in bytes.

Returns

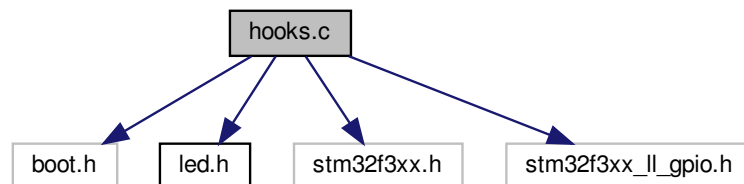
1 if the key was correct, 0 otherwise.

7.595 hooks.c File Reference

Bootloader callback source file.

```
#include "boot.h"
#include "led.h"
#include "stm32f3xx.h"
#include "stm32f3xx_ll_gpio.h"
```

Include dependency graph for ARMCM4_STM32F3_Discovery_F303VC_CubeIDE/Boot/App/hooks.c:



Functions

- [blt_bool CpuUserProgramStartHook](#) (void)
Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.
- void [UsbConnectHook](#) (blt_bool connect)
Callback that gets called whenever the USB device should be connected to the USB bus.
- void [UsbEnterLowPowerModeHook](#) (void)
Callback that gets called whenever the USB host requests the device to enter a low power mode.
- void [UsbLeaveLowPowerModeHook](#) (void)
Callback that gets called whenever the USB host requests the device to exit low power mode.
- void [CopInitHook](#) (void)
Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.
- void [CopServiceHook](#) (void)
Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.
- void [BackDoorInitHook](#) (void)
Initializes the backdoor entry option.
- [blt_bool BackDoorEntryHook](#) (void)
Checks if a backdoor entry is requested.
- void [NvmInitHook](#) (void)
Callback that gets called at the start of the internal NVM driver initialization routine.
- void [NvmReinitHook](#) (void)

Callback that gets called at the start of a firmware update to reinitialize the NVM driver.

- [blt_int8u NvmWriteHook](#) ([blt_addr](#) addr, [blt_int32u](#) len, [blt_int8u](#) *data)

Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the data hasn't been written yet.

- [blt_int8u NvmEraseHook](#) ([blt_addr](#) addr, [blt_int32u](#) len)

Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the memory hasn't been erased yet.

- [blt_bool NvmDoneHook](#) (void)

Callback that gets called at the end of the NVM programming session.

- [blt_int8u XcpGetSeedHook](#) ([blt_int8u](#) resource, [blt_int8u](#) *seed)

Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the GET_SEED command.

- [blt_int8u XcpVerifyKeyHook](#) ([blt_int8u](#) resource, [blt_int8u](#) *key, [blt_int8u](#) len)

Called by the UNLOCK command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

7.595.1 Detailed Description

Bootloader callback source file.

7.595.2 Function Documentation

7.595.2.1 BackDoorEntryHook()

```
blt_bool BackDoorEntryHook (
    void )
```

Checks if a backdoor entry is requested.

Returns

BLT_TRUE if the backdoor entry is requested, BLT_FALSE otherwise.

7.595.2.2 BackDoorInitHook()

```
void BackDoorInitHook (
    void )
```

Initializes the backdoor entry option.

Returns

none.

7.595.2.3 CopInitHook()

```
void CopInitHook (  
    void )
```

Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.

Returns

none.

7.595.2.4 CopServiceHook()

```
void CopServiceHook (  
    void )
```

Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.

Returns

none.

7.595.2.5 CpuUserProgramStartHook()

```
blt_bool CpuUserProgramStartHook (  
    void )
```

Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.

Returns

BLT_TRUE if it is okay to start the user program, BLT_FALSE to keep keep the bootloader active.

7.595.2.6 NvmDoneHook()

```
blt_bool NvmDoneHook (  
    void )
```

Callback that gets called at the end of the NVM programming session.

Returns

BLT_TRUE is successful, BLT_FALSE otherwise.

7.595.2.7 NvmEraseHook()

```
blt_int8u NvmEraseHook (
    blt_addr addr,
    blt_int32u len )
```

Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the memory hasn't been erased yet.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.

Returns

BLT_NVM_OKAY if successful, BLT_NVM_NOT_IN_RANGE if the address is not within the supported memory range, or BLT_NVM_ERROR if the erase operation failed.

7.595.2.8 NvmInitHook()

```
void NvmInitHook (
    void )
```

Callback that gets called at the start of the internal NVM driver initialization routine.

Returns

none.

7.595.2.9 NvmReinitHook()

```
void NvmReinitHook (
    void )
```

Callback that gets called at the start of a firmware update to reinitialize the NVM driver.

Returns

none.

7.595.2.10 NvmWriteHook()

```
blt_int8u NvmWriteHook (
    blt_addr addr,
    blt_int32u len,
    blt_int8u * data )
```

Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the data hasn't been written yet.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.
<i>data</i>	Pointer to the data buffer.

Returns

BLT_NVM_OKAY if successful, BLT_NVM_NOT_IN_RANGE if the address is not within the supported memory range, or BLT_NVM_ERROR if the write operation failed.

7.595.2.11 UsbConnectHook()

```
void UsbConnectHook (
    blt_bool connect )
```

Callback that gets called whenever the USB device should be connected to the USB bus.

Parameters

<i>connect</i>	BLT_TRUE to connect and BLT_FALSE to disconnect.
----------------	--

Returns

none.

7.595.2.12 UsbEnterLowPowerModeHook()

```
void UsbEnterLowPowerModeHook (
    void )
```

Callback that gets called whenever the USB host requests the device to enter a low power mode.

Returns

none.

7.595.2.13 UsbLeaveLowPowerModeHook()

```
void UsbLeaveLowPowerModeHook (
    void )
```

Callback that gets called whenever the USB host requests the device to exit low power mode.

Returns

none.

7.595.2.14 XcpGetSeedHook()

```
blt_int8u XcpGetSeedHook (
    blt_int8u resource,
    blt_int8u * seed )
```

Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the GET_SEED command.

Parameters

<i>resource</i>	Resource that the seed is requested for (XCP_RES_XXX).
<i>seed</i>	Pointer to byte buffer where the seed will be stored.

Returns

Length of the seed in bytes.

7.595.2.15 XcpVerifyKeyHook()

```
blt_int8u XcpVerifyKeyHook (
    blt_int8u resource,
    blt_int8u * key,
    blt_int8u len )
```

Called by the UNLOCK command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

Parameters

<i>resource</i>	resource to unlock (XCP_RES_XXX).
<i>key</i>	pointer to the byte buffer holding the key.
<i>len</i>	length of the key in bytes.

Returns

1 if the key was correct, 0 otherwise.

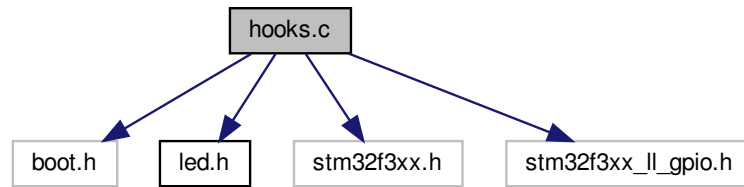
7.596 hooks.c File Reference

Bootloader callback source file.

```
#include "boot.h"
#include "led.h"
#include "stm32f3xx.h"
```

```
#include "stm32f3xx_ll_gpio.h"
```

Include dependency graph for ARMCM4_STM32F3_Discovery_F303VC_GCC/Boot/hooks.c:



Functions

- [blt_bool CpuUserProgramStartHook](#) (void)

Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.
- void [UsbConnectHook](#) (blt_bool connect)

Callback that gets called whenever the USB device should be connected to the USB bus.
- void [UsbEnterLowPowerModeHook](#) (void)

Callback that gets called whenever the USB host requests the device to enter a low power mode.
- void [UsbLeaveLowPowerModeHook](#) (void)

Callback that gets called whenever the USB host requests the device to exit low power mode.
- void [CopInitHook](#) (void)

Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.
- void [CopServiceHook](#) (void)

Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.
- void [BackDoorInitHook](#) (void)

Initializes the backdoor entry option.
- [blt_bool BackDoorEntryHook](#) (void)

Checks if a backdoor entry is requested.
- void [NvmInitHook](#) (void)

Callback that gets called at the start of the internal NVM driver initialization routine.
- void [NvmReinitHook](#) (void)

Callback that gets called at the start of a firmware update to reinitialize the NVM driver.
- [blt_int8u NvmWriteHook](#) (blt_addr addr, blt_int32u len, blt_int8u *data)

Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the data hasn't been written yet.
- [blt_int8u NvmEraseHook](#) (blt_addr addr, blt_int32u len)

Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the memory hasn't been erased yet.
- [blt_bool NvmDoneHook](#) (void)

Callback that gets called at the end of the NVM programming session.
- [blt_int8u XcpGetSeedHook](#) (blt_int8u resource, blt_int8u *seed)

Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the GET_SEED command.

- `blt_int8u XcpVerifyKeyHook (blt_int8u resource, blt_int8u *key, blt_int8u len)`

Called by the UNLOCK command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

7.596.1 Detailed Description

Bootloader callback source file.

7.596.2 Function Documentation

7.596.2.1 BackDoorEntryHook()

```
blt_bool BackDoorEntryHook (  
    void )
```

Checks if a backdoor entry is requested.

Returns

BLT_TRUE if the backdoor entry is requested, BLT_FALSE otherwise.

7.596.2.2 BackDoorInitHook()

```
void BackDoorInitHook (  
    void )
```

Initializes the backdoor entry option.

Returns

none.

7.596.2.3 CopInitHook()

```
void CopInitHook (  
    void )
```

Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.

Returns

none.

7.596.2.4 CopServiceHook()

```
void CopServiceHook (
    void )
```

Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.

Returns

none.

7.596.2.5 CpuUserProgramStartHook()

```
blt_bool CpuUserProgramStartHook (
    void )
```

Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.

Returns

BLT_TRUE if it is okay to start the user program, BLT_FALSE to keep keep the bootloader active.

7.596.2.6 NvmDoneHook()

```
blt_bool NvmDoneHook (
    void )
```

Callback that gets called at the end of the NVM programming session.

Returns

BLT_TRUE is successful, BLT_FALSE otherwise.

7.596.2.7 NvmEraseHook()

```
blt_int8u NvmEraseHook (
    blt_addr addr,
    blt_int32u len )
```

Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the memory hasn't been erased yet.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.

Returns

BLT_NVM_OKAY if successful, BLT_NVM_NOT_IN_RANGE if the address is not within the supported memory range, or BLT_NVM_ERROR if the erase operation failed.

7.596.2.8 NvmInitHook()

```
void NvmInitHook (  
    void )
```

Callback that gets called at the start of the internal NVM driver initialization routine.

Returns

none.

7.596.2.9 NvmReinitHook()

```
void NvmReinitHook (  
    void )
```

Callback that gets called at the start of a firmware update to reinitialize the NVM driver.

Returns

none.

7.596.2.10 NvmWriteHook()

```
blt_int8u NvmWriteHook (  
    blt_addr addr,  
    blt_int32u len,  
    blt_int8u * data )
```

Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the data hasn't been written yet.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.
<i>data</i>	Pointer to the data buffer.

Returns

BLT_NVM_OKAY if successful, BLT_NVM_NOT_IN_RANGE if the address is not within the supported memory range, or BLT_NVM_ERROR if the write operation failed.

7.596.2.11 UsbConnectHook()

```
void UsbConnectHook (
    blt_bool connect )
```

Callback that gets called whenever the USB device should be connected to the USB bus.

Parameters

<i>connect</i>	BLT_TRUE to connect and BLT_FALSE to disconnect.
----------------	--

Returns

none.

7.596.2.12 UsbEnterLowPowerModeHook()

```
void UsbEnterLowPowerModeHook (
    void )
```

Callback that gets called whenever the USB host requests the device to enter a low power mode.

Returns

none.

7.596.2.13 UsbLeaveLowPowerModeHook()

```
void UsbLeaveLowPowerModeHook (
    void )
```

Callback that gets called whenever the USB host requests the device to exit low power mode.

Returns

none.

7.596.2.14 XcpGetSeedHook()

```
blt_int8u XcpGetSeedHook (
    blt_int8u resource,
    blt_int8u * seed )
```

Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the GET_SEED command.

Parameters

<i>resource</i>	Resource that the seed is requested for (XCP_RES_XXX).
<i>seed</i>	Pointer to byte buffer where the seed will be stored.

Returns

Length of the seed in bytes.

7.596.2.15 XcpVerifyKeyHook()

```
blt_int8u XcpVerifyKeyHook (
    blt_int8u resource,
    blt_int8u * key,
    blt_int8u len )
```

Called by the UNLOCK command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

Parameters

<i>resource</i>	resource to unlock (XCP_RES_XXX).
<i>key</i>	pointer to the byte buffer holding the key.
<i>len</i>	length of the key in bytes.

Returns

1 if the key was correct, 0 otherwise.

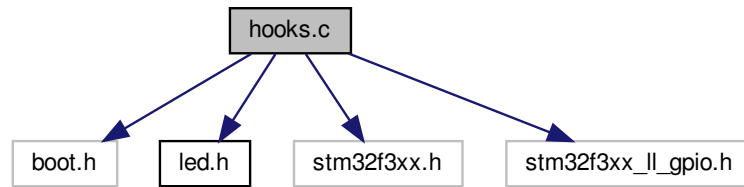
7.597 hooks.c File Reference

Bootloader callback source file.

```
#include "boot.h"
#include "led.h"
#include "stm32f3xx.h"
```

```
#include "stm32f3xx_ll_gpio.h"
```

Include dependency graph for ARMCM4_STM32F3_Discovery_F303VC_IAR/Boot/hooks.c:



Functions

- [blt_bool CpuUserProgramStartHook](#) (void)

Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.
- void [UsbConnectHook](#) ([blt_bool](#) connect)

Callback that gets called whenever the USB device should be connected to the USB bus.
- void [UsbEnterLowPowerModeHook](#) (void)

Callback that gets called whenever the USB host requests the device to enter a low power mode.
- void [UsbLeaveLowPowerModeHook](#) (void)

Callback that gets called whenever the USB host requests the device to exit low power mode.
- void [CopInitHook](#) (void)

Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.
- void [CopServiceHook](#) (void)

Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.
- void [BackDoorInitHook](#) (void)

Initializes the backdoor entry option.
- [blt_bool BackDoorEntryHook](#) (void)

Checks if a backdoor entry is requested.
- void [NvmInitHook](#) (void)

Callback that gets called at the start of the internal NVM driver initialization routine.
- void [NvmReinitHook](#) (void)

Callback that gets called at the start of a firmware update to reinitialize the NVM driver.
- [blt_int8u NvmWriteHook](#) ([blt_addr](#) addr, [blt_int32u](#) len, [blt_int8u](#) *data)

Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the data hasn't been written yet.
- [blt_int8u NvmEraseHook](#) ([blt_addr](#) addr, [blt_int32u](#) len)

Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the memory hasn't been erased yet.
- [blt_bool NvmDoneHook](#) (void)

Callback that gets called at the end of the NVM programming session.
- [blt_int8u XcpGetSeedHook](#) ([blt_int8u](#) resource, [blt_int8u](#) *seed)

Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the GET_SEED command.

- `blt_int8u XcpVerifyKeyHook (blt_int8u resource, blt_int8u *key, blt_int8u len)`

Called by the UNLOCK command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

7.597.1 Detailed Description

Bootloader callback source file.

7.597.2 Function Documentation

7.597.2.1 BackDoorEntryHook()

```
blt_bool BackDoorEntryHook (  
    void )
```

Checks if a backdoor entry is requested.

Returns

BLT_TRUE if the backdoor entry is requested, BLT_FALSE otherwise.

7.597.2.2 BackDoorInitHook()

```
void BackDoorInitHook (  
    void )
```

Initializes the backdoor entry option.

Returns

none.

7.597.2.3 CopInitHook()

```
void CopInitHook (  
    void )
```

Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.

Returns

none.

7.597.2.4 CopServiceHook()

```
void CopServiceHook (
    void )
```

Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.

Returns

none.

7.597.2.5 CpuUserProgramStartHook()

```
blt_bool CpuUserProgramStartHook (
    void )
```

Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.

Returns

BLT_TRUE if it is okay to start the user program, BLT_FALSE to keep keep the bootloader active.

7.597.2.6 NvmDoneHook()

```
blt_bool NvmDoneHook (
    void )
```

Callback that gets called at the end of the NVM programming session.

Returns

BLT_TRUE is successful, BLT_FALSE otherwise.

7.597.2.7 NvmEraseHook()

```
blt_int8u NvmEraseHook (
    blt_addr addr,
    blt_int32u len )
```

Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the memory hasn't been erased yet.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.

Returns

BLT_NVM_OKAY if successful, BLT_NVM_NOT_IN_RANGE if the address is not within the supported memory range, or BLT_NVM_ERROR if the erase operation failed.

7.597.2.8 NvmInitHook()

```
void NvmInitHook (  
    void )
```

Callback that gets called at the start of the internal NVM driver initialization routine.

Returns

none.

7.597.2.9 NvmReinitHook()

```
void NvmReinitHook (  
    void )
```

Callback that gets called at the start of a firmware update to reinitialize the NVM driver.

Returns

none.

7.597.2.10 NvmWriteHook()

```
blt_int8u NvmWriteHook (  
    blt_addr addr,  
    blt_int32u len,  
    blt_int8u * data )
```

Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the data hasn't been written yet.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.
<i>data</i>	Pointer to the data buffer.

Returns

BLT_NVM_OKAY if successful, BLT_NVM_NOT_IN_RANGE if the address is not within the supported memory range, or BLT_NVM_ERROR if the write operation failed.

7.597.2.11 UsbConnectHook()

```
void UsbConnectHook (
    blt_bool connect )
```

Callback that gets called whenever the USB device should be connected to the USB bus.

Parameters

<i>connect</i>	BLT_TRUE to connect and BLT_FALSE to disconnect.
----------------	--

Returns

none.

7.597.2.12 UsbEnterLowPowerModeHook()

```
void UsbEnterLowPowerModeHook (
    void )
```

Callback that gets called whenever the USB host requests the device to enter a low power mode.

Returns

none.

7.597.2.13 UsbLeaveLowPowerModeHook()

```
void UsbLeaveLowPowerModeHook (
    void )
```

Callback that gets called whenever the USB host requests the device to exit low power mode.

Returns

none.

7.597.2.14 XcpGetSeedHook()

```
blt_int8u XcpGetSeedHook (
    blt_int8u resource,
    blt_int8u * seed )
```

Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the GET_SEED command.

Parameters

<i>resource</i>	Resource that the seed is requested for (XCP_RES_XXX).
<i>seed</i>	Pointer to byte buffer where the seed will be stored.

Returns

Length of the seed in bytes.

7.597.2.15 XcpVerifyKeyHook()

```
blt_int8u XcpVerifyKeyHook (
    blt_int8u resource,
    blt_int8u * key,
    blt_int8u len )
```

Called by the UNLOCK command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

Parameters

<i>resource</i>	resource to unlock (XCP_RES_XXX).
<i>key</i>	pointer to the byte buffer holding the key.
<i>len</i>	length of the key in bytes.

Returns

1 if the key was correct, 0 otherwise.

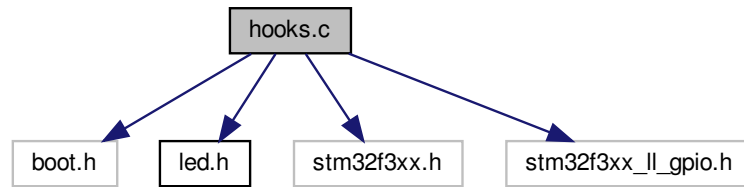
7.598 hooks.c File Reference

Bootloader callback source file.

```
#include "boot.h"
#include "led.h"
#include "stm32f3xx.h"
```

```
#include "stm32f3xx_ll_gpio.h"
```

Include dependency graph for ARMCM4_STM32F3_Discovery_F303VC_Keil/Boot/hooks.c:



Functions

- [blt_bool CpuUserProgramStartHook](#) (void)

Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.
- void [UsbConnectHook](#) ([blt_bool](#) connect)

Callback that gets called whenever the USB device should be connected to the USB bus.
- void [UsbEnterLowPowerModeHook](#) (void)

Callback that gets called whenever the USB host requests the device to enter a low power mode.
- void [UsbLeaveLowPowerModeHook](#) (void)

Callback that gets called whenever the USB host requests the device to exit low power mode.
- void [CopInitHook](#) (void)

Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.
- void [CopServiceHook](#) (void)

Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.
- void [BackDoorInitHook](#) (void)

Initializes the backdoor entry option.
- [blt_bool BackDoorEntryHook](#) (void)

Checks if a backdoor entry is requested.
- void [NvmInitHook](#) (void)

Callback that gets called at the start of the internal NVM driver initialization routine.
- void [NvmReinitHook](#) (void)

Callback that gets called at the start of a firmware update to reinitialize the NVM driver.
- [blt_int8u NvmWriteHook](#) ([blt_addr](#) addr, [blt_int32u](#) len, [blt_int8u](#) *data)

Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the data hasn't been written yet.
- [blt_int8u NvmEraseHook](#) ([blt_addr](#) addr, [blt_int32u](#) len)

Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the memory hasn't been erased yet.
- [blt_bool NvmDoneHook](#) (void)

Callback that gets called at the end of the NVM programming session.
- [blt_int8u XcpGetSeedHook](#) ([blt_int8u](#) resource, [blt_int8u](#) *seed)

Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the GET_SEED command.

- `blt_int8u XcpVerifyKeyHook (blt_int8u resource, blt_int8u *key, blt_int8u len)`

Called by the UNLOCK command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

7.598.1 Detailed Description

Bootloader callback source file.

7.598.2 Function Documentation

7.598.2.1 BackDoorEntryHook()

```
blt_bool BackDoorEntryHook (  
    void )
```

Checks if a backdoor entry is requested.

Returns

BLT_TRUE if the backdoor entry is requested, BLT_FALSE otherwise.

7.598.2.2 BackDoorInitHook()

```
void BackDoorInitHook (  
    void )
```

Initializes the backdoor entry option.

Returns

none.

7.598.2.3 CopInitHook()

```
void CopInitHook (  
    void )
```

Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.

Returns

none.

7.598.2.4 CopServiceHook()

```
void CopServiceHook (
    void )
```

Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.

Returns

none.

7.598.2.5 CpuUserProgramStartHook()

```
blt_bool CpuUserProgramStartHook (
    void )
```

Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.

Returns

BLT_TRUE if it is okay to start the user program, BLT_FALSE to keep keep the bootloader active.

7.598.2.6 NvmDoneHook()

```
blt_bool NvmDoneHook (
    void )
```

Callback that gets called at the end of the NVM programming session.

Returns

BLT_TRUE is successful, BLT_FALSE otherwise.

7.598.2.7 NvmEraseHook()

```
blt_int8u NvmEraseHook (
    blt_addr addr,
    blt_int32u len )
```

Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the memory hasn't been erased yet.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.

Returns

BLT_NVM_OKAY if successful, BLT_NVM_NOT_IN_RANGE if the address is not within the supported memory range, or BLT_NVM_ERROR if the erase operation failed.

7.598.2.8 NvmInitHook()

```
void NvmInitHook (  
    void )
```

Callback that gets called at the start of the internal NVM driver initialization routine.

Returns

none.

7.598.2.9 NvmReinitHook()

```
void NvmReinitHook (  
    void )
```

Callback that gets called at the start of a firmware update to reinitialize the NVM driver.

Returns

none.

7.598.2.10 NvmWriteHook()

```
blt_int8u NvmWriteHook (  
    blt_addr addr,  
    blt_int32u len,  
    blt_int8u * data )
```

Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the data hasn't been written yet.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.
<i>data</i>	Pointer to the data buffer.

Returns

BLT_NVM_OKAY if successful, BLT_NVM_NOT_IN_RANGE if the address is not within the supported memory range, or BLT_NVM_ERROR if the write operation failed.

7.598.2.11 UsbConnectHook()

```
void UsbConnectHook (
    blt_bool connect )
```

Callback that gets called whenever the USB device should be connected to the USB bus.

Parameters

<i>connect</i>	BLT_TRUE to connect and BLT_FALSE to disconnect.
----------------	--

Returns

none.

7.598.2.12 UsbEnterLowPowerModeHook()

```
void UsbEnterLowPowerModeHook (
    void )
```

Callback that gets called whenever the USB host requests the device to enter a low power mode.

Returns

none.

7.598.2.13 UsbLeaveLowPowerModeHook()

```
void UsbLeaveLowPowerModeHook (
    void )
```

Callback that gets called whenever the USB host requests the device to exit low power mode.

Returns

none.

7.598.2.14 XcpGetSeedHook()

```
blt_int8u XcpGetSeedHook (
    blt_int8u resource,
    blt_int8u * seed )
```

Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the GET_SEED command.

Parameters

<i>resource</i>	Resource that the seed is requested for (XCP_RES_XXX).
<i>seed</i>	Pointer to byte buffer where the seed will be stored.

Returns

Length of the seed in bytes.

7.598.2.15 XcpVerifyKeyHook()

```
blt_int8u XcpVerifyKeyHook (
    blt_int8u resource,
    blt_int8u * key,
    blt_int8u len )
```

Called by the UNLOCK command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

Parameters

<i>resource</i>	resource to unlock (XCP_RES_XXX).
<i>key</i>	pointer to the byte buffer holding the key.
<i>len</i>	length of the key in bytes.

Returns

1 if the key was correct, 0 otherwise.

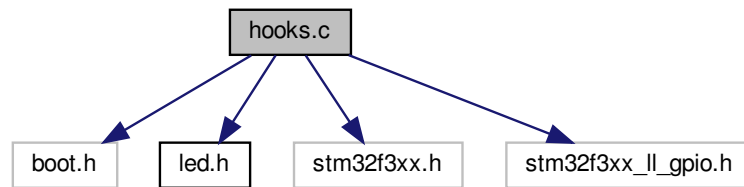
7.599 hooks.c File Reference

Bootloader callback source file.

```
#include "boot.h"
#include "led.h"
#include "stm32f3xx.h"
```

```
#include "stm32f3xx_ll_gpio.h"
```

Include dependency graph for ARMCM4_STM32F3_Nucleo_F303K8_CubeIDE/Boot/App/hooks.c:



Functions

- [blt_bool CpuUserProgramStartHook](#) (void)

Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.
- void [CopInitHook](#) (void)

Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.
- void [CopServiceHook](#) (void)

Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.
- void [BackDoorInitHook](#) (void)

Initializes the backdoor entry option.
- [blt_bool BackDoorEntryHook](#) (void)

Checks if a backdoor entry is requested.
- void [NvmInitHook](#) (void)

Callback that gets called at the start of the internal NVM driver initialization routine.
- void [NvmReinitHook](#) (void)

Callback that gets called at the start of a firmware update to reinitialize the NVM driver.
- [blt_int8u NvmWriteHook](#) ([blt_addr](#) addr, [blt_int32u](#) len, [blt_int8u](#) *data)

Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then `BLT_NVM_NOT_IN_RANGE` must be returned to indicate that the data hasn't been written yet.
- [blt_int8u NvmEraseHook](#) ([blt_addr](#) addr, [blt_int32u](#) len)

Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then `BLT_NVM_NOT_IN_RANGE` must be returned to indicate that the memory hasn't been erased yet.
- [blt_bool NvmDoneHook](#) (void)

Callback that gets called at the end of the NVM programming session.
- [blt_int8u XcpGetSeedHook](#) ([blt_int8u](#) resource, [blt_int8u](#) *seed)

Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the `GET_SEED` command.
- [blt_int8u XcpVerifyKeyHook](#) ([blt_int8u](#) resource, [blt_int8u](#) *key, [blt_int8u](#) len)

Called by the `UNLOCK` command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

7.599.1 Detailed Description

Bootloader callback source file.

7.599.2 Function Documentation

7.599.2.1 BackDoorEntryHook()

```
blt_bool BackDoorEntryHook (  
    void )
```

Checks if a backdoor entry is requested.

Returns

BLT_TRUE if the backdoor entry is requested, BLT_FALSE otherwise.

7.599.2.2 BackDoorInitHook()

```
void BackDoorInitHook (  
    void )
```

Initializes the backdoor entry option.

Returns

none.

7.599.2.3 CopInitHook()

```
void CopInitHook (  
    void )
```

Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.

Returns

none.

7.599.2.4 CopServiceHook()

```
void CopServiceHook (
    void )
```

Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.

Returns

none.

7.599.2.5 CpuUserProgramStartHook()

```
blt_bool CpuUserProgramStartHook (
    void )
```

Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.

Returns

BLT_TRUE if it is okay to start the user program, BLT_FALSE to keep keep the bootloader active.

7.599.2.6 NvmDoneHook()

```
blt_bool NvmDoneHook (
    void )
```

Callback that gets called at the end of the NVM programming session.

Returns

BLT_TRUE is successful, BLT_FALSE otherwise.

7.599.2.7 NvmEraseHook()

```
blt_int8u NvmEraseHook (
    blt_addr addr,
    blt_int32u len )
```

Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the memory hasn't been erased yet.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.

Returns

BLT_NVM_OKAY if successful, BLT_NVM_NOT_IN_RANGE if the address is not within the supported memory range, or BLT_NVM_ERROR if the erase operation failed.

7.599.2.8 NvmInitHook()

```
void NvmInitHook (
    void )
```

Callback that gets called at the start of the internal NVM driver initialization routine.

Returns

none.

7.599.2.9 NvmReinitHook()

```
void NvmReinitHook (
    void )
```

Callback that gets called at the start of a firmware update to reinitialize the NVM driver.

Returns

none.

7.599.2.10 NvmWriteHook()

```
blt_int8u NvmWriteHook (
    blt_addr addr,
    blt_int32u len,
    blt_int8u * data )
```

Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the data hasn't been written yet.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.
<i>data</i>	Pointer to the data buffer.

Returns

BLT_NVM_OKAY if successful, BLT_NVM_NOT_IN_RANGE if the address is not within the supported memory range, or BLT_NVM_ERROR if the write operation failed.

7.599.2.11 XcpGetSeedHook()

```
blt_int8u XcpGetSeedHook (
    blt_int8u resource,
    blt_int8u * seed )
```

Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the GET_SEED command.

Parameters

<i>resource</i>	Resource that the seed is requested for (XCP_RES_XXX).
<i>seed</i>	Pointer to byte buffer where the seed will be stored.

Returns

Length of the seed in bytes.

7.599.2.12 XcpVerifyKeyHook()

```
blt_int8u XcpVerifyKeyHook (
    blt_int8u resource,
    blt_int8u * key,
    blt_int8u len )
```

Called by the UNLOCK command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

Parameters

<i>resource</i>	resource to unlock (XCP_RES_XXX).
<i>key</i>	pointer to the byte buffer holding the key.
<i>len</i>	length of the key in bytes.

Returns

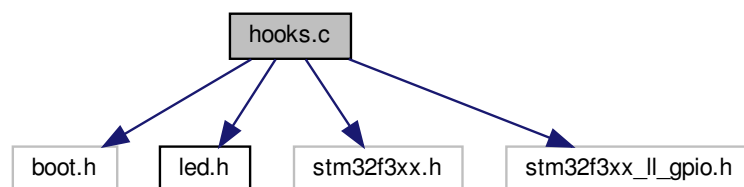
1 if the key was correct, 0 otherwise.

7.600 hooks.c File Reference

Bootloader callback source file.

```
#include "boot.h"
#include "led.h"
#include "stm32f3xx.h"
#include "stm32f3xx_ll_gpio.h"
```

Include dependency graph for ARMCM4_STM32F3_Nucleo_F303K8_GCC/Boot/hooks.c:



Functions

- [blt_bool CpuUserProgramStartHook](#) (void)
Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.
- void [CopInitHook](#) (void)
Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.
- void [CopServiceHook](#) (void)
Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.
- void [BackDoorInitHook](#) (void)
Initializes the backdoor entry option.
- [blt_bool BackDoorEntryHook](#) (void)
Checks if a backdoor entry is requested.
- void [NvmInitHook](#) (void)
Callback that gets called at the start of the internal NVM driver initialization routine.
- void [NvmReinitHook](#) (void)
Callback that gets called at the start of a firmware update to reinitialize the NVM driver.
- [blt_int8u NvmWriteHook](#) ([blt_addr](#) addr, [blt_int32u](#) len, [blt_int8u](#) *data)
Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the data hasn't been written yet.
- [blt_int8u NvmEraseHook](#) ([blt_addr](#) addr, [blt_int32u](#) len)

Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then `BLT_NVM_NOT_IN_RANGE` must be returned to indicate that the memory hasn't been erased yet.

- `blt_bool NvmDoneHook` (void)

Callback that gets called at the end of the NVM programming session.

- `blt_int8u XcpGetSeedHook` (`blt_int8u` resource, `blt_int8u *`seed)

Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the `GET_SEED` command.

- `blt_int8u XcpVerifyKeyHook` (`blt_int8u` resource, `blt_int8u *`key, `blt_int8u` len)

Called by the `UNLOCK` command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

7.600.1 Detailed Description

Bootloader callback source file.

7.600.2 Function Documentation

7.600.2.1 BackDoorEntryHook()

```
blt_bool BackDoorEntryHook (
    void )
```

Checks if a backdoor entry is requested.

Returns

`BLT_TRUE` if the backdoor entry is requested, `BLT_FALSE` otherwise.

7.600.2.2 BackDoorInitHook()

```
void BackDoorInitHook (
    void )
```

Initializes the backdoor entry option.

Returns

none.

7.600.2.3 CopInitHook()

```
void CopInitHook (
    void )
```

Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.

Returns

none.

7.600.2.4 CopServiceHook()

```
void CopServiceHook (
    void )
```

Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.

Returns

none.

7.600.2.5 CpuUserProgramStartHook()

```
blt_bool CpuUserProgramStartHook (
    void )
```

Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.

Returns

BLT_TRUE if it is okay to start the user program, BLT_FALSE to keep keep the bootloader active.

7.600.2.6 NvmDoneHook()

```
blt_bool NvmDoneHook (
    void )
```

Callback that gets called at the end of the NVM programming session.

Returns

BLT_TRUE is successful, BLT_FALSE otherwise.

7.600.2.7 NvmEraseHook()

```
blt_int8u NvmEraseHook (
    blt_addr addr,
    blt_int32u len )
```

Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the memory hasn't been erased yet.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.

Returns

BLT_NVM_OKAY if successful, BLT_NVM_NOT_IN_RANGE if the address is not within the supported memory range, or BLT_NVM_ERROR if the erase operation failed.

7.600.2.8 NvmInitHook()

```
void NvmInitHook (
    void )
```

Callback that gets called at the start of the internal NVM driver initialization routine.

Returns

none.

7.600.2.9 NvmReinitHook()

```
void NvmReinitHook (
    void )
```

Callback that gets called at the start of a firmware update to reinitialize the NVM driver.

Returns

none.

7.600.2.10 NvmWriteHook()

```
blt_int8u NvmWriteHook (
    blt_addr addr,
    blt_int32u len,
    blt_int8u * data )
```

Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the data hasn't been written yet.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.
<i>data</i>	Pointer to the data buffer.

Returns

BLT_NVM_OKAY if successful, BLT_NVM_NOT_IN_RANGE if the address is not within the supported memory range, or BLT_NVM_ERROR if the write operation failed.

7.600.2.11 XcpGetSeedHook()

```
blt_int8u XcpGetSeedHook (
    blt_int8u resource,
    blt_int8u * seed )
```

Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the GET_SEED command.

Parameters

<i>resource</i>	Resource that the seed is requested for (XCP_RES_XXX).
<i>seed</i>	Pointer to byte buffer where the seed will be stored.

Returns

Length of the seed in bytes.

7.600.2.12 XcpVerifyKeyHook()

```
blt_int8u XcpVerifyKeyHook (
    blt_int8u resource,
    blt_int8u * key,
    blt_int8u len )
```

Called by the UNLOCK command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

Parameters

<i>resource</i>	resource to unlock (XCP_RES_XXX).
<i>key</i>	pointer to the byte buffer holding the key.
<i>len</i>	length of the key in bytes.

Returns

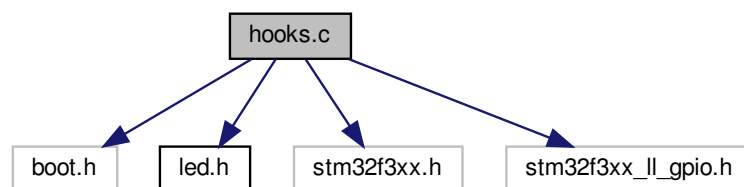
1 if the key was correct, 0 otherwise.

7.601 hooks.c File Reference

Bootloader callback source file.

```
#include "boot.h"
#include "led.h"
#include "stm32f3xx.h"
#include "stm32f3xx_ll_gpio.h"
```

Include dependency graph for ARMCM4_STM32F3_Nucleo_F303K8_IAR/Boot/hooks.c:



Functions

- [blt_bool CpuUserProgramStartHook](#) (void)
Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.
- void [CopInitHook](#) (void)
Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.
- void [CopServiceHook](#) (void)
Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.
- void [BackDoorInitHook](#) (void)
Initializes the backdoor entry option.
- [blt_bool BackDoorEntryHook](#) (void)
Checks if a backdoor entry is requested.
- void [NvmInitHook](#) (void)
Callback that gets called at the start of the internal NVM driver initialization routine.
- void [NvmReinitHook](#) (void)
Callback that gets called at the start of a firmware update to reinitialize the NVM driver.
- [blt_int8u NvmWriteHook](#) ([blt_addr](#) addr, [blt_int32u](#) len, [blt_int8u](#) *data)
Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the data hasn't been written yet.
- [blt_int8u NvmEraseHook](#) ([blt_addr](#) addr, [blt_int32u](#) len)

Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then `BLT_NVM_NOT_IN_RANGE` must be returned to indicate that the memory hasn't been erased yet.

- `blt_bool NvmDoneHook` (void)

Callback that gets called at the end of the NVM programming session.

- `blt_int8u XcpGetSeedHook` (`blt_int8u` resource, `blt_int8u` *seed)

Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the `GET_SEED` command.

- `blt_int8u XcpVerifyKeyHook` (`blt_int8u` resource, `blt_int8u` *key, `blt_int8u` len)

Called by the `UNLOCK` command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

7.601.1 Detailed Description

Bootloader callback source file.

7.601.2 Function Documentation

7.601.2.1 BackDoorEntryHook()

```
blt_bool BackDoorEntryHook (
    void )
```

Checks if a backdoor entry is requested.

Returns

`BLT_TRUE` if the backdoor entry is requested, `BLT_FALSE` otherwise.

7.601.2.2 BackDoorInitHook()

```
void BackDoorInitHook (
    void )
```

Initializes the backdoor entry option.

Returns

none.

7.601.2.3 CopInitHook()

```
void CopInitHook (  
    void )
```

Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.

Returns

none.

7.601.2.4 CopServiceHook()

```
void CopServiceHook (  
    void )
```

Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.

Returns

none.

7.601.2.5 CpuUserProgramStartHook()

```
blt_bool CpuUserProgramStartHook (  
    void )
```

Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.

Returns

BLT_TRUE if it is okay to start the user program, BLT_FALSE to keep keep the bootloader active.

7.601.2.6 NvmDoneHook()

```
blt_bool NvmDoneHook (  
    void )
```

Callback that gets called at the end of the NVM programming session.

Returns

BLT_TRUE is successful, BLT_FALSE otherwise.

7.601.2.7 NvmEraseHook()

```
blt_int8u NvmEraseHook (
    blt_addr addr,
    blt_int32u len )
```

Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the memory hasn't been erased yet.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.

Returns

BLT_NVM_OKAY if successful, BLT_NVM_NOT_IN_RANGE if the address is not within the supported memory range, or BLT_NVM_ERROR if the erase operation failed.

7.601.2.8 NvmInitHook()

```
void NvmInitHook (
    void )
```

Callback that gets called at the start of the internal NVM driver initialization routine.

Returns

none.

7.601.2.9 NvmReinitHook()

```
void NvmReinitHook (
    void )
```

Callback that gets called at the start of a firmware update to reinitialize the NVM driver.

Returns

none.

7.601.2.10 NvmWriteHook()

```
blt_int8u NvmWriteHook (
    blt_addr addr,
    blt_int32u len,
    blt_int8u * data )
```

Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the data hasn't been written yet.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.
<i>data</i>	Pointer to the data buffer.

Returns

BLT_NVM_OKAY if successful, BLT_NVM_NOT_IN_RANGE if the address is not within the supported memory range, or BLT_NVM_ERROR if the write operation failed.

7.601.2.11 XcpGetSeedHook()

```
blt_int8u XcpGetSeedHook (
    blt_int8u resource,
    blt_int8u * seed )
```

Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the GET_SEED command.

Parameters

<i>resource</i>	Resource that the seed is requested for (XCP_RES_XXX).
<i>seed</i>	Pointer to byte buffer where the seed will be stored.

Returns

Length of the seed in bytes.

7.601.2.12 XcpVerifyKeyHook()

```
blt_int8u XcpVerifyKeyHook (
    blt_int8u resource,
    blt_int8u * key,
    blt_int8u len )
```

Called by the UNLOCK command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

Parameters

<i>resource</i>	resource to unlock (XCP_RES_XXX).
<i>key</i>	pointer to the byte buffer holding the key.
<i>len</i>	length of the key in bytes.

Returns

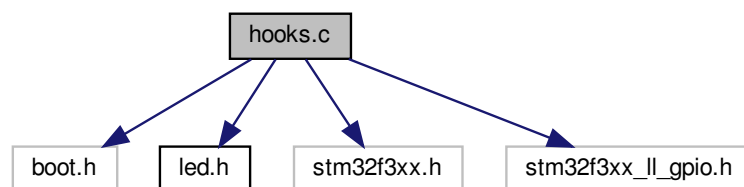
1 if the key was correct, 0 otherwise.

7.602 hooks.c File Reference

Bootloader callback source file.

```
#include "boot.h"
#include "led.h"
#include "stm32f3xx.h"
#include "stm32f3xx_ll_gpio.h"
```

Include dependency graph for ARMCM4_STM32F3_Nucleo_F303K8_Keil/Boot/hooks.c:



Functions

- [blt_bool CpuUserProgramStartHook](#) (void)
Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.
- void [CopInitHook](#) (void)
Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.
- void [CopServiceHook](#) (void)
Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.
- void [BackDoorInitHook](#) (void)
Initializes the backdoor entry option.
- [blt_bool BackDoorEntryHook](#) (void)
Checks if a backdoor entry is requested.
- void [NvmInitHook](#) (void)
Callback that gets called at the start of the internal NVM driver initialization routine.
- void [NvmReinitHook](#) (void)
Callback that gets called at the start of a firmware update to reinitialize the NVM driver.
- [blt_int8u NvmWriteHook](#) ([blt_addr](#) addr, [blt_int32u](#) len, [blt_int8u](#) *data)
Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the data hasn't been written yet.
- [blt_int8u NvmEraseHook](#) ([blt_addr](#) addr, [blt_int32u](#) len)

Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then `BLT_NVM_NOT_IN_RANGE` must be returned to indicate that the memory hasn't been erased yet.

- `blt_bool NvmDoneHook` (void)

Callback that gets called at the end of the NVM programming session.

- `blt_int8u XcpGetSeedHook` (`blt_int8u` resource, `blt_int8u *seed`)

Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the `GET_SEED` command.

- `blt_int8u XcpVerifyKeyHook` (`blt_int8u` resource, `blt_int8u *key`, `blt_int8u len`)

Called by the `UNLOCK` command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

7.602.1 Detailed Description

Bootloader callback source file.

7.602.2 Function Documentation

7.602.2.1 BackDoorEntryHook()

```
blt_bool BackDoorEntryHook (
    void )
```

Checks if a backdoor entry is requested.

Returns

`BLT_TRUE` if the backdoor entry is requested, `BLT_FALSE` otherwise.

7.602.2.2 BackDoorInitHook()

```
void BackDoorInitHook (
    void )
```

Initializes the backdoor entry option.

Returns

none.

7.602.2.3 CopInitHook()

```
void CopInitHook (
    void )
```

Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.

Returns

none.

7.602.2.4 CopServiceHook()

```
void CopServiceHook (
    void )
```

Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.

Returns

none.

7.602.2.5 CpuUserProgramStartHook()

```
blt_bool CpuUserProgramStartHook (
    void )
```

Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.

Returns

BLT_TRUE if it is okay to start the user program, BLT_FALSE to keep keep the bootloader active.

7.602.2.6 NvmDoneHook()

```
blt_bool NvmDoneHook (
    void )
```

Callback that gets called at the end of the NVM programming session.

Returns

BLT_TRUE is successful, BLT_FALSE otherwise.

7.602.2.7 NvmEraseHook()

```
blt_int8u NvmEraseHook (
    blt_addr addr,
    blt_int32u len )
```

Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the memory hasn't been erased yet.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.

Returns

BLT_NVM_OKAY if successful, BLT_NVM_NOT_IN_RANGE if the address is not within the supported memory range, or BLT_NVM_ERROR if the erase operation failed.

7.602.2.8 NvmInitHook()

```
void NvmInitHook (
    void )
```

Callback that gets called at the start of the internal NVM driver initialization routine.

Returns

none.

7.602.2.9 NvmReinitHook()

```
void NvmReinitHook (
    void )
```

Callback that gets called at the start of a firmware update to reinitialize the NVM driver.

Returns

none.

7.602.2.10 NvmWriteHook()

```
blt_int8u NvmWriteHook (
    blt_addr addr,
    blt_int32u len,
    blt_int8u * data )
```

Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the data hasn't been written yet.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.
<i>data</i>	Pointer to the data buffer.

Returns

BLT_NVM_OKAY if successful, BLT_NVM_NOT_IN_RANGE if the address is not within the supported memory range, or BLT_NVM_ERROR if the write operation failed.

7.602.2.11 XcpGetSeedHook()

```
blt_int8u XcpGetSeedHook (
    blt_int8u resource,
    blt_int8u * seed )
```

Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the GET_SEED command.

Parameters

<i>resource</i>	Resource that the seed is requested for (XCP_RES_XXX).
<i>seed</i>	Pointer to byte buffer where the seed will be stored.

Returns

Length of the seed in bytes.

7.602.2.12 XcpVerifyKeyHook()

```
blt_int8u XcpVerifyKeyHook (
    blt_int8u resource,
    blt_int8u * key,
    blt_int8u len )
```

Called by the UNLOCK command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

Parameters

<i>resource</i>	resource to unlock (XCP_RES_XXX).
<i>key</i>	pointer to the byte buffer holding the key.
<i>len</i>	length of the key in bytes.

Returns

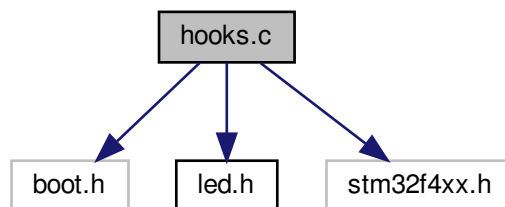
1 if the key was correct, 0 otherwise.

7.603 hooks.c File Reference

Bootloader callback source file.

```
#include "boot.h"
#include "led.h"
#include "stm32f4xx.h"
```

Include dependency graph for ARMCM4_STM32F4_Nucleo_F429ZI_CubeIDE/Boot/App/hooks.c:



Functions

- void [BackDoorInitHook](#) (void)
Initializes the backdoor entry option.
- [blt_bool BackDoorEntryHook](#) (void)
Checks if a backdoor entry is requested.
- [blt_bool CpuUserProgramStartHook](#) (void)
Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.
- void [CopInitHook](#) (void)
Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.
- void [CopServiceHook](#) (void)
Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.
- void [UsbConnectHook](#) ([blt_bool](#) connect)
Callback that gets called whenever the USB device should be connected to the USB bus.
- void [UsbEnterLowPowerModeHook](#) (void)
Callback that gets called whenever the USB host requests the device to enter a low power mode.
- void [UsbLeaveLowPowerModeHook](#) (void)
Callback that gets called whenever the USB host requests the device to exit low power mode.
- void [NvmInitHook](#) (void)
Callback that gets called at the start of the internal NVM driver initialization routine.
- void [NvmReinitHook](#) (void)

Callback that gets called at the start of a firmware update to reinitialize the NVM driver.

- `blt_int8u NvmWriteHook (blt_addr addr, blt_int32u len, blt_int8u *data)`

Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then `BLT_NVM_NOT_IN_RANGE` must be returned to indicate that the data hasn't been written yet.

- `blt_int8u NvmEraseHook (blt_addr addr, blt_int32u len)`

Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then `BLT_NVM_NOT_IN_RANGE` must be returned to indicate that the memory hasn't been erased yet.

- `blt_bool NvmDoneHook (void)`

Callback that gets called at the end of the NVM programming session.

- `blt_int8u XcpGetSeedHook (blt_int8u resource, blt_int8u *seed)`

Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the `GET_SEED` command.

- `blt_int8u XcpVerifyKeyHook (blt_int8u resource, blt_int8u *key, blt_int8u len)`

Called by the `UNLOCK` command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

7.603.1 Detailed Description

Bootloader callback source file.

7.603.2 Function Documentation

7.603.2.1 BackDoorEntryHook()

```
blt_bool BackDoorEntryHook (
    void )
```

Checks if a backdoor entry is requested.

Returns

`BLT_TRUE` if the backdoor entry is requested, `BLT_FALSE` otherwise.

7.603.2.2 BackDoorInitHook()

```
void BackDoorInitHook (
    void )
```

Initializes the backdoor entry option.

Returns

none.

7.603.2.3 CopInitHook()

```
void CopInitHook (
    void )
```

Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.

Returns

none.

7.603.2.4 CopServiceHook()

```
void CopServiceHook (
    void )
```

Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.

Returns

none.

7.603.2.5 CpuUserProgramStartHook()

```
blt_bool CpuUserProgramStartHook (
    void )
```

Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.

Returns

BLT_TRUE if it is okay to start the user program, BLT_FALSE to keep keep the bootloader active.

7.603.2.6 NvmDoneHook()

```
blt_bool NvmDoneHook (
    void )
```

Callback that gets called at the end of the NVM programming session.

Returns

BLT_TRUE is successful, BLT_FALSE otherwise.

7.603.2.7 NvmEraseHook()

```
blt_int8u NvmEraseHook (
    blt_addr addr,
    blt_int32u len )
```

Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the memory hasn't been erased yet.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.

Returns

BLT_NVM_OKAY if successful, BLT_NVM_NOT_IN_RANGE if the address is not within the supported memory range, or BLT_NVM_ERROR if the erase operation failed.

7.603.2.8 NvmInitHook()

```
void NvmInitHook (
    void )
```

Callback that gets called at the start of the internal NVM driver initialization routine.

Returns

none.

7.603.2.9 NvmReinitHook()

```
void NvmReinitHook (
    void )
```

Callback that gets called at the start of a firmware update to reinitialize the NVM driver.

Returns

none.

7.603.2.10 NvmWriteHook()

```
blt_int8u NvmWriteHook (
    blt_addr addr,
    blt_int32u len,
    blt_int8u * data )
```

Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the data hasn't been written yet.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.
<i>data</i>	Pointer to the data buffer.

Returns

BLT_NVM_OKAY if successful, BLT_NVM_NOT_IN_RANGE if the address is not within the supported memory range, or BLT_NVM_ERROR if the write operation failed.

7.603.2.11 UsbConnectHook()

```
void UsbConnectHook (
    blt_bool connect )
```

Callback that gets called whenever the USB device should be connected to the USB bus.

Parameters

<i>connect</i>	BLT_TRUE to connect and BLT_FALSE to disconnect.
----------------	--

Returns

none.

7.603.2.12 UsbEnterLowPowerModeHook()

```
void UsbEnterLowPowerModeHook (
    void )
```

Callback that gets called whenever the USB host requests the device to enter a low power mode.

Returns

none.

7.603.2.13 UsbLeaveLowPowerModeHook()

```
void UsbLeaveLowPowerModeHook (
    void )
```

Callback that gets called whenever the USB host requests the device to exit low power mode.

Returns

none.

7.603.2.14 XcpGetSeedHook()

```
blt_int8u XcpGetSeedHook (
    blt_int8u resource,
    blt_int8u * seed )
```

Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the GET_SEED command.

Parameters

<i>resource</i>	Resource that the seed is requested for (XCP_RES_XXX).
<i>seed</i>	Pointer to byte buffer where the seed will be stored.

Returns

Length of the seed in bytes.

7.603.2.15 XcpVerifyKeyHook()

```
blt_int8u XcpVerifyKeyHook (
    blt_int8u resource,
    blt_int8u * key,
    blt_int8u len )
```

Called by the UNLOCK command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

Parameters

<i>resource</i>	resource to unlock (XCP_RES_XXX).
<i>key</i>	pointer to the byte buffer holding the key.
<i>len</i>	length of the key in bytes.

Returns

1 if the key was correct, 0 otherwise.

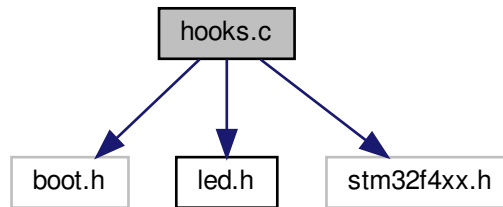
7.604 hooks.c File Reference

Bootloader callback source file.

```
#include "boot.h"
#include "led.h"
```

```
#include "stm32f4xx.h"
```

Include dependency graph for ARMCM4_STM32F4_Nucleo_F429ZI_GCC/Boot/hooks.c:



Functions

- void [BackDoorInitHook](#) (void)
Initializes the backdoor entry option.
- [blt_bool](#) [BackDoorEntryHook](#) (void)
Checks if a backdoor entry is requested.
- [blt_bool](#) [CpuUserProgramStartHook](#) (void)
Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.
- void [CopInitHook](#) (void)
Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.
- void [CopServiceHook](#) (void)
Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.
- void [UsbConnectHook](#) ([blt_bool](#) connect)
Callback that gets called whenever the USB device should be connected to the USB bus.
- void [UsbEnterLowPowerModeHook](#) (void)
Callback that gets called whenever the USB host requests the device to enter a low power mode.
- void [UsbLeaveLowPowerModeHook](#) (void)
Callback that gets called whenever the USB host requests the device to exit low power mode.
- void [NvmInitHook](#) (void)
Callback that gets called at the start of the internal NVM driver initialization routine.
- void [NvmReinitHook](#) (void)
Callback that gets called at the start of a firmware update to reinitialize the NVM driver.
- [blt_int8u](#) [NvmWriteHook](#) ([blt_addr](#) addr, [blt_int32u](#) len, [blt_int8u](#) *data)
Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the data hasn't been written yet.
- [blt_int8u](#) [NvmEraseHook](#) ([blt_addr](#) addr, [blt_int32u](#) len)
Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the memory hasn't been erased yet.
- [blt_bool](#) [NvmDoneHook](#) (void)
Callback that gets called at the end of the NVM programming session.

- [blt_int8u XcpGetSeedHook](#) ([blt_int8u](#) resource, [blt_int8u](#) *seed)

Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the GET_SEED command.

- [blt_int8u XcpVerifyKeyHook](#) ([blt_int8u](#) resource, [blt_int8u](#) *key, [blt_int8u](#) len)

Called by the UNLOCK command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

7.604.1 Detailed Description

Bootloader callback source file.

7.604.2 Function Documentation

7.604.2.1 BackDoorEntryHook()

```
blt_bool BackDoorEntryHook (  
    void )
```

Checks if a backdoor entry is requested.

Returns

BLT_TRUE if the backdoor entry is requested, BLT_FALSE otherwise.

7.604.2.2 BackDoorInitHook()

```
void BackDoorInitHook (  
    void )
```

Initializes the backdoor entry option.

Returns

none.

7.604.2.3 CopInitHook()

```
void CopInitHook (  
    void )
```

Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.

Returns

none.

7.604.2.4 CopServiceHook()

```
void CopServiceHook (  
    void )
```

Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.

Returns

none.

7.604.2.5 CpuUserProgramStartHook()

```
blt_bool CpuUserProgramStartHook (  
    void )
```

Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.

Returns

BLT_TRUE if it is okay to start the user program, BLT_FALSE to keep keep the bootloader active.

7.604.2.6 NvmDoneHook()

```
blt_bool NvmDoneHook (  
    void )
```

Callback that gets called at the end of the NVM programming session.

Returns

BLT_TRUE is successful, BLT_FALSE otherwise.

7.604.2.7 NvmEraseHook()

```
blt_int8u NvmEraseHook (
    blt_addr addr,
    blt_int32u len )
```

Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the memory hasn't been erased yet.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.

Returns

BLT_NVM_OKAY if successful, BLT_NVM_NOT_IN_RANGE if the address is not within the supported memory range, or BLT_NVM_ERROR if the erase operation failed.

7.604.2.8 NvmInitHook()

```
void NvmInitHook (
    void )
```

Callback that gets called at the start of the internal NVM driver initialization routine.

Returns

none.

7.604.2.9 NvmReinitHook()

```
void NvmReinitHook (
    void )
```

Callback that gets called at the start of a firmware update to reinitialize the NVM driver.

Returns

none.

7.604.2.10 NvmWriteHook()

```
blt_int8u NvmWriteHook (
    blt_addr addr,
    blt_int32u len,
    blt_int8u * data )
```

Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the data hasn't been written yet.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.
<i>data</i>	Pointer to the data buffer.

Returns

BLT_NVM_OKAY if successful, BLT_NVM_NOT_IN_RANGE if the address is not within the supported memory range, or BLT_NVM_ERROR if the write operation failed.

7.604.2.11 UsbConnectHook()

```
void UsbConnectHook (
    blt_bool connect )
```

Callback that gets called whenever the USB device should be connected to the USB bus.

Parameters

<i>connect</i>	BLT_TRUE to connect and BLT_FALSE to disconnect.
----------------	--

Returns

none.

7.604.2.12 UsbEnterLowPowerModeHook()

```
void UsbEnterLowPowerModeHook (
    void )
```

Callback that gets called whenever the USB host requests the device to enter a low power mode.

Returns

none.

7.604.2.13 UsbLeaveLowPowerModeHook()

```
void UsbLeaveLowPowerModeHook (
    void )
```

Callback that gets called whenever the USB host requests the device to exit low power mode.

Returns

none.

7.604.2.14 XcpGetSeedHook()

```
blt_int8u XcpGetSeedHook (
    blt_int8u resource,
    blt_int8u * seed )
```

Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the GET_SEED command.

Parameters

<i>resource</i>	Resource that the seed is requested for (XCP_RES_XXX).
<i>seed</i>	Pointer to byte buffer where the seed will be stored.

Returns

Length of the seed in bytes.

7.604.2.15 XcpVerifyKeyHook()

```
blt_int8u XcpVerifyKeyHook (
    blt_int8u resource,
    blt_int8u * key,
    blt_int8u len )
```

Called by the UNLOCK command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

Parameters

<i>resource</i>	resource to unlock (XCP_RES_XXX).
<i>key</i>	pointer to the byte buffer holding the key.
<i>len</i>	length of the key in bytes.

Returns

1 if the key was correct, 0 otherwise.

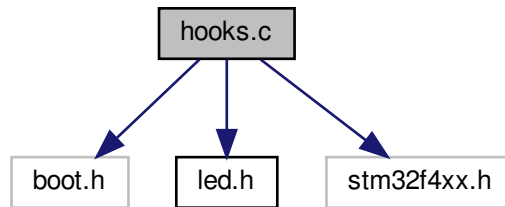
7.605 hooks.c File Reference

Bootloader callback source file.

```
#include "boot.h"
#include "led.h"
```

```
#include "stm32f4xx.h"
```

Include dependency graph for ARMC4_STM32F4_Nucleo_F429ZI_IAR/Boot/hooks.c:



Functions

- void [BackDoorInitHook](#) (void)
Initializes the backdoor entry option.
- [blt_bool](#) [BackDoorEntryHook](#) (void)
Checks if a backdoor entry is requested.
- [blt_bool](#) [CpuUserProgramStartHook](#) (void)
Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.
- void [CopInitHook](#) (void)
Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.
- void [CopServiceHook](#) (void)
Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.
- void [UsbConnectHook](#) ([blt_bool](#) connect)
Callback that gets called whenever the USB device should be connected to the USB bus.
- void [UsbEnterLowPowerModeHook](#) (void)
Callback that gets called whenever the USB host requests the device to enter a low power mode.
- void [UsbLeaveLowPowerModeHook](#) (void)
Callback that gets called whenever the USB host requests the device to exit low power mode.
- void [NvmInitHook](#) (void)
Callback that gets called at the start of the internal NVM driver initialization routine.
- void [NvmReinitHook](#) (void)
Callback that gets called at the start of a firmware update to reinitialize the NVM driver.
- [blt_int8u](#) [NvmWriteHook](#) ([blt_addr](#) addr, [blt_int32u](#) len, [blt_int8u](#) *data)
Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the data hasn't been written yet.
- [blt_int8u](#) [NvmEraseHook](#) ([blt_addr](#) addr, [blt_int32u](#) len)
Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the memory hasn't been erased yet.
- [blt_bool](#) [NvmDoneHook](#) (void)
Callback that gets called at the end of the NVM programming session.

- [blt_int8u XcpGetSeedHook](#) ([blt_int8u](#) resource, [blt_int8u](#) *seed)

Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the GET_SEED command.

- [blt_int8u XcpVerifyKeyHook](#) ([blt_int8u](#) resource, [blt_int8u](#) *key, [blt_int8u](#) len)

Called by the UNLOCK command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

7.605.1 Detailed Description

Bootloader callback source file.

7.605.2 Function Documentation

7.605.2.1 BackDoorEntryHook()

```
blt_bool BackDoorEntryHook (  
    void )
```

Checks if a backdoor entry is requested.

Returns

BLT_TRUE if the backdoor entry is requested, BLT_FALSE otherwise.

7.605.2.2 BackDoorInitHook()

```
void BackDoorInitHook (  
    void )
```

Initializes the backdoor entry option.

Returns

none.

7.605.2.3 CopInitHook()

```
void CopInitHook (
    void )
```

Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.

Returns

none.

7.605.2.4 CopServiceHook()

```
void CopServiceHook (
    void )
```

Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.

Returns

none.

7.605.2.5 CpuUserProgramStartHook()

```
blt_bool CpuUserProgramStartHook (
    void )
```

Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.

Returns

BLT_TRUE if it is okay to start the user program, BLT_FALSE to keep keep the bootloader active.

7.605.2.6 NvmDoneHook()

```
blt_bool NvmDoneHook (
    void )
```

Callback that gets called at the end of the NVM programming session.

Returns

BLT_TRUE is successful, BLT_FALSE otherwise.

7.605.2.7 NvmEraseHook()

```
blt_int8u NvmEraseHook (
    blt_addr addr,
    blt_int32u len )
```

Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the memory hasn't been erased yet.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.

Returns

BLT_NVM_OKAY if successful, BLT_NVM_NOT_IN_RANGE if the address is not within the supported memory range, or BLT_NVM_ERROR if the erase operation failed.

7.605.2.8 NvmInitHook()

```
void NvmInitHook (
    void )
```

Callback that gets called at the start of the internal NVM driver initialization routine.

Returns

none.

7.605.2.9 NvmReinitHook()

```
void NvmReinitHook (
    void )
```

Callback that gets called at the start of a firmware update to reinitialize the NVM driver.

Returns

none.

7.605.2.10 NvmWriteHook()

```
blt_int8u NvmWriteHook (
    blt_addr addr,
    blt_int32u len,
    blt_int8u * data )
```

Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the data hasn't been written yet.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.
<i>data</i>	Pointer to the data buffer.

Returns

BLT_NVM_OKAY if successful, BLT_NVM_NOT_IN_RANGE if the address is not within the supported memory range, or BLT_NVM_ERROR if the write operation failed.

7.605.2.11 UsbConnectHook()

```
void UsbConnectHook (
    blt_bool connect )
```

Callback that gets called whenever the USB device should be connected to the USB bus.

Parameters

<i>connect</i>	BLT_TRUE to connect and BLT_FALSE to disconnect.
----------------	--

Returns

none.

7.605.2.12 UsbEnterLowPowerModeHook()

```
void UsbEnterLowPowerModeHook (
    void )
```

Callback that gets called whenever the USB host requests the device to enter a low power mode.

Returns

none.

7.605.2.13 UsbLeaveLowPowerModeHook()

```
void UsbLeaveLowPowerModeHook (
    void )
```

Callback that gets called whenever the USB host requests the device to exit low power mode.

Returns

none.

7.605.2.14 XcpGetSeedHook()

```
blt_int8u XcpGetSeedHook (
    blt_int8u resource,
    blt_int8u * seed )
```

Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the GET_SEED command.

Parameters

<i>resource</i>	Resource that the seed if requested for (XCP_RES_XXX).
<i>seed</i>	Pointer to byte buffer wher the seed will be stored.

Returns

Length of the seed in bytes.

7.605.2.15 XcpVerifyKeyHook()

```
blt_int8u XcpVerifyKeyHook (
    blt_int8u resource,
    blt_int8u * key,
    blt_int8u len )
```

Called by the UNLOCK command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

Parameters

<i>resource</i>	resource to unlock (XCP_RES_XXX).
<i>key</i>	pointer to the byte buffer holding the key.
<i>len</i>	length of the key in bytes.

Returns

1 if the key was correct, 0 otherwise.

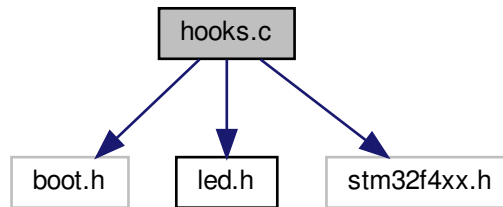
7.606 hooks.c File Reference

Bootloader callback source file.

```
#include "boot.h"
#include "led.h"
```

```
#include "stm32f4xx.h"
```

Include dependency graph for ARMCM4_STM32F4_Nucleo_F429ZI_Keil/Boot/hooks.c:



Functions

- void [BackDoorInitHook](#) (void)
Initializes the backdoor entry option.
- [blt_bool BackDoorEntryHook](#) (void)
Checks if a backdoor entry is requested.
- [blt_bool CpuUserProgramStartHook](#) (void)
Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.
- void [CopInitHook](#) (void)
Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.
- void [CopServiceHook](#) (void)
Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.
- void [UsbConnectHook](#) ([blt_bool](#) connect)
Callback that gets called whenever the USB device should be connected to the USB bus.
- void [UsbEnterLowPowerModeHook](#) (void)
Callback that gets called whenever the USB host requests the device to enter a low power mode.
- void [UsbLeaveLowPowerModeHook](#) (void)
Callback that gets called whenever the USB host requests the device to exit low power mode.
- void [NvmInitHook](#) (void)
Callback that gets called at the start of the internal NVM driver initialization routine.
- void [NvmReinitHook](#) (void)
Callback that gets called at the start of a firmware update to reinitialize the NVM driver.
- [blt_int8u NvmWriteHook](#) ([blt_addr](#) addr, [blt_int32u](#) len, [blt_int8u](#) *data)
Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the data hasn't been written yet.
- [blt_int8u NvmEraseHook](#) ([blt_addr](#) addr, [blt_int32u](#) len)
Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the memory hasn't been erased yet.
- [blt_bool NvmDoneHook](#) (void)
Callback that gets called at the end of the NVM programming session.

- [blt_int8u XcpGetSeedHook](#) ([blt_int8u](#) resource, [blt_int8u](#) *seed)

Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the GET_SEED command.

- [blt_int8u XcpVerifyKeyHook](#) ([blt_int8u](#) resource, [blt_int8u](#) *key, [blt_int8u](#) len)

Called by the UNLOCK command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

7.606.1 Detailed Description

Bootloader callback source file.

7.606.2 Function Documentation

7.606.2.1 BackDoorEntryHook()

```
blt_bool BackDoorEntryHook (  
    void )
```

Checks if a backdoor entry is requested.

Returns

BLT_TRUE if the backdoor entry is requested, BLT_FALSE otherwise.

7.606.2.2 BackDoorInitHook()

```
void BackDoorInitHook (  
    void )
```

Initializes the backdoor entry option.

Returns

none.

7.606.2.3 CopInitHook()

```
void CopInitHook (  
    void )
```

Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.

Returns

none.

7.606.2.4 CopServiceHook()

```
void CopServiceHook (  
    void )
```

Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.

Returns

none.

7.606.2.5 CpuUserProgramStartHook()

```
blt_bool CpuUserProgramStartHook (  
    void )
```

Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.

Returns

BLT_TRUE if it is okay to start the user program, BLT_FALSE to keep keep the bootloader active.

7.606.2.6 NvmDoneHook()

```
blt_bool NvmDoneHook (  
    void )
```

Callback that gets called at the end of the NVM programming session.

Returns

BLT_TRUE is successful, BLT_FALSE otherwise.

7.606.2.7 NvmEraseHook()

```
blt_int8u NvmEraseHook (
    blt_addr addr,
    blt_int32u len )
```

Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the memory hasn't been erased yet.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.

Returns

BLT_NVM_OKAY if successful, BLT_NVM_NOT_IN_RANGE if the address is not within the supported memory range, or BLT_NVM_ERROR if the erase operation failed.

7.606.2.8 NvmInitHook()

```
void NvmInitHook (
    void )
```

Callback that gets called at the start of the internal NVM driver initialization routine.

Returns

none.

7.606.2.9 NvmReinitHook()

```
void NvmReinitHook (
    void )
```

Callback that gets called at the start of a firmware update to reinitialize the NVM driver.

Returns

none.

7.606.2.10 NvmWriteHook()

```
blt_int8u NvmWriteHook (
    blt_addr addr,
    blt_int32u len,
    blt_int8u * data )
```

Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the data hasn't been written yet.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.
<i>data</i>	Pointer to the data buffer.

Returns

BLT_NVM_OKAY if successful, BLT_NVM_NOT_IN_RANGE if the address is not within the supported memory range, or BLT_NVM_ERROR if the write operation failed.

7.606.2.11 UsbConnectHook()

```
void UsbConnectHook (
    blt_bool connect )
```

Callback that gets called whenever the USB device should be connected to the USB bus.

Parameters

<i>connect</i>	BLT_TRUE to connect and BLT_FALSE to disconnect.
----------------	--

Returns

none.

7.606.2.12 UsbEnterLowPowerModeHook()

```
void UsbEnterLowPowerModeHook (
    void )
```

Callback that gets called whenever the USB host requests the device to enter a low power mode.

Returns

none.

7.606.2.13 UsbLeaveLowPowerModeHook()

```
void UsbLeaveLowPowerModeHook (
    void )
```

Callback that gets called whenever the USB host requests the device to exit low power mode.

Returns

none.

7.606.2.14 XcpGetSeedHook()

```
blt_int8u XcpGetSeedHook (
    blt_int8u resource,
    blt_int8u * seed )
```

Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the GET_SEED command.

Parameters

<i>resource</i>	Resource that the seed is requested for (XCP_RES_XXX).
<i>seed</i>	Pointer to byte buffer where the seed will be stored.

Returns

Length of the seed in bytes.

7.606.2.15 XcpVerifyKeyHook()

```
blt_int8u XcpVerifyKeyHook (
    blt_int8u resource,
    blt_int8u * key,
    blt_int8u len )
```

Called by the UNLOCK command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

Parameters

<i>resource</i>	resource to unlock (XCP_RES_XXX).
<i>key</i>	pointer to the byte buffer holding the key.
<i>len</i>	length of the key in bytes.

Returns

1 if the key was correct, 0 otherwise.

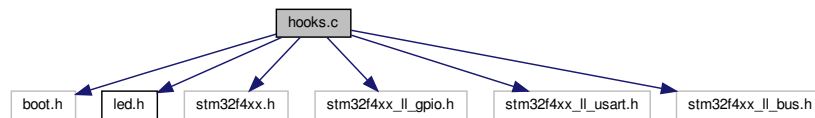
7.607 hooks.c File Reference

Bootloader callback source file.

```
#include "boot.h"
#include "led.h"
#include "stm32f4xx.h"
#include "stm32f4xx_ll_gpio.h"
#include "stm32f4xx_ll_usart.h"
```

```
#include "stm32f4xx_ll_bus.h"
```

Include dependency graph for ARMC4_STM32F4_Olimex_STM32P405_CubeIDE/Boot/App/hooks.c:



Functions

- void [BackDoorInitHook](#) (void)
Initializes the backdoor entry option.
- [blt_bool](#) [BackDoorEntryHook](#) (void)
Checks if a backdoor entry is requested.
- [blt_bool](#) [CpuUserProgramStartHook](#) (void)
Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.
- void [CopInitHook](#) (void)
Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.
- void [CopServiceHook](#) (void)
Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.
- void [NvmInitHook](#) (void)
Callback that gets called at the start of the internal NVM driver initialization routine.
- void [NvmReinitHook](#) (void)
Callback that gets called at the start of a firmware update to reinitialize the NVM driver.
- [blt_int8u](#) [NvmWriteHook](#) ([blt_addr](#) addr, [blt_int32u](#) len, [blt_int8u](#) *data)
Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the data hasn't been written yet.
- [blt_int8u](#) [NvmEraseHook](#) ([blt_addr](#) addr, [blt_int32u](#) len)
Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the memory hasn't been erased yet.
- [blt_bool](#) [NvmDoneHook](#) (void)
Callback that gets called at the end of the NVM programming session.
- void [UsbConnectHook](#) ([blt_bool](#) connect)
Callback that gets called whenever the USB device should be connected to the USB bus.
- void [UsbEnterLowPowerModeHook](#) (void)
Callback that gets called whenever the USB host requests the device to enter a low power mode.
- void [UsbLeaveLowPowerModeHook](#) (void)
Callback that gets called whenever the USB host requests the device to exit low power mode.
- [blt_bool](#) [FileIsFirmwareUpdateRequestedHook](#) (void)
Callback that gets called to check whether a firmware update from local file storage should be started. This could for example be when a switch is pressed, when a certain file is found on the local file storage, etc.
- const [blt_char](#) * [FileGetFirmwareFilenameHook](#) (void)

Callback to obtain the filename of the firmware file that should be used during the firmware update from the local file storage. This hook function is called at the beginning of the firmware update from local storage sequence.

- void [FileFirmwareUpdateStartedHook](#) (void)

Callback that gets called to inform the application that a firmware update from local storage just started.

- void [FileFirmwareUpdateCompletedHook](#) (void)

Callback that gets called to inform the application that a firmware update was successfully completed.

- void [FileFirmwareUpdateErrorHook](#) (blt_int8u error_code)

Callback that gets called in case an error occurred during a firmware update. Refer to [<file.h>](#) for a list of available error codes.

- void [FileFirmwareUpdateLogHook](#) (blt_char *info_string)

Callback that gets called each time new log information becomes available during a firmware update.

- [blt_int8u XcpGetSeedHook](#) (blt_int8u resource, [blt_int8u](#) *seed)

Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the GET_SEED command.

- [blt_int8u XcpVerifyKeyHook](#) (blt_int8u resource, [blt_int8u](#) *key, [blt_int8u](#) len)

Called by the UNLOCK command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

Variables

- static const [blt_char firmwareFilename](#) [] = "/demoprogram_olimex_stm32p405.srec"

Firmware filename.

-

```
struct {
    FIL handle
    blt_bool canUse
} logfile
```

Data structure for grouping log-file related information.

7.607.1 Detailed Description

Bootloader callback source file.

7.607.2 Function Documentation

7.607.2.1 BackDoorEntryHook()

```
blt_bool BackDoorEntryHook (
    void )
```

Checks if a backdoor entry is requested.

Returns

BLT_TRUE if the backdoor entry is requested, BLT_FALSE otherwise.

7.607.2.2 BackDoorInitHook()

```
void BackDoorInitHook (  
    void )
```

Initializes the backdoor entry option.

Returns

none.

7.607.2.3 CopInitHook()

```
void CopInitHook (  
    void )
```

Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.

Returns

none.

7.607.2.4 CopServiceHook()

```
void CopServiceHook (  
    void )
```

Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.

Returns

none.

7.607.2.5 CpuUserProgramStartHook()

```
blt_bool CpuUserProgramStartHook (  
    void )
```

Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.

Returns

BLT_TRUE if it is okay to start the user program, BLT_FALSE to keep keep the bootloader active.

7.607.2.6 FileFirmwareUpdateCompletedHook()

```
void FileFirmwareUpdateCompletedHook (  
    void )
```

Callback that gets called to inform the application that a firmware update was successfully completed.

Returns

none.

7.607.2.7 FileFirmwareUpdateErrorHook()

```
void FileFirmwareUpdateErrorHook (  
    blt_int8u error_code )
```

Callback that gets called in case an error occurred during a firmware update. Refer to <[file.h](#)> for a list of available error codes.

Returns

none.

7.607.2.8 FileFirmwareUpdateLogHook()

```
void FileFirmwareUpdateLogHook (  
    blt_char * info_string )
```

Callback that gets called each time new log information becomes available during a firmware update.

Parameters

<i>info_string</i>	Pointer to a character array with the log entry info.
--------------------	---

Returns

none.

7.607.2.9 FileFirmwareUpdateStartedHook()

```
void FileFirmwareUpdateStartedHook (  
    void )
```

Callback that gets called to inform the application that a firmware update from local storage just started.

Returns

none.

7.607.2.10 FileGetFirmwareFilenameHook()

```
const blt_char* FileGetFirmwareFilenameHook (
    void )
```

Callback to obtain the filename of the firmware file that should be used during the firmware update from the local file storage. This hook function is called at the beginning of the firmware update from local storage sequence.

Returns

valid firmware filename with full path or BLT_NULL.

7.607.2.11 FileIsFirmwareUpdateRequestedHook()

```
blt_bool FileIsFirmwareUpdateRequestedHook (
    void )
```

Callback that gets called to check whether a firmware update from local file storage should be started. This could for example be when a switch is pressed, when a certain file is found on the local file storage, etc.

Returns

BLT_TRUE if a firmware update is requested, BLT_FALSE otherwise.

7.607.2.12 NvmDoneHook()

```
blt_bool NvmDoneHook (
    void )
```

Callback that gets called at the end of the NVM programming session.

Returns

BLT_TRUE is successful, BLT_FALSE otherwise.

7.607.2.13 NvmEraseHook()

```
blt_int8u NvmEraseHook (
    blt_addr addr,
    blt_int32u len )
```

Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the memory hasn't been erased yet.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.

Returns

BLT_NVM_OKAY if successful, BLT_NVM_NOT_IN_RANGE if the address is not within the supported memory range, or BLT_NVM_ERROR if the erase operation failed.

7.607.2.14 NvmInitHook()

```
void NvmInitHook (  
    void )
```

Callback that gets called at the start of the internal NVM driver initialization routine.

Returns

none.

7.607.2.15 NvmReinitHook()

```
void NvmReinitHook (  
    void )
```

Callback that gets called at the start of a firmware update to reinitialize the NVM driver.

Returns

none.

7.607.2.16 NvmWriteHook()

```
blt_int8u NvmWriteHook (  
    blt_addr addr,  
    blt_int32u len,  
    blt_int8u * data )
```

Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the data hasn't been written yet.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.
<i>data</i>	Pointer to the data buffer.

Returns

BLT_NVM_OKAY if successful, BLT_NVM_NOT_IN_RANGE if the address is not within the supported memory range, or BLT_NVM_ERROR if the write operation failed.

7.607.2.17 UsbConnectHook()

```
void UsbConnectHook (
    blt_bool connect )
```

Callback that gets called whenever the USB device should be connected to the USB bus.

Parameters

<i>connect</i>	BLT_TRUE to connect and BLT_FALSE to disconnect.
----------------	--

Returns

none.

7.607.2.18 UsbEnterLowPowerModeHook()

```
void UsbEnterLowPowerModeHook (
    void )
```

Callback that gets called whenever the USB host requests the device to enter a low power mode.

Returns

none.

7.607.2.19 UsbLeaveLowPowerModeHook()

```
void UsbLeaveLowPowerModeHook (
    void )
```

Callback that gets called whenever the USB host requests the device to exit low power mode.

Returns

none.

7.607.2.20 XcpGetSeedHook()

```
blt_int8u XcpGetSeedHook (
    blt_int8u resource,
    blt_int8u * seed )
```

Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the GET_SEED command.

Parameters

<i>resource</i>	Resource that the seed is requested for (XCP_RES_XXX).
<i>seed</i>	Pointer to byte buffer where the seed will be stored.

Returns

Length of the seed in bytes.

7.607.2.21 XcpVerifyKeyHook()

```
blt_int8u XcpVerifyKeyHook (
    blt_int8u resource,
    blt_int8u * key,
    blt_int8u len )
```

Called by the UNLOCK command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

Parameters

<i>resource</i>	resource to unlock (XCP_RES_XXX).
<i>key</i>	pointer to the byte buffer holding the key.
<i>len</i>	length of the key in bytes.

Returns

1 if the key was correct, 0 otherwise.

7.607.3 Variable Documentation

7.607.3.1 canUse

```
blt_bool canUse
```

Flag to indicate if the log-file can be used.

7.607.3.2 handle

FIL handle

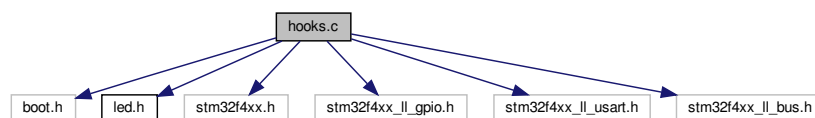
FatFS handle to the log-file.

7.608 hooks.c File Reference

Bootloader callback source file.

```
#include "boot.h"
#include "led.h"
#include "stm32f4xx.h"
#include "stm32f4xx_ll_gpio.h"
#include "stm32f4xx_ll_usart.h"
#include "stm32f4xx_ll_bus.h"
```

Include dependency graph for ARMC4_M4_STM32F4_Olimex_STM32P405_GCC/Boot/hooks.c:



Functions

- void [BackDoorInitHook](#) (void)
Initializes the backdoor entry option.
- [blt_bool](#) [BackDoorEntryHook](#) (void)
Checks if a backdoor entry is requested.
- [blt_bool](#) [CpuUserProgramStartHook](#) (void)
Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.
- void [CopInitHook](#) (void)
Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.
- void [CopServiceHook](#) (void)
Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.
- void [NvmInitHook](#) (void)
Callback that gets called at the start of the internal NVM driver initialization routine.
- void [NvmReinitHook](#) (void)
Callback that gets called at the start of a firmware update to reinitialize the NVM driver.
- [blt_int8u](#) [NvmWriteHook](#) ([blt_addr](#) addr, [blt_int32u](#) len, [blt_int8u](#) *data)
Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the data hasn't been written yet.
- [blt_int8u](#) [NvmEraseHook](#) ([blt_addr](#) addr, [blt_int32u](#) len)

Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then `BLT_NVM_NOT_IN_RANGE` must be returned to indicate that the memory hasn't been erased yet.

- `blt_bool NvmDoneHook` (void)

Callback that gets called at the end of the NVM programming session.

- void `UsbConnectHook` (`blt_bool` connect)

Callback that gets called whenever the USB device should be connected to the USB bus.

- void `UsbEnterLowPowerModeHook` (void)

Callback that gets called whenever the USB host requests the device to enter a low power mode.

- void `UsbLeaveLowPowerModeHook` (void)

Callback that gets called whenever the USB host requests the device to exit low power mode.

- `blt_bool` `FileIsFirmwareUpdateRequestedHook` (void)

Callback that gets called to check whether a firmware update from local file storage should be started. This could for example be when a switch is pressed, when a certain file is found on the local file storage, etc.

- const `blt_char` * `FileGetFirmwareFilenameHook` (void)

Callback to obtain the filename of the firmware file that should be used during the firmware update from the local file storage. This hook function is called at the beginning of the firmware update from local storage sequence.

- void `FileFirmwareUpdateStartedHook` (void)

Callback that gets called to inform the application that a firmware update from local storage just started.

- void `FileFirmwareUpdateCompletedHook` (void)

Callback that gets called to inform the application that a firmware update was successfully completed.

- void `FileFirmwareUpdateErrorHook` (`blt_int8u` error_code)

Callback that gets called in case an error occurred during a firmware update. Refer to [<file.h>](#) for a list of available error codes.

- void `FileFirmwareUpdateLogHook` (`blt_char` *info_string)

Callback that gets called each time new log information becomes available during a firmware update.

- `blt_int8u` `XcpGetSeedHook` (`blt_int8u` resource, `blt_int8u` *seed)

Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the `GET_SEED` command.

- `blt_int8u` `XcpVerifyKeyHook` (`blt_int8u` resource, `blt_int8u` *key, `blt_int8u` len)

Called by the `UNLOCK` command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

Variables

- static const `blt_char` `firmwareFilename` [] = `"/demoprogram_olimex_stm32p405.srec"`

Firmware filename.

-

```
struct {
    FIL handle
    blt_bool canUse
} logfile
```

Data structure for grouping log-file related information.

7.608.1 Detailed Description

Bootloader callback source file.

7.608.2 Function Documentation

7.608.2.1 BackDoorEntryHook()

```
blt_bool BackDoorEntryHook (  
    void )
```

Checks if a backdoor entry is requested.

Returns

BLT_TRUE if the backdoor entry is requested, BLT_FALSE otherwise.

7.608.2.2 BackDoorInitHook()

```
void BackDoorInitHook (  
    void )
```

Initializes the backdoor entry option.

Returns

none.

7.608.2.3 CopInitHook()

```
void CopInitHook (  
    void )
```

Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.

Returns

none.

7.608.2.4 CopServiceHook()

```
void CopServiceHook (
    void )
```

Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.

Returns

none.

7.608.2.5 CpuUserProgramStartHook()

```
blt_bool CpuUserProgramStartHook (
    void )
```

Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.

Returns

BLT_TRUE if it is okay to start the user program, BLT_FALSE to keep keep the bootloader active.

7.608.2.6 FileFirmwareUpdateCompletedHook()

```
void FileFirmwareUpdateCompletedHook (
    void )
```

Callback that gets called to inform the application that a firmware update was successfully completed.

Returns

none.

7.608.2.7 FileFirmwareUpdateErrorHook()

```
void FileFirmwareUpdateErrorHook (
    blt_int8u error_code )
```

Callback that gets called in case an error occurred during a firmware update. Refer to <[file.h](#)> for a list of available error codes.

Returns

none.

7.608.2.8 FileFirmwareUpdateLogHook()

```
void FileFirmwareUpdateLogHook (
    blt_char * info_string )
```

Callback that gets called each time new log information becomes available during a firmware update.

Parameters

<i>info_string</i>	Pointer to a character array with the log entry info.
--------------------	---

Returns

none.

7.608.2.9 FileFirmwareUpdateStartedHook()

```
void FileFirmwareUpdateStartedHook (  
    void )
```

Callback that gets called to inform the application that a firmware update from local storage just started.

Returns

none.

7.608.2.10 FileGetFirmwareFilenameHook()

```
const blt_char* FileGetFirmwareFilenameHook (  
    void )
```

Callback to obtain the filename of the firmware file that should be used during the firmware update from the local file storage. This hook function is called at the beginning of the firmware update from local storage sequence.

Returns

valid firmware filename with full path or BLT_NULL.

7.608.2.11 FileIsFirmwareUpdateRequestedHook()

```
blt_bool FileIsFirmwareUpdateRequestedHook (  
    void )
```

Callback that gets called to check whether a firmware update from local file storage should be started. This could for example be when a switch is pressed, when a certain file is found on the local file storage, etc.

Returns

BLT_TRUE if a firmware update is requested, BLT_FALSE otherwise.

7.608.2.12 NvmDoneHook()

```
blt_bool NvmDoneHook (
    void )
```

Callback that gets called at the end of the NVM programming session.

Returns

BLT_TRUE is successful, BLT_FALSE otherwise.

7.608.2.13 NvmEraseHook()

```
blt_int8u NvmEraseHook (
    blt_addr addr,
    blt_int32u len )
```

Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the memory hasn't been erased yet.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.

Returns

BLT_NVM_OKAY if successful, BLT_NVM_NOT_IN_RANGE if the address is not within the supported memory range, or BLT_NVM_ERROR if the erase operation failed.

7.608.2.14 NvmInitHook()

```
void NvmInitHook (
    void )
```

Callback that gets called at the start of the internal NVM driver initialization routine.

Returns

none.

7.608.2.15 NvmReinitHook()

```
void NvmReinitHook (  
    void )
```

Callback that gets called at the start of a firmware update to reinitialize the NVM driver.

Returns

none.

7.608.2.16 NvmWriteHook()

```
blt_int8u NvmWriteHook (  
    blt_addr addr,  
    blt_int32u len,  
    blt_int8u * data )
```

Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the data hasn't been written yet.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.
<i>data</i>	Pointer to the data buffer.

Returns

BLT_NVM_OKAY if successful, BLT_NVM_NOT_IN_RANGE if the address is not within the supported memory range, or BLT_NVM_ERROR if the write operation failed.

7.608.2.17 UsbConnectHook()

```
void UsbConnectHook (  
    blt_bool connect )
```

Callback that gets called whenever the USB device should be connected to the USB bus.

Parameters

<i>connect</i>	BLT_TRUE to connect and BLT_FALSE to disconnect.
----------------	--

Returns

none.

7.608.2.18 UsbEnterLowPowerModeHook()

```
void UsbEnterLowPowerModeHook (  
    void )
```

Callback that gets called whenever the USB host requests the device to enter a low power mode.

Returns

none.

7.608.2.19 UsbLeaveLowPowerModeHook()

```
void UsbLeaveLowPowerModeHook (  
    void )
```

Callback that gets called whenever the USB host requests the device to exit low power mode.

Returns

none.

7.608.2.20 XcpGetSeedHook()

```
blt_int8u XcpGetSeedHook (  
    blt_int8u resource,  
    blt_int8u * seed )
```

Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the GET_SEED command.

Parameters

<i>resource</i>	Resource that the seed is requested for (XCP_RES_XXX).
<i>seed</i>	Pointer to byte buffer where the seed will be stored.

Returns

Length of the seed in bytes.

7.608.2.21 XcpVerifyKeyHook()

```
blt_int8u XcpVerifyKeyHook (
    blt_int8u resource,
    blt_int8u * key,
    blt_int8u len )
```

Called by the UNLOCK command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

Parameters

<i>resource</i>	resource to unlock (XCP_RES_XXX).
<i>key</i>	pointer to the byte buffer holding the key.
<i>len</i>	length of the key in bytes.

Returns

1 if the key was correct, 0 otherwise.

7.608.3 Variable Documentation**7.608.3.1 canUse**

```
blt_bool canUse
```

Flag to indicate if the log-file can be used.

7.608.3.2 handle

```
FIL handle
```

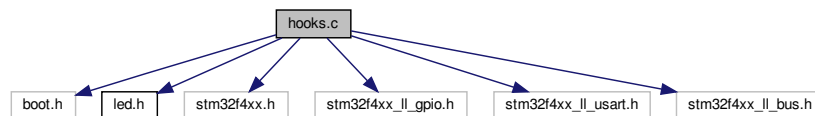
FatFS handle to the log-file.

7.609 hooks.c File Reference

Bootloader callback source file.

```
#include "boot.h"
#include "led.h"
#include "stm32f4xx.h"
#include "stm32f4xx_ll_gpio.h"
#include "stm32f4xx_ll_usart.h"
#include "stm32f4xx_ll_bus.h"
```

Include dependency graph for ARMC4_STM32F4_Olimex_STM32P405_IAR/Boot/hooks.c:



Functions

- void [BackDoorInitHook](#) (void)
Initializes the backdoor entry option.
- [blt_bool](#) [BackDoorEntryHook](#) (void)
Checks if a backdoor entry is requested.
- [blt_bool](#) [CpuUserProgramStartHook](#) (void)
Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.
- void [CopInitHook](#) (void)
Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.
- void [CopServiceHook](#) (void)
Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.
- void [NvmInitHook](#) (void)
Callback that gets called at the start of the internal NVM driver initialization routine.
- void [NvmReinitHook](#) (void)
Callback that gets called at the start of a firmware update to reinitialize the NVM driver.
- [blt_int8u](#) [NvmWriteHook](#) ([blt_addr](#) addr, [blt_int32u](#) len, [blt_int8u](#) *data)
Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the data hasn't been written yet.
- [blt_int8u](#) [NvmEraseHook](#) ([blt_addr](#) addr, [blt_int32u](#) len)
Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the memory hasn't been erased yet.
- [blt_bool](#) [NvmDoneHook](#) (void)
Callback that gets called at the end of the NVM programming session.
- void [UsbConnectHook](#) ([blt_bool](#) connect)
Callback that gets called whenever the USB device should be connected to the USB bus.

- void [UsbEnterLowPowerModeHook](#) (void)
Callback that gets called whenever the USB host requests the device to enter a low power mode.
- void [UsbLeaveLowPowerModeHook](#) (void)
Callback that gets called whenever the USB host requests the device to exit low power mode.
- [blt_bool](#) [FileIsFirmwareUpdateRequestedHook](#) (void)
Callback that gets called to check whether a firmware update from local file storage should be started. This could for example be when a switch is pressed, when a certain file is found on the local file storage, etc.
- const [blt_char](#) * [FileGetFirmwareFilenameHook](#) (void)
Callback to obtain the filename of the firmware file that should be used during the firmware update from the local file storage. This hook function is called at the beginning of the firmware update from local storage sequence.
- void [FileFirmwareUpdateStartedHook](#) (void)
Callback that gets called to inform the application that a firmware update from local storage just started.
- void [FileFirmwareUpdateCompletedHook](#) (void)
Callback that gets called to inform the application that a firmware update was successfully completed.
- void [FileFirmwareUpdateErrorHook](#) ([blt_int8u](#) error_code)
Callback that gets called in case an error occurred during a firmware update. Refer to <[file.h](#)> for a list of available error codes.
- void [FileFirmwareUpdateLogHook](#) ([blt_char](#) *info_string)
Callback that gets called each time new log information becomes available during a firmware update.
- [blt_int8u](#) [XcpGetSeedHook](#) ([blt_int8u](#) resource, [blt_int8u](#) *seed)
Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the GET_SEED command.
- [blt_int8u](#) [XcpVerifyKeyHook](#) ([blt_int8u](#) resource, [blt_int8u](#) *key, [blt_int8u](#) len)
Called by the UNLOCK command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

Variables

- static const [blt_char](#) [firmwareFilename](#) [] = "/demoprogram_olimex_stm32p405.srec"
Firmware filename.
- ```

struct {
 FIL handle
 blt_bool canUse
} logfile

```

*Data structure for grouping log-file related information.*

## 7.609.1 Detailed Description

Bootloader callback source file.

## 7.609.2 Function Documentation

### 7.609.2.1 BackDoorEntryHook()

```
blt_bool BackDoorEntryHook (
 void)
```

Checks if a backdoor entry is requested.

#### Returns

BLT\_TRUE if the backdoor entry is requested, BLT\_FALSE otherwise.

### 7.609.2.2 BackDoorInitHook()

```
void BackDoorInitHook (
 void)
```

Initializes the backdoor entry option.

#### Returns

none.

### 7.609.2.3 CopInitHook()

```
void CopInitHook (
 void)
```

Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.

#### Returns

none.

### 7.609.2.4 CopServiceHook()

```
void CopServiceHook (
 void)
```

Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.

#### Returns

none.

#### 7.609.2.5 CpuUserProgramStartHook()

```
blt_bool CpuUserProgramStartHook (
 void)
```

Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.

##### Returns

BLT\_TRUE if it is okay to start the user program, BLT\_FALSE to keep keep the bootloader active.

#### 7.609.2.6 FileFirmwareUpdateCompletedHook()

```
void FileFirmwareUpdateCompletedHook (
 void)
```

Callback that gets called to inform the application that a firmware update was successfully completed.

##### Returns

none.

#### 7.609.2.7 FileFirmwareUpdateErrorHook()

```
void FileFirmwareUpdateErrorHook (
 blt_int8u error_code)
```

Callback that gets called in case an error occurred during a firmware update. Refer to <[file.h](#)> for a list of available error codes.

##### Returns

none.

#### 7.609.2.8 FileFirmwareUpdateLogHook()

```
void FileFirmwareUpdateLogHook (
 blt_char * info_string)
```

Callback that gets called each time new log information becomes available during a firmware update.

**Parameters**

|                    |                                                       |
|--------------------|-------------------------------------------------------|
| <i>info_string</i> | Pointer to a character array with the log entry info. |
|--------------------|-------------------------------------------------------|

**Returns**

none.

**7.609.2.9 FileFirmwareUpdateStartedHook()**

```
void FileFirmwareUpdateStartedHook (
 void)
```

Callback that gets called to inform the application that a firmware update from local storage just started.

**Returns**

none.

**7.609.2.10 FileGetFirmwareFilenameHook()**

```
const blt_char* FileGetFirmwareFilenameHook (
 void)
```

Callback to obtain the filename of the firmware file that should be used during the firmware update from the local file storage. This hook function is called at the beginning of the firmware update from local storage sequence.

**Returns**

valid firmware filename with full path or BLT\_NULL.

**7.609.2.11 FileIsFirmwareUpdateRequestedHook()**

```
blt_bool FileIsFirmwareUpdateRequestedHook (
 void)
```

Callback that gets called to check whether a firmware update from local file storage should be started. This could for example be when a switch is pressed, when a certain file is found on the local file storage, etc.

**Returns**

BLT\_TRUE if a firmware update is requested, BLT\_FALSE otherwise.

#### 7.609.2.12 NvmDoneHook()

```
blt_bool NvmDoneHook (
 void)
```

Callback that gets called at the end of the NVM programming session.

##### Returns

BLT\_TRUE is successful, BLT\_FALSE otherwise.

#### 7.609.2.13 NvmEraseHook()

```
blt_int8u NvmEraseHook (
 blt_addr addr,
 blt_int32u len)
```

Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the memory hasn't been erased yet.

##### Parameters

|             |                  |
|-------------|------------------|
| <i>addr</i> | Start address.   |
| <i>len</i>  | Length in bytes. |

##### Returns

BLT\_NVM\_OKAY if successful, BLT\_NVM\_NOT\_IN\_RANGE if the address is not within the supported memory range, or BLT\_NVM\_ERROR if the erase operation failed.

#### 7.609.2.14 NvmInitHook()

```
void NvmInitHook (
 void)
```

Callback that gets called at the start of the internal NVM driver initialization routine.

##### Returns

none.



### 7.609.2.15 NvmReinitHook()

```
void NvmReinitHook (
 void)
```

Callback that gets called at the start of a firmware update to reinitialize the NVM driver.

#### Returns

none.

### 7.609.2.16 NvmWriteHook()

```
blt_int8u NvmWriteHook (
 blt_addr addr,
 blt_int32u len,
 blt_int8u * data)
```

Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the data hasn't been written yet.

#### Parameters

|             |                             |
|-------------|-----------------------------|
| <i>addr</i> | Start address.              |
| <i>len</i>  | Length in bytes.            |
| <i>data</i> | Pointer to the data buffer. |

#### Returns

BLT\_NVM\_OKAY if successful, BLT\_NVM\_NOT\_IN\_RANGE if the address is not within the supported memory range, or BLT\_NVM\_ERROR if the write operation failed.

### 7.609.2.17 UsbConnectHook()

```
void UsbConnectHook (
 blt_bool connect)
```

Callback that gets called whenever the USB device should be connected to the USB bus.

#### Parameters

|                |                                                  |
|----------------|--------------------------------------------------|
| <i>connect</i> | BLT_TRUE to connect and BLT_FALSE to disconnect. |
|----------------|--------------------------------------------------|

**Returns**

none.

**7.609.2.18 UsbEnterLowPowerModeHook()**

```
void UsbEnterLowPowerModeHook (
 void)
```

Callback that gets called whenever the USB host requests the device to enter a low power mode.

**Returns**

none.

**7.609.2.19 UsbLeaveLowPowerModeHook()**

```
void UsbLeaveLowPowerModeHook (
 void)
```

Callback that gets called whenever the USB host requests the device to exit low power mode.

**Returns**

none.

**7.609.2.20 XcpGetSeedHook()**

```
blt_int8u XcpGetSeedHook (
 blt_int8u resource,
 blt_int8u * seed)
```

Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the GET\_SEED command.

**Parameters**

|                 |                                                        |
|-----------------|--------------------------------------------------------|
| <i>resource</i> | Resource that the seed is requested for (XCP_RES_XXX). |
| <i>seed</i>     | Pointer to byte buffer where the seed will be stored.  |

**Returns**

Length of the seed in bytes.

**7.609.2.21 XcpVerifyKeyHook()**

```
blt_int8u XcpVerifyKeyHook (
 blt_int8u resource,
 blt_int8u * key,
 blt_int8u len)
```

Called by the UNLOCK command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

**Parameters**

|                 |                                             |
|-----------------|---------------------------------------------|
| <i>resource</i> | resource to unlock (XCP_RES_XXX).           |
| <i>key</i>      | pointer to the byte buffer holding the key. |
| <i>len</i>      | length of the key in bytes.                 |

**Returns**

1 if the key was correct, 0 otherwise.

**7.609.3 Variable Documentation****7.609.3.1 canUse**

```
blt_bool canUse
```

Flag to indicate if the log-file can be used.

**7.609.3.2 handle**

```
FIL handle
```

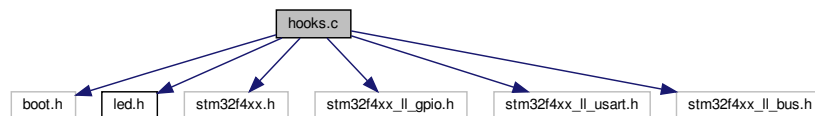
FatFS handle to the log-file.

## 7.610 hooks.c File Reference

Bootloader callback source file.

```
#include "boot.h"
#include "led.h"
#include "stm32f4xx.h"
#include "stm32f4xx_ll_gpio.h"
#include "stm32f4xx_ll_usart.h"
#include "stm32f4xx_ll_bus.h"
```

Include dependency graph for ARMC4\_STM32F4\_Olimex\_STM32P405\_Keil/Boot/hooks.c:



## Functions

- void [BackDoorInitHook](#) (void)  
*Initializes the backdoor entry option.*
- [blt\\_bool](#) [BackDoorEntryHook](#) (void)  
*Checks if a backdoor entry is requested.*
- [blt\\_bool](#) [CpuUserProgramStartHook](#) (void)  
*Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.*
- void [CopInitHook](#) (void)  
*Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.*
- void [CopServiceHook](#) (void)  
*Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.*
- void [NvmInitHook](#) (void)  
*Callback that gets called at the start of the internal NVM driver initialization routine.*
- void [NvmReinitHook](#) (void)  
*Callback that gets called at the start of a firmware update to reinitialize the NVM driver.*
- [blt\\_int8u](#) [NvmWriteHook](#) ([blt\\_addr](#) addr, [blt\\_int32u](#) len, [blt\\_int8u](#) \*data)  
*Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the data hasn't been written yet.*
- [blt\\_int8u](#) [NvmEraseHook](#) ([blt\\_addr](#) addr, [blt\\_int32u](#) len)  
*Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the memory hasn't been erased yet.*
- [blt\\_bool](#) [NvmDoneHook](#) (void)  
*Callback that gets called at the end of the NVM programming session.*
- void [UsbConnectHook](#) ([blt\\_bool](#) connect)  
*Callback that gets called whenever the USB device should be connected to the USB bus.*

- void [UsbEnterLowPowerModeHook](#) (void)  
*Callback that gets called whenever the USB host requests the device to enter a low power mode.*
- void [UsbLeaveLowPowerModeHook](#) (void)  
*Callback that gets called whenever the USB host requests the device to exit low power mode.*
- [blt\\_bool](#) [FileIsFirmwareUpdateRequestedHook](#) (void)  
*Callback that gets called to check whether a firmware update from local file storage should be started. This could for example be when a switch is pressed, when a certain file is found on the local file storage, etc.*
- const [blt\\_char](#) \* [FileGetFirmwareFilenameHook](#) (void)  
*Callback to obtain the filename of the firmware file that should be used during the firmware update from the local file storage. This hook function is called at the beginning of the firmware update from local storage sequence.*
- void [FileFirmwareUpdateStartedHook](#) (void)  
*Callback that gets called to inform the application that a firmware update from local storage just started.*
- void [FileFirmwareUpdateCompletedHook](#) (void)  
*Callback that gets called to inform the application that a firmware update was successfully completed.*
- void [FileFirmwareUpdateErrorHook](#) ([blt\\_int8u](#) error\_code)  
*Callback that gets called in case an error occurred during a firmware update. Refer to <[file.h](#)> for a list of available error codes.*
- void [FileFirmwareUpdateLogHook](#) ([blt\\_char](#) \*info\_string)  
*Callback that gets called each time new log information becomes available during a firmware update.*
- [blt\\_int8u](#) [XcpGetSeedHook](#) ([blt\\_int8u](#) resource, [blt\\_int8u](#) \*seed)  
*Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the GET\_SEED command.*
- [blt\\_int8u](#) [XcpVerifyKeyHook](#) ([blt\\_int8u](#) resource, [blt\\_int8u](#) \*key, [blt\\_int8u](#) len)  
*Called by the UNLOCK command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.*

## Variables

- static const [blt\\_char](#) [firmwareFilename](#) [ ] = "/demoprogram\_olimex\_stm32p405.srec"  
*Firmware filename.*
- ```

struct {
    FIL handle
    blt\_bool canUse
} logfile

```

Data structure for grouping log-file related information.

7.610.1 Detailed Description

Bootloader callback source file.

7.610.2 Function Documentation

7.610.2.1 BackDoorEntryHook()

```
blt_bool BackDoorEntryHook (  
    void )
```

Checks if a backdoor entry is requested.

Returns

BLT_TRUE if the backdoor entry is requested, BLT_FALSE otherwise.

7.610.2.2 BackDoorInitHook()

```
void BackDoorInitHook (  
    void )
```

Initializes the backdoor entry option.

Returns

none.

7.610.2.3 CopInitHook()

```
void CopInitHook (  
    void )
```

Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.

Returns

none.

7.610.2.4 CopServiceHook()

```
void CopServiceHook (  
    void )
```

Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.

Returns

none.

7.610.2.5 CpuUserProgramStartHook()

```
blt_bool CpuUserProgramStartHook (
    void )
```

Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.

Returns

BLT_TRUE if it is okay to start the user program, BLT_FALSE to keep keep the bootloader active.

7.610.2.6 FileFirmwareUpdateCompletedHook()

```
void FileFirmwareUpdateCompletedHook (
    void )
```

Callback that gets called to inform the application that a firmware update was successfully completed.

Returns

none.

7.610.2.7 FileFirmwareUpdateErrorHook()

```
void FileFirmwareUpdateErrorHook (
    blt_int8u error_code )
```

Callback that gets called in case an error occurred during a firmware update. Refer to <[file.h](#)> for a list of available error codes.

Returns

none.

7.610.2.8 FileFirmwareUpdateLogHook()

```
void FileFirmwareUpdateLogHook (
    blt_char * info_string )
```

Callback that gets called each time new log information becomes available during a firmware update.

Parameters

<i>info_string</i>	Pointer to a character array with the log entry info.
--------------------	---

Returns

none.

7.610.2.9 FileFirmwareUpdateStartedHook()

```
void FileFirmwareUpdateStartedHook (  
    void )
```

Callback that gets called to inform the application that a firmware update from local storage just started.

Returns

none.

7.610.2.10 FileGetFirmwareFilenameHook()

```
const blt_char* FileGetFirmwareFilenameHook (  
    void )
```

Callback to obtain the filename of the firmware file that should be used during the firmware update from the local file storage. This hook function is called at the beginning of the firmware update from local storage sequence.

Returns

valid firmware filename with full path or BLT_NULL.

7.610.2.11 FileIsFirmwareUpdateRequestedHook()

```
blt_bool FileIsFirmwareUpdateRequestedHook (  
    void )
```

Callback that gets called to check whether a firmware update from local file storage should be started. This could for example be when a switch is pressed, when a certain file is found on the local file storage, etc.

Returns

BLT_TRUE if a firmware update is requested, BLT_FALSE otherwise.

7.610.2.12 NvmDoneHook()

```
blt_bool NvmDoneHook (
    void )
```

Callback that gets called at the end of the NVM programming session.

Returns

BLT_TRUE is successful, BLT_FALSE otherwise.

7.610.2.13 NvmEraseHook()

```
blt_int8u NvmEraseHook (
    blt_addr addr,
    blt_int32u len )
```

Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the memory hasn't been erased yet.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.

Returns

BLT_NVM_OKAY if successful, BLT_NVM_NOT_IN_RANGE if the address is not within the supported memory range, or BLT_NVM_ERROR if the erase operation failed.

7.610.2.14 NvmInitHook()

```
void NvmInitHook (
    void )
```

Callback that gets called at the start of the internal NVM driver initialization routine.

Returns

none.

7.610.2.15 NvmReinitHook()

```
void NvmReinitHook (
    void )
```

Callback that gets called at the start of a firmware update to reinitialize the NVM driver.

Returns

none.

7.610.2.16 NvmWriteHook()

```
blt_int8u NvmWriteHook (
    blt_addr addr,
    blt_int32u len,
    blt_int8u * data )
```

Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the data hasn't been written yet.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.
<i>data</i>	Pointer to the data buffer.

Returns

BLT_NVM_OKAY if successful, BLT_NVM_NOT_IN_RANGE if the address is not within the supported memory range, or BLT_NVM_ERROR if the write operation failed.

7.610.2.17 UsbConnectHook()

```
void UsbConnectHook (
    blt_bool connect )
```

Callback that gets called whenever the USB device should be connected to the USB bus.

Parameters

<i>connect</i>	BLT_TRUE to connect and BLT_FALSE to disconnect.
----------------	--

Returns

none.

7.610.2.18 UsbEnterLowPowerModeHook()

```
void UsbEnterLowPowerModeHook (  
    void )
```

Callback that gets called whenever the USB host requests the device to enter a low power mode.

Returns

none.

7.610.2.19 UsbLeaveLowPowerModeHook()

```
void UsbLeaveLowPowerModeHook (  
    void )
```

Callback that gets called whenever the USB host requests the device to exit low power mode.

Returns

none.

7.610.2.20 XcpGetSeedHook()

```
blt_int8u XcpGetSeedHook (  
    blt_int8u resource,  
    blt_int8u * seed )
```

Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the GET_SEED command.

Parameters

<i>resource</i>	Resource that the seed is requested for (XCP_RES_XXX).
<i>seed</i>	Pointer to byte buffer where the seed will be stored.

Returns

Length of the seed in bytes.

7.610.2.21 XcpVerifyKeyHook()

```
blt_int8u XcpVerifyKeyHook (
    blt_int8u resource,
    blt_int8u * key,
    blt_int8u len )
```

Called by the UNLOCK command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

Parameters

<i>resource</i>	resource to unlock (XCP_RES_XXX).
<i>key</i>	pointer to the byte buffer holding the key.
<i>len</i>	length of the key in bytes.

Returns

1 if the key was correct, 0 otherwise.

7.610.3 Variable Documentation**7.610.3.1 canUse**

```
blt_bool canUse
```

Flag to indicate if the log-file can be used.

7.610.3.2 handle

```
FIL handle
```

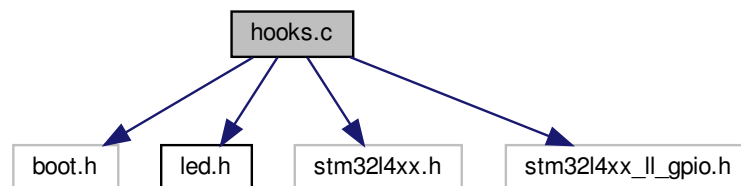
FatFS handle to the log-file.

7.611 hooks.c File Reference

Bootloader callback source file.

```
#include "boot.h"
#include "led.h"
#include "stm32l4xx.h"
#include "stm32l4xx_ll_gpio.h"
```

Include dependency graph for ARMC4_STM32L4_Nucleo_L476RG_CubeIDE/Boot/App/hooks.c:



Functions

- [blt_bool CpuUserProgramStartHook](#) (void)
Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.
- void [CopInitHook](#) (void)
Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.
- void [CopServiceHook](#) (void)
Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.
- void [BackDoorInitHook](#) (void)
Initializes the backdoor entry option.
- [blt_bool BackDoorEntryHook](#) (void)
Checks if a backdoor entry is requested.
- void [NvmInitHook](#) (void)
Callback that gets called at the start of the internal NVM driver initialization routine.
- void [NvmReinitHook](#) (void)
Callback that gets called at the start of a firmware update to reinitialize the NVM driver.
- [blt_int8u NvmWriteHook](#) ([blt_addr](#) addr, [blt_int32u](#) len, [blt_int8u](#) *data)
Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the data hasn't been written yet.
- [blt_int8u NvmEraseHook](#) ([blt_addr](#) addr, [blt_int32u](#) len)
Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the memory hasn't been erased yet.
- [blt_bool NvmDoneHook](#) (void)
Callback that gets called at the end of the NVM programming session.

- [blt_int8u XcpGetSeedHook](#) ([blt_int8u](#) resource, [blt_int8u](#) *seed)

Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the GET_SEED command.

- [blt_int8u XcpVerifyKeyHook](#) ([blt_int8u](#) resource, [blt_int8u](#) *key, [blt_int8u](#) len)

Called by the UNLOCK command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

7.611.1 Detailed Description

Bootloader callback source file.

7.611.2 Function Documentation

7.611.2.1 BackDoorEntryHook()

```
blt_bool BackDoorEntryHook (  
    void )
```

Checks if a backdoor entry is requested.

Returns

BLT_TRUE if the backdoor entry is requested, BLT_FALSE otherwise.

7.611.2.2 BackDoorInitHook()

```
void BackDoorInitHook (  
    void )
```

Initializes the backdoor entry option.

Returns

none.

7.611.2.3 CopInitHook()

```
void CopInitHook (
    void )
```

Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.

Returns

none.

7.611.2.4 CopServiceHook()

```
void CopServiceHook (
    void )
```

Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.

Returns

none.

7.611.2.5 CpuUserProgramStartHook()

```
blt_bool CpuUserProgramStartHook (
    void )
```

Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.

Returns

BLT_TRUE if it is okay to start the user program, BLT_FALSE to keep keep the bootloader active.

7.611.2.6 NvmDoneHook()

```
blt_bool NvmDoneHook (
    void )
```

Callback that gets called at the end of the NVM programming session.

Returns

BLT_TRUE is successful, BLT_FALSE otherwise.

7.611.2.7 NvmEraseHook()

```
blt_int8u NvmEraseHook (
    blt_addr addr,
    blt_int32u len )
```

Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the memory hasn't been erased yet.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.

Returns

BLT_NVM_OKAY if successful, BLT_NVM_NOT_IN_RANGE if the address is not within the supported memory range, or BLT_NVM_ERROR if the erase operation failed.

7.611.2.8 NvmInitHook()

```
void NvmInitHook (
    void )
```

Callback that gets called at the start of the internal NVM driver initialization routine.

Returns

none.

7.611.2.9 NvmReinitHook()

```
void NvmReinitHook (
    void )
```

Callback that gets called at the start of a firmware update to reinitialize the NVM driver.

Returns

none.

7.611.2.10 NvmWriteHook()

```
blt_int8u NvmWriteHook (
    blt_addr addr,
    blt_int32u len,
    blt_int8u * data )
```

Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the data hasn't been written yet.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.
<i>data</i>	Pointer to the data buffer.

Returns

BLT_NVM_OKAY if successful, BLT_NVM_NOT_IN_RANGE if the address is not within the supported memory range, or BLT_NVM_ERROR if the write operation failed.

7.611.2.11 XcpGetSeedHook()

```
blt_int8u XcpGetSeedHook (
    blt_int8u resource,
    blt_int8u * seed )
```

Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the GET_SEED command.

Parameters

<i>resource</i>	Resource that the seed is requested for (XCP_RES_XXX).
<i>seed</i>	Pointer to byte buffer where the seed will be stored.

Returns

Length of the seed in bytes.

7.611.2.12 XcpVerifyKeyHook()

```
blt_int8u XcpVerifyKeyHook (
    blt_int8u resource,
    blt_int8u * key,
    blt_int8u len )
```

Called by the UNLOCK command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

Parameters

<i>resource</i>	resource to unlock (XCP_RES_XXX).
<i>key</i>	pointer to the byte buffer holding the key.
<i>len</i>	length of the key in bytes.

Returns

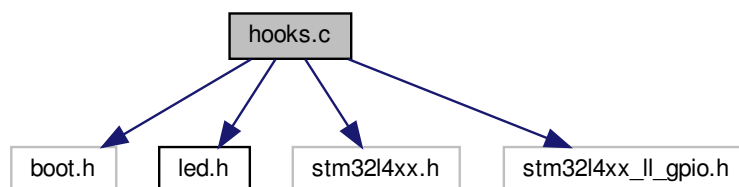
1 if the key was correct, 0 otherwise.

7.612 hooks.c File Reference

Bootloader callback source file.

```
#include "boot.h"
#include "led.h"
#include "stm32l4xx.h"
#include "stm32l4xx_ll_gpio.h"
```

Include dependency graph for ARMCM4_STM32L4_Nucleo_L476RG_GCC/Boot/hooks.c:



Functions

- [blt_bool CpuUserProgramStartHook](#) (void)
Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.
- void [CopInitHook](#) (void)
Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.
- void [CopServiceHook](#) (void)
Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.
- void [BackDoorInitHook](#) (void)
Initializes the backdoor entry option.
- [blt_bool BackDoorEntryHook](#) (void)
Checks if a backdoor entry is requested.
- void [NvmInitHook](#) (void)
Callback that gets called at the start of the internal NVM driver initialization routine.
- void [NvmReinitHook](#) (void)
Callback that gets called at the start of a firmware update to reinitialize the NVM driver.
- [blt_int8u NvmWriteHook](#) ([blt_addr](#) addr, [blt_int32u](#) len, [blt_int8u](#) *data)
Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the data hasn't been written yet.
- [blt_int8u NvmEraseHook](#) ([blt_addr](#) addr, [blt_int32u](#) len)

Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then `BLT_NVM_NOT_IN_RANGE` must be returned to indicate that the memory hasn't been erased yet.

- `blt_bool NvmDoneHook` (void)

Callback that gets called at the end of the NVM programming session.

- `blt_int8u XcpGetSeedHook` (`blt_int8u` resource, `blt_int8u *seed`)

Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the `GET_SEED` command.

- `blt_int8u XcpVerifyKeyHook` (`blt_int8u` resource, `blt_int8u *key`, `blt_int8u len`)

Called by the `UNLOCK` command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

7.612.1 Detailed Description

Bootloader callback source file.

7.612.2 Function Documentation

7.612.2.1 BackDoorEntryHook()

```
blt_bool BackDoorEntryHook (
    void )
```

Checks if a backdoor entry is requested.

Returns

`BLT_TRUE` if the backdoor entry is requested, `BLT_FALSE` otherwise.

7.612.2.2 BackDoorInitHook()

```
void BackDoorInitHook (
    void )
```

Initializes the backdoor entry option.

Returns

none.

7.612.2.3 CopInitHook()

```
void CopInitHook (  
    void )
```

Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.

Returns

none.

7.612.2.4 CopServiceHook()

```
void CopServiceHook (  
    void )
```

Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.

Returns

none.

7.612.2.5 CpuUserProgramStartHook()

```
blt_bool CpuUserProgramStartHook (  
    void )
```

Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.

Returns

BLT_TRUE if it is okay to start the user program, BLT_FALSE to keep keep the bootloader active.

7.612.2.6 NvmDoneHook()

```
blt_bool NvmDoneHook (  
    void )
```

Callback that gets called at the end of the NVM programming session.

Returns

BLT_TRUE is successful, BLT_FALSE otherwise.

7.612.2.7 NvmEraseHook()

```
blt_int8u NvmEraseHook (
    blt_addr addr,
    blt_int32u len )
```

Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the memory hasn't been erased yet.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.

Returns

BLT_NVM_OKAY if successful, BLT_NVM_NOT_IN_RANGE if the address is not within the supported memory range, or BLT_NVM_ERROR if the erase operation failed.

7.612.2.8 NvmInitHook()

```
void NvmInitHook (
    void )
```

Callback that gets called at the start of the internal NVM driver initialization routine.

Returns

none.

7.612.2.9 NvmReinitHook()

```
void NvmReinitHook (
    void )
```

Callback that gets called at the start of a firmware update to reinitialize the NVM driver.

Returns

none.

7.612.2.10 NvmWriteHook()

```
blt_int8u NvmWriteHook (
    blt_addr addr,
    blt_int32u len,
    blt_int8u * data )
```

Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the data hasn't been written yet.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.
<i>data</i>	Pointer to the data buffer.

Returns

BLT_NVM_OKAY if successful, BLT_NVM_NOT_IN_RANGE if the address is not within the supported memory range, or BLT_NVM_ERROR if the write operation failed.

7.612.2.11 XcpGetSeedHook()

```
blt_int8u XcpGetSeedHook (
    blt_int8u resource,
    blt_int8u * seed )
```

Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the GET_SEED command.

Parameters

<i>resource</i>	Resource that the seed is requested for (XCP_RES_XXX).
<i>seed</i>	Pointer to byte buffer where the seed will be stored.

Returns

Length of the seed in bytes.

7.612.2.12 XcpVerifyKeyHook()

```
blt_int8u XcpVerifyKeyHook (
    blt_int8u resource,
    blt_int8u * key,
    blt_int8u len )
```

Called by the UNLOCK command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

Parameters

<i>resource</i>	resource to unlock (XCP_RES_XXX).
<i>key</i>	pointer to the byte buffer holding the key.
<i>len</i>	length of the key in bytes.

Returns

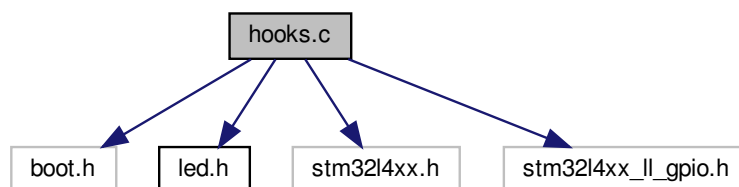
1 if the key was correct, 0 otherwise.

7.613 hooks.c File Reference

Bootloader callback source file.

```
#include "boot.h"
#include "led.h"
#include "stm32l4xx.h"
#include "stm32l4xx_ll_gpio.h"
```

Include dependency graph for ARMCM4_STM32L4_Nucleo_L476RG_IAR/Boot/hooks.c:



Functions

- [blt_bool CpuUserProgramStartHook](#) (void)
Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.
- void [CopInitHook](#) (void)
Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.
- void [CopServiceHook](#) (void)
Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.
- void [BackDoorInitHook](#) (void)
Initializes the backdoor entry option.
- [blt_bool BackDoorEntryHook](#) (void)
Checks if a backdoor entry is requested.
- void [NvmInitHook](#) (void)
Callback that gets called at the start of the internal NVM driver initialization routine.
- void [NvmReinitHook](#) (void)
Callback that gets called at the start of a firmware update to reinitialize the NVM driver.
- [blt_int8u NvmWriteHook](#) ([blt_addr](#) addr, [blt_int32u](#) len, [blt_int8u](#) *data)
Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the data hasn't been written yet.
- [blt_int8u NvmEraseHook](#) ([blt_addr](#) addr, [blt_int32u](#) len)

Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then `BLT_NVM_NOT_IN_RANGE` must be returned to indicate that the memory hasn't been erased yet.

- `blt_bool NvmDoneHook` (void)

Callback that gets called at the end of the NVM programming session.

- `blt_int8u XcpGetSeedHook` (`blt_int8u` resource, `blt_int8u *seed`)

Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the `GET_SEED` command.

- `blt_int8u XcpVerifyKeyHook` (`blt_int8u` resource, `blt_int8u *key`, `blt_int8u len`)

Called by the `UNLOCK` command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

7.613.1 Detailed Description

Bootloader callback source file.

7.613.2 Function Documentation

7.613.2.1 BackDoorEntryHook()

```
blt_bool BackDoorEntryHook (
    void )
```

Checks if a backdoor entry is requested.

Returns

`BLT_TRUE` if the backdoor entry is requested, `BLT_FALSE` otherwise.

7.613.2.2 BackDoorInitHook()

```
void BackDoorInitHook (
    void )
```

Initializes the backdoor entry option.

Returns

none.

7.613.2.3 CopInitHook()

```
void CopInitHook (  
    void )
```

Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.

Returns

none.

7.613.2.4 CopServiceHook()

```
void CopServiceHook (  
    void )
```

Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.

Returns

none.

7.613.2.5 CpuUserProgramStartHook()

```
blt_bool CpuUserProgramStartHook (  
    void )
```

Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.

Returns

BLT_TRUE if it is okay to start the user program, BLT_FALSE to keep keep the bootloader active.

7.613.2.6 NvmDoneHook()

```
blt_bool NvmDoneHook (  
    void )
```

Callback that gets called at the end of the NVM programming session.

Returns

BLT_TRUE is successful, BLT_FALSE otherwise.

7.613.2.7 NvmEraseHook()

```
blt_int8u NvmEraseHook (
    blt_addr addr,
    blt_int32u len )
```

Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the memory hasn't been erased yet.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.

Returns

BLT_NVM_OKAY if successful, BLT_NVM_NOT_IN_RANGE if the address is not within the supported memory range, or BLT_NVM_ERROR if the erase operation failed.

7.613.2.8 NvmInitHook()

```
void NvmInitHook (
    void )
```

Callback that gets called at the start of the internal NVM driver initialization routine.

Returns

none.

7.613.2.9 NvmReinitHook()

```
void NvmReinitHook (
    void )
```

Callback that gets called at the start of a firmware update to reinitialize the NVM driver.

Returns

none.

7.613.2.10 NvmWriteHook()

```
blt_int8u NvmWriteHook (
    blt_addr addr,
    blt_int32u len,
    blt_int8u * data )
```

Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the data hasn't been written yet.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.
<i>data</i>	Pointer to the data buffer.

Returns

BLT_NVM_OKAY if successful, BLT_NVM_NOT_IN_RANGE if the address is not within the supported memory range, or BLT_NVM_ERROR if the write operation failed.

7.613.2.11 XcpGetSeedHook()

```
blt_int8u XcpGetSeedHook (
    blt_int8u resource,
    blt_int8u * seed )
```

Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the GET_SEED command.

Parameters

<i>resource</i>	Resource that the seed is requested for (XCP_RES_XXX).
<i>seed</i>	Pointer to byte buffer where the seed will be stored.

Returns

Length of the seed in bytes.

7.613.2.12 XcpVerifyKeyHook()

```
blt_int8u XcpVerifyKeyHook (
    blt_int8u resource,
    blt_int8u * key,
    blt_int8u len )
```

Called by the UNLOCK command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

Parameters

<i>resource</i>	resource to unlock (XCP_RES_XXX).
<i>key</i>	pointer to the byte buffer holding the key.
<i>len</i>	length of the key in bytes.

Returns

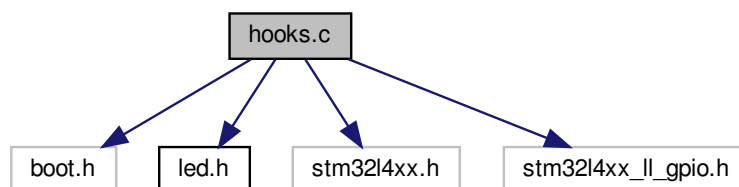
1 if the key was correct, 0 otherwise.

7.614 hooks.c File Reference

Bootloader callback source file.

```
#include "boot.h"
#include "led.h"
#include "stm32l4xx.h"
#include "stm32l4xx_ll_gpio.h"
```

Include dependency graph for ARMCM4_STM32L4_Nucleo_L476RG_Keil/Boot/hooks.c:



Functions

- [blt_bool CpuUserProgramStartHook](#) (void)
Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.
- void [CopInitHook](#) (void)
Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.
- void [CopServiceHook](#) (void)
Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.
- void [BackDoorInitHook](#) (void)
Initializes the backdoor entry option.
- [blt_bool BackDoorEntryHook](#) (void)
Checks if a backdoor entry is requested.
- void [NvmInitHook](#) (void)
Callback that gets called at the start of the internal NVM driver initialization routine.
- void [NvmReinitHook](#) (void)
Callback that gets called at the start of a firmware update to reinitialize the NVM driver.
- [blt_int8u NvmWriteHook](#) ([blt_addr](#) addr, [blt_int32u](#) len, [blt_int8u](#) *data)
Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the data hasn't been written yet.
- [blt_int8u NvmEraseHook](#) ([blt_addr](#) addr, [blt_int32u](#) len)

Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then `BLT_NVM_NOT_IN_RANGE` must be returned to indicate that the memory hasn't been erased yet.

- `blt_bool NvmDoneHook` (void)

Callback that gets called at the end of the NVM programming session.

- `blt_int8u XcpGetSeedHook` (`blt_int8u` resource, `blt_int8u *seed`)

Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the `GET_SEED` command.

- `blt_int8u XcpVerifyKeyHook` (`blt_int8u` resource, `blt_int8u *key`, `blt_int8u len`)

Called by the `UNLOCK` command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

7.614.1 Detailed Description

Bootloader callback source file.

7.614.2 Function Documentation

7.614.2.1 BackDoorEntryHook()

```
blt_bool BackDoorEntryHook (
    void )
```

Checks if a backdoor entry is requested.

Returns

`BLT_TRUE` if the backdoor entry is requested, `BLT_FALSE` otherwise.

7.614.2.2 BackDoorInitHook()

```
void BackDoorInitHook (
    void )
```

Initializes the backdoor entry option.

Returns

none.

7.614.2.3 CopInitHook()

```
void CopInitHook (  
    void )
```

Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.

Returns

none.

7.614.2.4 CopServiceHook()

```
void CopServiceHook (  
    void )
```

Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.

Returns

none.

7.614.2.5 CpuUserProgramStartHook()

```
blt_bool CpuUserProgramStartHook (  
    void )
```

Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.

Returns

BLT_TRUE if it is okay to start the user program, BLT_FALSE to keep keep the bootloader active.

7.614.2.6 NvmDoneHook()

```
blt_bool NvmDoneHook (  
    void )
```

Callback that gets called at the end of the NVM programming session.

Returns

BLT_TRUE is successful, BLT_FALSE otherwise.

7.614.2.7 NvmEraseHook()

```
blt_int8u NvmEraseHook (
    blt_addr addr,
    blt_int32u len )
```

Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the memory hasn't been erased yet.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.

Returns

BLT_NVM_OKAY if successful, BLT_NVM_NOT_IN_RANGE if the address is not within the supported memory range, or BLT_NVM_ERROR if the erase operation failed.

7.614.2.8 NvmInitHook()

```
void NvmInitHook (
    void )
```

Callback that gets called at the start of the internal NVM driver initialization routine.

Returns

none.

7.614.2.9 NvmReinitHook()

```
void NvmReinitHook (
    void )
```

Callback that gets called at the start of a firmware update to reinitialize the NVM driver.

Returns

none.

7.614.2.10 NvmWriteHook()

```
blt_int8u NvmWriteHook (
    blt_addr addr,
    blt_int32u len,
    blt_int8u * data )
```

Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the data hasn't been written yet.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.
<i>data</i>	Pointer to the data buffer.

Returns

BLT_NVM_OKAY if successful, BLT_NVM_NOT_IN_RANGE if the address is not within the supported memory range, or BLT_NVM_ERROR if the write operation failed.

7.614.2.11 XcpGetSeedHook()

```
blt_int8u XcpGetSeedHook (
    blt_int8u resource,
    blt_int8u * seed )
```

Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the GET_SEED command.

Parameters

<i>resource</i>	Resource that the seed is requested for (XCP_RES_XXX).
<i>seed</i>	Pointer to byte buffer where the seed will be stored.

Returns

Length of the seed in bytes.

7.614.2.12 XcpVerifyKeyHook()

```
blt_int8u XcpVerifyKeyHook (
    blt_int8u resource,
    blt_int8u * key,
    blt_int8u len )
```

Called by the UNLOCK command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

Parameters

<i>resource</i>	resource to unlock (XCP_RES_XXX).
<i>key</i>	pointer to the byte buffer holding the key.
<i>len</i>	length of the key in bytes.

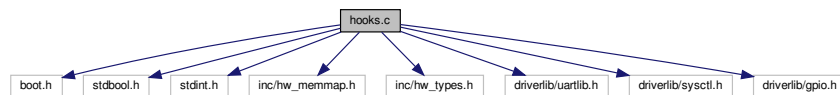
Returns

1 if the key was correct, 0 otherwise.

7.615 hooks.c File Reference

Bootloader callback source file.

```
#include "boot.h"
#include <stdbool.h>
#include <stdint.h>
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "driverlib/uartlib.h"
#include "driverlib/sysctl.h"
#include "driverlib/gpio.h"
Include dependency graph for ARMCM4_TM4C_DK_TM4C123G_IAR/Boot/hooks.c:
```



Functions

- void [UsbConnectHook](#) ([blt_bool](#) connect)
Callback that gets called whenever the USB device should be connected to the USB bus.
- void [UsbEnterLowPowerModeHook](#) (void)
Callback that gets called whenever the USB host requests the device to enter a low power mode.
- void [UsbLeaveLowPowerModeHook](#) (void)
Callback that gets called whenever the USB host requests the device to exit low power mode.
- void [BackDoorInitHook](#) (void)
Initializes the backdoor entry option.
- [blt_bool](#) [BackDoorEntryHook](#) (void)
Checks if a backdoor entry is requested.
- [blt_bool](#) [CpuUserProgramStartHook](#) (void)
Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.
- void [NvmInitHook](#) (void)
Callback that gets called at the start of the internal NVM driver initialization routine.
- void [NvmReinitHook](#) (void)
Callback that gets called at the start of a firmware update to reinitialize the NVM driver.
- [blt_int8u](#) [NvmWriteHook](#) ([blt_addr](#) addr, [blt_int32u](#) len, [blt_int8u](#) *data)
Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the data hasn't been written yet.
- [blt_int8u](#) [NvmEraseHook](#) ([blt_addr](#) addr, [blt_int32u](#) len)

Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then `BLT_NVM_NOT_IN_RANGE` must be returned to indicate that the memory hasn't been erased yet.

- `blt_bool NvmDoneHook (void)`

Callback that gets called at the end of the NVM programming session.

- `void CopInitHook (void)`

Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.

- `void CopServiceHook (void)`

Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.

- `blt_bool FileIsFirmwareUpdateRequestedHook (void)`

Callback that gets called to check whether a firmware update from local file storage should be started. This could for example be when a switch is pressed, when a certain file is found on the local file storage, etc.

- `const blt_char * FileGetFirmwareFilenameHook (void)`

Callback to obtain the filename of the firmware file that should be used during the firmware update from the local file storage. This hook function is called at the beginning of the firmware update from local storage sequence.

- `void FileFirmwareUpdateStartedHook (void)`

Callback that gets called to inform the application that a firmware update from local storage just started.

- `void FileFirmwareUpdateCompletedHook (void)`

Callback that gets called to inform the application that a firmware update was successfully completed.

- `void FileFirmwareUpdateErrorHook (blt_int8u error_code)`

Callback that gets called in case an error occurred during a firmware update. Refer to `<file.h>` for a list of available error codes.

- `void FileFirmwareUpdateLogHook (blt_char *info_string)`

Callback that gets called each time new log information becomes available during a firmware update.

- `blt_int8u XcpGetSeedHook (blt_int8u resource, blt_int8u *seed)`

Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the `GET_SEED` command.

- `blt_int8u XcpVerifyKeyHook (blt_int8u resource, blt_int8u *key, blt_int8u len)`

Called by the `UNLOCK` command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

Variables

- `static const blt_char firmwareFilename [] = "/demoprogram_dk_tm4c123g.srec"`

Firmware filename.

-

```
struct {
    FIL handle
    blt_bool canUse
} logfile
```

Data structure for grouping log-file related information.

7.615.1 Detailed Description

Bootloader callback source file.

7.615.2 Function Documentation

7.615.2.1 BackDoorEntryHook()

```
blt_bool BackDoorEntryHook (  
    void )
```

Checks if a backdoor entry is requested.

Returns

BLT_TRUE if the backdoor entry is requested, BLT_FALSE otherwise.

7.615.2.2 BackDoorInitHook()

```
void BackDoorInitHook (  
    void )
```

Initializes the backdoor entry option.

Returns

none.

7.615.2.3 CopInitHook()

```
void CopInitHook (  
    void )
```

Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.

Returns

none.

7.615.2.4 CopServiceHook()

```
void CopServiceHook (
    void )
```

Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.

Returns

none.

7.615.2.5 CpuUserProgramStartHook()

```
blt_bool CpuUserProgramStartHook (
    void )
```

Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.

Returns

BLT_TRUE if it is okay to start the user program, BLT_FALSE to keep keep the bootloader active.

7.615.2.6 FileFirmwareUpdateCompletedHook()

```
void FileFirmwareUpdateCompletedHook (
    void )
```

Callback that gets called to inform the application that a firmware update was successfully completed.

Returns

none.

7.615.2.7 FileFirmwareUpdateErrorHook()

```
void FileFirmwareUpdateErrorHook (
    blt_int8u error_code )
```

Callback that gets called in case an error occurred during a firmware update. Refer to <[file.h](#)> for a list of available error codes.

Returns

none.

7.615.2.8 FileFirmwareUpdateLogHook()

```
void FileFirmwareUpdateLogHook (
    blt_char * info_string )
```

Callback that gets called each time new log information becomes available during a firmware update.

Parameters

<i>info_string</i>	Pointer to a character array with the log entry info.
--------------------	---

Returns

none.

7.615.2.9 FileFirmwareUpdateStartedHook()

```
void FileFirmwareUpdateStartedHook (  
    void )
```

Callback that gets called to inform the application that a firmware update from local storage just started.

Returns

none.

7.615.2.10 FileGetFirmwareFilenameHook()

```
const blt_char* FileGetFirmwareFilenameHook (  
    void )
```

Callback to obtain the filename of the firmware file that should be used during the firmware update from the local file storage. This hook function is called at the beginning of the firmware update from local storage sequence.

Returns

valid firmware filename with full path or BLT_NULL.

7.615.2.11 FileIsFirmwareUpdateRequestedHook()

```
blt_bool FileIsFirmwareUpdateRequestedHook (  
    void )
```

Callback that gets called to check whether a firmware update from local file storage should be started. This could for example be when a switch is pressed, when a certain file is found on the local file storage, etc.

Returns

BLT_TRUE if a firmware update is requested, BLT_FALSE otherwise.

7.615.2.12 NvmDoneHook()

```
blt_bool NvmDoneHook (
    void )
```

Callback that gets called at the end of the NVM programming session.

Returns

BLT_TRUE is successful, BLT_FALSE otherwise.

7.615.2.13 NvmEraseHook()

```
blt_int8u NvmEraseHook (
    blt_addr addr,
    blt_int32u len )
```

Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the memory hasn't been erased yet.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.

Returns

BLT_NVM_OKAY if successful, BLT_NVM_NOT_IN_RANGE if the address is not within the supported memory range, or BLT_NVM_ERROR is the erase operation failed.

7.615.2.14 NvmInitHook()

```
void NvmInitHook (
    void )
```

Callback that gets called at the start of the internal NVM driver initialization routine.

Returns

none.

7.615.2.15 NvmReinitHook()

```
void NvmReinitHook (
    void )
```

Callback that gets called at the start of a firmware update to reinitialize the NVM driver.

Returns

none.

7.615.2.16 NvmWriteHook()

```
blt_int8u NvmWriteHook (
    blt_addr addr,
    blt_int32u len,
    blt_int8u * data )
```

Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then `BLT_NVM_NOT_IN_RANGE` must be returned to indicate that the data hasn't been written yet.

Parameters

<i>addr</i>	Start address.
<i>len</i>	Length in bytes.
<i>data</i>	Pointer to the data buffer.

Returns

`BLT_NVM_OKAY` if successful, `BLT_NVM_NOT_IN_RANGE` if the address is not within the supported memory range, or `BLT_NVM_ERROR` if the write operation failed.

7.615.2.17 UsbConnectHook()

```
void UsbConnectHook (
    blt_bool connect )
```

Callback that gets called whenever the USB device should be connected to the USB bus.

Parameters

<i>connect</i>	<code>BLT_TRUE</code> to connect and <code>BLT_FALSE</code> to disconnect.
----------------	--

Returns

none.

7.615.2.18 UsbEnterLowPowerModeHook()

```
void UsbEnterLowPowerModeHook (  
    void )
```

Callback that gets called whenever the USB host requests the device to enter a low power mode.

Returns

none.

7.615.2.19 UsbLeaveLowPowerModeHook()

```
void UsbLeaveLowPowerModeHook (  
    void )
```

Callback that gets called whenever the USB host requests the device to exit low power mode.

Returns

none.

7.615.2.20 XcpGetSeedHook()

```
blt_int8u XcpGetSeedHook (  
    blt_int8u resource,  
    blt_int8u * seed )
```

Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the GET_SEED command.

Parameters

<i>resource</i>	Resource that the seed is requested for (XCP_RES_XXX).
<i>seed</i>	Pointer to byte buffer where the seed will be stored.

Returns

Length of the seed in bytes.

7.615.2.21 XcpVerifyKeyHook()

```
blt_int8u XcpVerifyKeyHook (
    blt_int8u resource,
    blt_int8u * key,
    blt_int8u len )
```

Called by the UNLOCK command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

Parameters

<i>resource</i>	resource to unlock (XCP_RES_XXX).
<i>key</i>	pointer to the byte buffer holding the key.
<i>len</i>	length of the key in bytes.

Returns

1 if the key was correct, 0 otherwise.

7.615.3 Variable Documentation**7.615.3.1 canUse**

```
blt_bool canUse
```

Flag to indicate if the log-file can be used.

7.615.3.2 handle

```
FIL handle
```

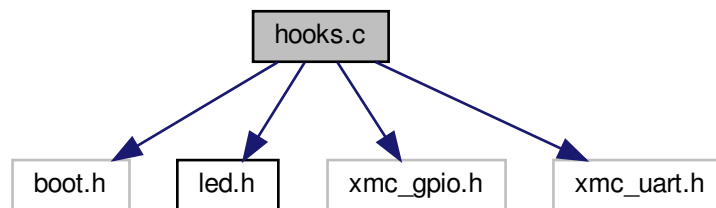
FatFS handle to the log-file.

7.616 hooks.c File Reference

Bootloader callback source file.

```
#include "boot.h"
#include "led.h"
#include "xmc_gpio.h"
#include "xmc_uart.h"
```

Include dependency graph for ARMC4700_Relax_Kit_GCC/Boot/hooks.c:



Functions

- void [BackDoorInitHook](#) (void)
Initializes the backdoor entry option.
- [blt_bool](#) [BackDoorEntryHook](#) (void)
Checks if a backdoor entry is requested.
- [blt_bool](#) [CpuUserProgramStartHook](#) (void)
Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.
- void [CopInitHook](#) (void)
Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.
- void [CopServiceHook](#) (void)
Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.
- void [NvmInitHook](#) (void)
Callback that gets called at the start of the internal NVM driver initialization routine.
- void [NvmReinitHook](#) (void)
Callback that gets called at the start of a firmware update to reinitialize the NVM driver.
- [blt_int8u](#) [NvmWriteHook](#) ([blt_addr](#) addr, [blt_int32u](#) len, [blt_int8u](#) *data)
Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the data hasn't been written yet.
- [blt_int8u](#) [NvmEraseHook](#) ([blt_addr](#) addr, [blt_int32u](#) len)
Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT_NVM_NOT_IN_RANGE must be returned to indicate that the memory hasn't been erased yet.
- [blt_bool](#) [NvmDoneHook](#) (void)

Callback that gets called at the end of the NVM programming session.

- `blt_bool` `FileIsFirmwareUpdateRequestedHook` (void)
Callback that gets called to check whether a firmware update from local file storage should be started. This could for example be when a switch is pressed, when a certain file is found on the local file storage, etc.
- `const blt_char *` `FileGetFirmwareFilenameHook` (void)
Callback to obtain the filename of the firmware file that should be used during the firmware update from the local file storage. This hook function is called at the beginning of the firmware update from local storage sequence.
- `void` `FileFirmwareUpdateStartedHook` (void)
Callback that gets called to inform the application that a firmware update from local storage just started.
- `void` `FileFirmwareUpdateCompletedHook` (void)
Callback that gets called to inform the application that a firmware update was successfully completed.
- `void` `FileFirmwareUpdateErrorHook` (`blt_int8u` error_code)
Callback that gets called in case an error occurred during a firmware update. Refer to <file.h> for a list of available error codes.
- `void` `FileFirmwareUpdateLogHook` (`blt_char *`info_string)
Callback that gets called each time new log information becomes available during a firmware update.
- `blt_int8u` `XcpGetSeedHook` (`blt_int8u` resource, `blt_int8u *`seed)
Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the GET_SEED command.
- `blt_int8u` `XcpVerifyKeyHook` (`blt_int8u` resource, `blt_int8u *`key, `blt_int8u` len)
Called by the UNLOCK command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

Variables

- `static const blt_char` `firmwareFilename` [] = `"/demoprogram_xmc4700.srec"`
Firmware filename.
- ```

struct {
 FIL handle
 blt_bool canUse
} logfile

```

*Data structure for grouping log-file related information.*

## 7.616.1 Detailed Description

Bootloader callback source file.

## 7.616.2 Function Documentation

### 7.616.2.1 BackDoorEntryHook()

```
blt_bool BackDoorEntryHook (
 void)
```

Checks if a backdoor entry is requested.

#### Returns

BLT\_TRUE if the backdoor entry is requested, BLT\_FALSE otherwise.

#### 7.616.2.2 BackDoorInitHook()

```
void BackDoorInitHook (
 void)
```

Initializes the backdoor entry option.

##### Returns

none.

#### 7.616.2.3 CopInitHook()

```
void CopInitHook (
 void)
```

Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.

##### Returns

none.

#### 7.616.2.4 CopServiceHook()

```
void CopServiceHook (
 void)
```

Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.

##### Returns

none.

#### 7.616.2.5 CpuUserProgramStartHook()

```
blt_bool CpuUserProgramStartHook (
 void)
```

Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.

##### Returns

BLT\_TRUE if it is okay to start the user program, BLT\_FALSE to keep keep the bootloader active.

### 7.616.2.6 FileFirmwareUpdateCompletedHook()

```
void FileFirmwareUpdateCompletedHook (
 void)
```

Callback that gets called to inform the application that a firmware update was successfully completed.

#### Returns

none.

### 7.616.2.7 FileFirmwareUpdateErrorHook()

```
void FileFirmwareUpdateErrorHook (
 blt_int8u error_code)
```

Callback that gets called in case an error occurred during a firmware update. Refer to <[file.h](#)> for a list of available error codes.

#### Returns

none.

### 7.616.2.8 FileFirmwareUpdateLogHook()

```
void FileFirmwareUpdateLogHook (
 blt_char * info_string)
```

Callback that gets called each time new log information becomes available during a firmware update.

#### Parameters

|                    |                                                       |
|--------------------|-------------------------------------------------------|
| <i>info_string</i> | Pointer to a character array with the log entry info. |
|--------------------|-------------------------------------------------------|

#### Returns

none.

### 7.616.2.9 FileFirmwareUpdateStartedHook()

```
void FileFirmwareUpdateStartedHook (
 void)
```

Callback that gets called to inform the application that a firmware update from local storage just started.

**Returns**

none.

**7.616.2.10 FileGetFirmwareFilenameHook()**

```
const blt_char* FileGetFirmwareFilenameHook (
 void)
```

Callback to obtain the filename of the firmware file that should be used during the firmware update from the local file storage. This hook function is called at the beginning of the firmware update from local storage sequence.

**Returns**

valid firmware filename with full path or BLT\_NULL.

**7.616.2.11 FileIsFirmwareUpdateRequestedHook()**

```
blt_bool FileIsFirmwareUpdateRequestedHook (
 void)
```

Callback that gets called to check whether a firmware update from local file storage should be started. This could for example be when a switch is pressed, when a certain file is found on the local file storage, etc.

**Returns**

BLT\_TRUE if a firmware update is requested, BLT\_FALSE otherwise.

**7.616.2.12 NvmDoneHook()**

```
blt_bool NvmDoneHook (
 void)
```

Callback that gets called at the end of the NVM programming session.

**Returns**

BLT\_TRUE is successful, BLT\_FALSE otherwise.

**7.616.2.13 NvmEraseHook()**

```
blt_int8u NvmEraseHook (
 blt_addr addr,
 blt_int32u len)
```

Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the memory hasn't been erased yet.

## Parameters

|             |                  |
|-------------|------------------|
| <i>addr</i> | Start address.   |
| <i>len</i>  | Length in bytes. |

## Returns

BLT\_NVM\_OKAY if successful, BLT\_NVM\_NOT\_IN\_RANGE if the address is not within the supported memory range, or BLT\_NVM\_ERROR if the erase operation failed.

## 7.616.2.14 NvmInitHook()

```
void NvmInitHook (
 void)
```

Callback that gets called at the start of the internal NVM driver initialization routine.

## Returns

none.

## 7.616.2.15 NvmReinitHook()

```
void NvmReinitHook (
 void)
```

Callback that gets called at the start of a firmware update to reinitialize the NVM driver.

## Returns

none.

## 7.616.2.16 NvmWriteHook()

```
blt_int8u NvmWriteHook (
 blt_addr addr,
 blt_int32u len,
 blt_int8u * data)
```

Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the data hasn't been written yet.

**Parameters**

|             |                             |
|-------------|-----------------------------|
| <i>addr</i> | Start address.              |
| <i>len</i>  | Length in bytes.            |
| <i>data</i> | Pointer to the data buffer. |

**Returns**

BLT\_NVM\_OKAY if successful, BLT\_NVM\_NOT\_IN\_RANGE if the address is not within the supported memory range, or BLT\_NVM\_ERROR if the write operation failed.

**7.616.2.17 XcpGetSeedHook()**

```
blt_int8u XcpGetSeedHook (
 blt_int8u resource,
 blt_int8u * seed)
```

Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the GET\_SEED command.

**Parameters**

|                 |                                                        |
|-----------------|--------------------------------------------------------|
| <i>resource</i> | Resource that the seed is requested for (XCP_RES_XXX). |
| <i>seed</i>     | Pointer to byte buffer where the seed will be stored.  |

**Returns**

Length of the seed in bytes.

**7.616.2.18 XcpVerifyKeyHook()**

```
blt_int8u XcpVerifyKeyHook (
 blt_int8u resource,
 blt_int8u * key,
 blt_int8u len)
```

Called by the UNLOCK command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

**Parameters**

|                 |                                             |
|-----------------|---------------------------------------------|
| <i>resource</i> | resource to unlock (XCP_RES_XXX).           |
| <i>key</i>      | pointer to the byte buffer holding the key. |
| <i>len</i>      | length of the key in bytes.                 |



### Returns

1 if the key was correct, 0 otherwise.

## 7.616.3 Variable Documentation

### 7.616.3.1 canUse

`blt_bool` canUse

Flag to indicate if the log-file can be used.

### 7.616.3.2 handle

`FIL` handle

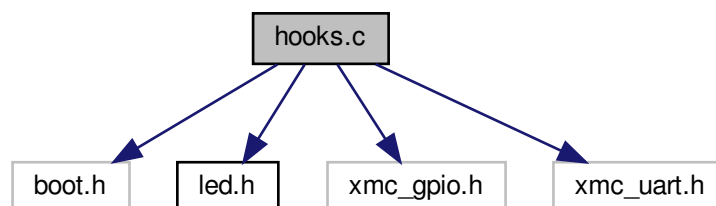
FatFS handle to the log-file.

## 7.617 hooks.c File Reference

Bootloader callback source file.

```
#include "boot.h"
#include "led.h"
#include "xmc_gpio.h"
#include "xmc_uart.h"
```

Include dependency graph for ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_IAR/Boot/hooks.c:



## Functions

- void [BackDoorInitHook](#) (void)  
*Initializes the backdoor entry option.*
- [blt\\_bool BackDoorEntryHook](#) (void)  
*Checks if a backdoor entry is requested.*
- [blt\\_bool CpuUserProgramStartHook](#) (void)  
*Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.*
- void [CopInitHook](#) (void)  
*Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.*
- void [CopServiceHook](#) (void)  
*Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.*
- void [NvmInitHook](#) (void)  
*Callback that gets called at the start of the internal NVM driver initialization routine.*
- void [NvmReinitHook](#) (void)  
*Callback that gets called at the start of a firmware update to reinitialize the NVM driver.*
- [blt\\_int8u NvmWriteHook](#) ([blt\\_addr](#) addr, [blt\\_int32u](#) len, [blt\\_int8u](#) \*data)  
*Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the data hasn't been written yet.*
- [blt\\_int8u NvmEraseHook](#) ([blt\\_addr](#) addr, [blt\\_int32u](#) len)  
*Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the memory hasn't been erased yet.*
- [blt\\_bool NvmDoneHook](#) (void)  
*Callback that gets called at the end of the NVM programming session.*
- [blt\\_bool FileIsFirmwareUpdateRequestedHook](#) (void)  
*Callback that gets called to check whether a firmware update from local file storage should be started. This could for example be when a switch is pressed, when a certain file is found on the local file storage, etc.*
- const [blt\\_char](#) \* [FileGetFirmwareFilenameHook](#) (void)  
*Callback to obtain the filename of the firmware file that should be used during the firmware update from the local file storage. This hook function is called at the beginning of the firmware update from local storage sequence.*
- void [FileFirmwareUpdateStartedHook](#) (void)  
*Callback that gets called to inform the application that a firmware update from local storage just started.*
- void [FileFirmwareUpdateCompletedHook](#) (void)  
*Callback that gets called to inform the application that a firmware update was successfully completed.*
- void [FileFirmwareUpdateErrorHook](#) ([blt\\_int8u](#) error\_code)  
*Callback that gets called in case an error occurred during a firmware update. Refer to <file.h> for a list of available error codes.*
- void [FileFirmwareUpdateLogHook](#) ([blt\\_char](#) \*info\_string)  
*Callback that gets called each time new log information becomes available during a firmware update.*
- [blt\\_int8u XcpGetSeedHook](#) ([blt\\_int8u](#) resource, [blt\\_int8u](#) \*seed)  
*Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the GET\_SEED command.*
- [blt\\_int8u XcpVerifyKeyHook](#) ([blt\\_int8u](#) resource, [blt\\_int8u](#) \*key, [blt\\_int8u](#) len)  
*Called by the UNLOCK command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.*

## Variables

- static const `blt_char firmwareFilename []` = `"/demoprogram_xmc4700.srec"`  
*Firmware filename.*
- struct {  
    FIL `handle`  
    `blt_bool` `canUse`  
} `logfile`

*Data structure for grouping log-file related information.*

### 7.617.1 Detailed Description

Bootloader callback source file.

### 7.617.2 Function Documentation

#### 7.617.2.1 BackDoorEntryHook()

```
blt_bool BackDoorEntryHook (
 void)
```

Checks if a backdoor entry is requested.

#### Returns

BLT\_TRUE if the backdoor entry is requested, BLT\_FALSE otherwise.

#### 7.617.2.2 BackDoorInitHook()

```
void BackDoorInitHook (
 void)
```

Initializes the backdoor entry option.

#### Returns

none.

#### 7.617.2.3 CopInitHook()

```
void CopInitHook (
 void)
```

Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.

##### Returns

none.

#### 7.617.2.4 CopServiceHook()

```
void CopServiceHook (
 void)
```

Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.

##### Returns

none.

#### 7.617.2.5 CpuUserProgramStartHook()

```
blt_bool CpuUserProgramStartHook (
 void)
```

Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.

##### Returns

BLT\_TRUE if it is okay to start the user program, BLT\_FALSE to keep keep the bootloader active.

#### 7.617.2.6 FileFirmwareUpdateCompletedHook()

```
void FileFirmwareUpdateCompletedHook (
 void)
```

Callback that gets called to inform the application that a firmware update was successfully completed.

##### Returns

none.

Referenced by FileTask().

### 7.617.2.7 FileFirmwareUpdateErrorHook()

```
void FileFirmwareUpdateErrorHook (
 blt_int8u error_code)
```

Callback that gets called in case an error occurred during a firmware update. Refer to <[file.h](#)> for a list of available error codes.

#### Returns

none.

Referenced by FileTask().

### 7.617.2.8 FileFirmwareUpdateLogHook()

```
void FileFirmwareUpdateLogHook (
 blt_char * info_string)
```

Callback that gets called each time new log information becomes available during a firmware update.

#### Parameters

|                    |                                                       |
|--------------------|-------------------------------------------------------|
| <i>info_string</i> | Pointer to a character array with the log entry info. |
|--------------------|-------------------------------------------------------|

#### Returns

none.

Referenced by FileTask().

### 7.617.2.9 FileFirmwareUpdateStartedHook()

```
void FileFirmwareUpdateStartedHook (
 void)
```

Callback that gets called to inform the application that a firmware update from local storage just started.

#### Returns

none.

Referenced by FileTask().

#### 7.617.2.10 FileGetFirmwareFilenameHook()

```
const blt_char* FileGetFirmwareFilenameHook (
 void)
```

Callback to obtain the filename of the firmware file that should be used during the firmware update from the local file storage. This hook function is called at the beginning of the firmware update from local storage sequence.

##### Returns

valid firmware filename with full path or BLT\_NULL.

Referenced by FileTask().

#### 7.617.2.11 FileIsFirmwareUpdateRequestedHook()

```
blt_bool FileIsFirmwareUpdateRequestedHook (
 void)
```

Callback that gets called to check whether a firmware update from local file storage should be started. This could for example be when a switch is pressed, when a certain file is found on the local file storage, etc.

##### Returns

BLT\_TRUE if a firmware update is requested, BLT\_FALSE otherwise.

Referenced by FileHandleFirmwareUpdateRequest().

#### 7.617.2.12 NvmDoneHook()

```
blt_bool NvmDoneHook (
 void)
```

Callback that gets called at the end of the NVM programming session.

##### Returns

BLT\_TRUE is successful, BLT\_FALSE otherwise.

#### 7.617.2.13 NvmEraseHook()

```
blt_int8u NvmEraseHook (
 blt_addr addr,
 blt_int32u len)
```

Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the memory hasn't been erased yet.

## Parameters

|             |                  |
|-------------|------------------|
| <i>addr</i> | Start address.   |
| <i>len</i>  | Length in bytes. |

## Returns

BLT\_NVM\_OKAY if successful, BLT\_NVM\_NOT\_IN\_RANGE if the address is not within the supported memory range, or BLT\_NVM\_ERROR if the erase operation failed.

## 7.617.2.14 NvmInitHook()

```
void NvmInitHook (
 void)
```

Callback that gets called at the start of the internal NVM driver initialization routine.

## Returns

none.

## 7.617.2.15 NvmReinitHook()

```
void NvmReinitHook (
 void)
```

Callback that gets called at the start of a firmware update to reinitialize the NVM driver.

## Returns

none.

## 7.617.2.16 NvmWriteHook()

```
blt_int8u NvmWriteHook (
 blt_addr addr,
 blt_int32u len,
 blt_int8u * data)
```

Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the data hasn't been written yet.

**Parameters**

|             |                             |
|-------------|-----------------------------|
| <i>addr</i> | Start address.              |
| <i>len</i>  | Length in bytes.            |
| <i>data</i> | Pointer to the data buffer. |

**Returns**

BLT\_NVM\_OKAY if successful, BLT\_NVM\_NOT\_IN\_RANGE if the address is not within the supported memory range, or BLT\_NVM\_ERROR if the write operation failed.

**7.617.2.17 XcpGetSeedHook()**

```
blt_int8u XcpGetSeedHook (
 blt_int8u resource,
 blt_int8u * seed)
```

Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the GET\_SEED command.

**Parameters**

|                 |                                                        |
|-----------------|--------------------------------------------------------|
| <i>resource</i> | Resource that the seed is requested for (XCP_RES_XXX). |
| <i>seed</i>     | Pointer to byte buffer where the seed will be stored.  |

**Returns**

Length of the seed in bytes.

**7.617.2.18 XcpVerifyKeyHook()**

```
blt_int8u XcpVerifyKeyHook (
 blt_int8u resource,
 blt_int8u * key,
 blt_int8u len)
```

Called by the UNLOCK command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

**Parameters**

|                 |                                             |
|-----------------|---------------------------------------------|
| <i>resource</i> | resource to unlock (XCP_RES_XXX).           |
| <i>key</i>      | pointer to the byte buffer holding the key. |
| <i>len</i>      | length of the key in bytes.                 |



**Returns**

1 if the key was correct, 0 otherwise.

**7.617.3 Variable Documentation****7.617.3.1 canUse**

`blt_bool` canUse

Flag to indicate if the log-file can be used.

**7.617.3.2 handle**

FIL handle

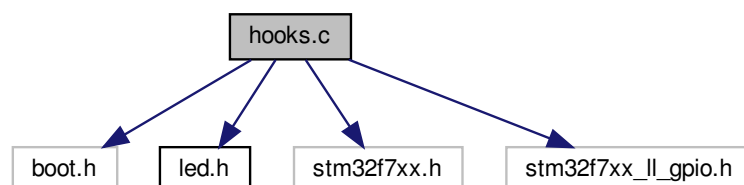
FatFS handle to the log-file.

**7.618 hooks.c File Reference**

Bootloader callback source file.

```
#include "boot.h"
#include "led.h"
#include "stm32f7xx.h"
#include "stm32f7xx_ll_gpio.h"
```

Include dependency graph for ARMCM7\_STM32F7\_Nucleo\_F746ZG\_CubeIDE/Boot/App/hooks.c:



## Functions

- [blt\\_bool CpuUserProgramStartHook](#) (void)  
*Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.*
- void [CopInitHook](#) (void)  
*Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.*
- void [CopServiceHook](#) (void)  
*Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.*
- void [UsbConnectHook](#) (blt\_bool connect)  
*Callback that gets called whenever the USB device should be connected to the USB bus.*
- void [UsbEnterLowPowerModeHook](#) (void)  
*Callback that gets called whenever the USB host requests the device to enter a low power mode.*
- void [UsbLeaveLowPowerModeHook](#) (void)  
*Callback that gets called whenever the USB host requests the device to exit low power mode.*
- void [BackDoorInitHook](#) (void)  
*Initializes the backdoor entry option.*
- [blt\\_bool BackDoorEntryHook](#) (void)  
*Checks if a backdoor entry is requested.*
- void [NvmInitHook](#) (void)  
*Callback that gets called at the start of the internal NVM driver initialization routine.*
- void [NvmReinitHook](#) (void)  
*Callback that gets called at the start of a firmware update to reinitialize the NVM driver.*
- [blt\\_int8u NvmWriteHook](#) (blt\_addr addr, blt\_int32u len, blt\_int8u \*data)  
*Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the data hasn't been written yet.*
- [blt\\_int8u NvmEraseHook](#) (blt\_addr addr, blt\_int32u len)  
*Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the memory hasn't been erased yet.*
- [blt\\_bool NvmDoneHook](#) (void)  
*Callback that gets called at the end of the NVM programming session.*
- [blt\\_int8u XcpGetSeedHook](#) (blt\_int8u resource, blt\_int8u \*seed)  
*Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the GET\_SEED command.*
- [blt\\_int8u XcpVerifyKeyHook](#) (blt\_int8u resource, blt\_int8u \*key, blt\_int8u len)  
*Called by the UNLOCK command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.*

### 7.618.1 Detailed Description

Bootloader callback source file.

### 7.618.2 Function Documentation

### 7.618.2.1 BackDoorEntryHook()

```
blt_bool BackDoorEntryHook (
 void)
```

Checks if a backdoor entry is requested.

#### Returns

BLT\_TRUE if the backdoor entry is requested, BLT\_FALSE otherwise.

### 7.618.2.2 BackDoorInitHook()

```
void BackDoorInitHook (
 void)
```

Initializes the backdoor entry option.

#### Returns

none.

### 7.618.2.3 CopInitHook()

```
void CopInitHook (
 void)
```

Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.

#### Returns

none.

### 7.618.2.4 CopServiceHook()

```
void CopServiceHook (
 void)
```

Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.

#### Returns

none.

#### 7.618.2.5 CpuUserProgramStartHook()

```
blt_bool CpuUserProgramStartHook (
 void)
```

Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.

##### Returns

BLT\_TRUE if it is okay to start the user program, BLT\_FALSE to keep keep the bootloader active.

#### 7.618.2.6 NvmDoneHook()

```
blt_bool NvmDoneHook (
 void)
```

Callback that gets called at the end of the NVM programming session.

##### Returns

BLT\_TRUE is successful, BLT\_FALSE otherwise.

#### 7.618.2.7 NvmEraseHook()

```
blt_int8u NvmEraseHook (
 blt_addr addr,
 blt_int32u len)
```

Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the memory hasn't been erased yet.

##### Parameters

|             |                  |
|-------------|------------------|
| <i>addr</i> | Start address.   |
| <i>len</i>  | Length in bytes. |

##### Returns

BLT\_NVM\_OKAY if successful, BLT\_NVM\_NOT\_IN\_RANGE if the address is not within the supported memory range, or BLT\_NVM\_ERROR is the erase operation failed.

### 7.618.2.8 NvmInitHook()

```
void NvmInitHook (
 void)
```

Callback that gets called at the start of the internal NVM driver initialization routine.

#### Returns

none.

### 7.618.2.9 NvmReinitHook()

```
void NvmReinitHook (
 void)
```

Callback that gets called at the start of a firmware update to reinitialize the NVM driver.

#### Returns

none.

### 7.618.2.10 NvmWriteHook()

```
blt_int8u NvmWriteHook (
 blt_addr addr,
 blt_int32u len,
 blt_int8u * data)
```

Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the data hasn't been written yet.

#### Parameters

|             |                             |
|-------------|-----------------------------|
| <i>addr</i> | Start address.              |
| <i>len</i>  | Length in bytes.            |
| <i>data</i> | Pointer to the data buffer. |

#### Returns

BLT\_NVM\_OKAY if successful, BLT\_NVM\_NOT\_IN\_RANGE if the address is not within the supported memory range, or BLT\_NVM\_ERROR if the write operation failed.

#### 7.618.2.11 UsbConnectHook()

```
void UsbConnectHook (
 blt_bool connect)
```

Callback that gets called whenever the USB device should be connected to the USB bus.

##### Parameters

|                |                                                  |
|----------------|--------------------------------------------------|
| <i>connect</i> | BLT_TRUE to connect and BLT_FALSE to disconnect. |
|----------------|--------------------------------------------------|

##### Returns

none.

#### 7.618.2.12 UsbEnterLowPowerModeHook()

```
void UsbEnterLowPowerModeHook (
 void)
```

Callback that gets called whenever the USB host requests the device to enter a low power mode.

##### Returns

none.

#### 7.618.2.13 UsbLeaveLowPowerModeHook()

```
void UsbLeaveLowPowerModeHook (
 void)
```

Callback that gets called whenever the USB host requests the device to exit low power mode.

##### Returns

none.

#### 7.618.2.14 XcpGetSeedHook()

```
blt_int8u XcpGetSeedHook (
 blt_int8u resource,
 blt_int8u * seed)
```

Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the GET\_SEED command.

## Parameters

|                 |                                                        |
|-----------------|--------------------------------------------------------|
| <i>resource</i> | Resource that the seed is requested for (XCP_RES_XXX). |
| <i>seed</i>     | Pointer to byte buffer where the seed will be stored.  |

## Returns

Length of the seed in bytes.

## 7.618.2.15 XcpVerifyKeyHook()

```
blt_int8u XcpVerifyKeyHook (
 blt_int8u resource,
 blt_int8u * key,
 blt_int8u len)
```

Called by the UNLOCK command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

## Parameters

|                 |                                             |
|-----------------|---------------------------------------------|
| <i>resource</i> | resource to unlock (XCP_RES_XXX).           |
| <i>key</i>      | pointer to the byte buffer holding the key. |
| <i>len</i>      | length of the key in bytes.                 |

## Returns

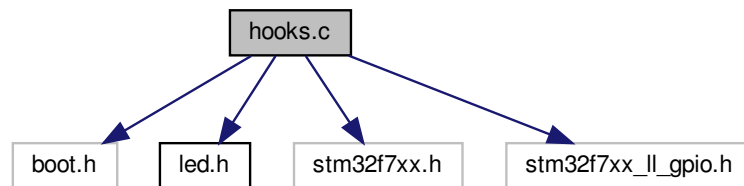
1 if the key was correct, 0 otherwise.

## 7.619 hooks.c File Reference

Bootloader callback source file.

```
#include "boot.h"
#include "led.h"
#include "stm32f7xx.h"
#include "stm32f7xx_ll_gpio.h"
```

Include dependency graph for ARMC7\_STM32F7\_Nucleo\_F746ZG\_GCC/Boot/hooks.c:



## Functions

- [blt\\_bool CpuUserProgramStartHook](#) (void)  
*Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.*
- void [CopInitHook](#) (void)  
*Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.*
- void [CopServiceHook](#) (void)  
*Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.*
- void [UsbConnectHook](#) (blt\_bool connect)  
*Callback that gets called whenever the USB device should be connected to the USB bus.*
- void [UsbEnterLowPowerModeHook](#) (void)  
*Callback that gets called whenever the USB host requests the device to enter a low power mode.*
- void [UsbLeaveLowPowerModeHook](#) (void)  
*Callback that gets called whenever the USB host requests the device to exit low power mode.*
- void [BackDoorInitHook](#) (void)  
*Initializes the backdoor entry option.*
- [blt\\_bool BackDoorEntryHook](#) (void)  
*Checks if a backdoor entry is requested.*
- void [NvmInitHook](#) (void)  
*Callback that gets called at the start of the internal NVM driver initialization routine.*
- void [NvmReinitHook](#) (void)  
*Callback that gets called at the start of a firmware update to reinitialize the NVM driver.*
- [blt\\_int8u NvmWriteHook](#) (blt\_addr addr, blt\_int32u len, blt\_int8u \*data)  
*Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the data hasn't been written yet.*
- [blt\\_int8u NvmEraseHook](#) (blt\_addr addr, blt\_int32u len)  
*Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the memory hasn't been erased yet.*
- [blt\\_bool NvmDoneHook](#) (void)  
*Callback that gets called at the end of the NVM programming session.*
- [blt\\_int8u XcpGetSeedHook](#) (blt\_int8u resource, blt\_int8u \*seed)



*Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the GET\_SEED command.*

- `blt_int8u XcpVerifyKeyHook (blt_int8u resource, blt_int8u *key, blt_int8u len)`

*Called by the UNLOCK command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.*

## 7.619.1 Detailed Description

Bootloader callback source file.

## 7.619.2 Function Documentation

### 7.619.2.1 BackDoorEntryHook()

```
blt_bool BackDoorEntryHook (
 void)
```

Checks if a backdoor entry is requested.

#### Returns

BLT\_TRUE if the backdoor entry is requested, BLT\_FALSE otherwise.

### 7.619.2.2 BackDoorInitHook()

```
void BackDoorInitHook (
 void)
```

Initializes the backdoor entry option.

#### Returns

none.

### 7.619.2.3 CopInitHook()

```
void CopInitHook (
 void)
```

Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.

#### Returns

none.

#### 7.619.2.4 CopServiceHook()

```
void CopServiceHook (
 void)
```

Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.

##### Returns

none.

#### 7.619.2.5 CpuUserProgramStartHook()

```
blt_bool CpuUserProgramStartHook (
 void)
```

Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.

##### Returns

BLT\_TRUE if it is okay to start the user program, BLT\_FALSE to keep keep the bootloader active.

#### 7.619.2.6 NvmDoneHook()

```
blt_bool NvmDoneHook (
 void)
```

Callback that gets called at the end of the NVM programming session.

##### Returns

BLT\_TRUE is successful, BLT\_FALSE otherwise.

#### 7.619.2.7 NvmEraseHook()

```
blt_int8u NvmEraseHook (
 blt_addr addr,
 blt_int32u len)
```

Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the memory hasn't been erased yet.

## Parameters

|             |                  |
|-------------|------------------|
| <i>addr</i> | Start address.   |
| <i>len</i>  | Length in bytes. |

## Returns

BLT\_NVM\_OKAY if successful, BLT\_NVM\_NOT\_IN\_RANGE if the address is not within the supported memory range, or BLT\_NVM\_ERROR if the erase operation failed.

## 7.619.2.8 NvmInitHook()

```
void NvmInitHook (
 void)
```

Callback that gets called at the start of the internal NVM driver initialization routine.

## Returns

none.

## 7.619.2.9 NvmReinitHook()

```
void NvmReinitHook (
 void)
```

Callback that gets called at the start of a firmware update to reinitialize the NVM driver.

## Returns

none.

## 7.619.2.10 NvmWriteHook()

```
blt_int8u NvmWriteHook (
 blt_addr addr,
 blt_int32u len,
 blt_int8u * data)
```

Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the data hasn't been written yet.

**Parameters**

|             |                             |
|-------------|-----------------------------|
| <i>addr</i> | Start address.              |
| <i>len</i>  | Length in bytes.            |
| <i>data</i> | Pointer to the data buffer. |

**Returns**

BLT\_NVM\_OKAY if successful, BLT\_NVM\_NOT\_IN\_RANGE if the address is not within the supported memory range, or BLT\_NVM\_ERROR if the write operation failed.

**7.619.2.11 UsbConnectHook()**

```
void UsbConnectHook (
 blt_bool connect)
```

Callback that gets called whenever the USB device should be connected to the USB bus.

**Parameters**

|                |                                                  |
|----------------|--------------------------------------------------|
| <i>connect</i> | BLT_TRUE to connect and BLT_FALSE to disconnect. |
|----------------|--------------------------------------------------|

**Returns**

none.

**7.619.2.12 UsbEnterLowPowerModeHook()**

```
void UsbEnterLowPowerModeHook (
 void)
```

Callback that gets called whenever the USB host requests the device to enter a low power mode.

**Returns**

none.

**7.619.2.13 UsbLeaveLowPowerModeHook()**

```
void UsbLeaveLowPowerModeHook (
 void)
```

Callback that gets called whenever the USB host requests the device to exit low power mode.

**Returns**

none.

#### 7.619.2.14 XcpGetSeedHook()

```
blt_int8u XcpGetSeedHook (
 blt_int8u resource,
 blt_int8u * seed)
```

Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the GET\_SEED command.

##### Parameters

|                 |                                                        |
|-----------------|--------------------------------------------------------|
| <i>resource</i> | Resource that the seed is requested for (XCP_RES_XXX). |
| <i>seed</i>     | Pointer to byte buffer where the seed will be stored.  |

##### Returns

Length of the seed in bytes.

#### 7.619.2.15 XcpVerifyKeyHook()

```
blt_int8u XcpVerifyKeyHook (
 blt_int8u resource,
 blt_int8u * key,
 blt_int8u len)
```

Called by the UNLOCK command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

##### Parameters

|                 |                                             |
|-----------------|---------------------------------------------|
| <i>resource</i> | resource to unlock (XCP_RES_XXX).           |
| <i>key</i>      | pointer to the byte buffer holding the key. |
| <i>len</i>      | length of the key in bytes.                 |

##### Returns

1 if the key was correct, 0 otherwise.

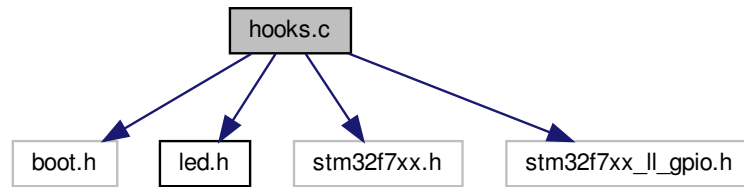
## 7.620 hooks.c File Reference

Bootloader callback source file.

```
#include "boot.h"
#include "led.h"
#include "stm32f7xx.h"
```

```
#include "stm32f7xx_ll_gpio.h"
```

Include dependency graph for ARMC7\_STM32F7\_Nucleo\_F746ZG\_IAR/Boot/hooks.c:



## Functions

- [blt\\_bool CpuUserProgramStartHook](#) (void)
 

*Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.*
- void [CopInitHook](#) (void)
 

*Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.*
- void [CopServiceHook](#) (void)
 

*Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.*
- void [UsbConnectHook](#) (blt\_bool connect)
 

*Callback that gets called whenever the USB device should be connected to the USB bus.*
- void [UsbEnterLowPowerModeHook](#) (void)
 

*Callback that gets called whenever the USB host requests the device to enter a low power mode.*
- void [UsbLeaveLowPowerModeHook](#) (void)
 

*Callback that gets called whenever the USB host requests the device to exit low power mode.*
- void [BackDoorInitHook](#) (void)
 

*Initializes the backdoor entry option.*
- [blt\\_bool BackDoorEntryHook](#) (void)
 

*Checks if a backdoor entry is requested.*
- void [NvmInitHook](#) (void)
 

*Callback that gets called at the start of the internal NVM driver initialization routine.*
- void [NvmReinitHook](#) (void)
 

*Callback that gets called at the start of a firmware update to reinitialize the NVM driver.*
- [blt\\_int8u NvmWriteHook](#) (blt\_addr addr, blt\_int32u len, blt\_int8u \*data)
 

*Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the data hasn't been written yet.*
- [blt\\_int8u NvmEraseHook](#) (blt\_addr addr, blt\_int32u len)
 

*Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the memory hasn't been erased yet.*
- [blt\\_bool NvmDoneHook](#) (void)
 

*Callback that gets called at the end of the NVM programming session.*
- [blt\\_int8u XcpGetSeedHook](#) (blt\_int8u resource, blt\_int8u \*seed)

*Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the GET\_SEED command.*

- `blt_int8u XcpVerifyKeyHook (blt_int8u resource, blt_int8u *key, blt_int8u len)`

*Called by the UNLOCK command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.*

## 7.620.1 Detailed Description

Bootloader callback source file.

## 7.620.2 Function Documentation

### 7.620.2.1 BackDoorEntryHook()

```
blt_bool BackDoorEntryHook (
 void)
```

Checks if a backdoor entry is requested.

#### Returns

BLT\_TRUE if the backdoor entry is requested, BLT\_FALSE otherwise.

### 7.620.2.2 BackDoorInitHook()

```
void BackDoorInitHook (
 void)
```

Initializes the backdoor entry option.

#### Returns

none.

### 7.620.2.3 CopInitHook()

```
void CopInitHook (
 void)
```

Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.

#### Returns

none.

#### 7.620.2.4 CopServiceHook()

```
void CopServiceHook (
 void)
```

Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.

##### Returns

none.

#### 7.620.2.5 CpuUserProgramStartHook()

```
blt_bool CpuUserProgramStartHook (
 void)
```

Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.

##### Returns

BLT\_TRUE if it is okay to start the user program, BLT\_FALSE to keep keep the bootloader active.

#### 7.620.2.6 NvmDoneHook()

```
blt_bool NvmDoneHook (
 void)
```

Callback that gets called at the end of the NVM programming session.

##### Returns

BLT\_TRUE is successful, BLT\_FALSE otherwise.

#### 7.620.2.7 NvmEraseHook()

```
blt_int8u NvmEraseHook (
 blt_addr addr,
 blt_int32u len)
```

Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the memory hasn't been erased yet.



**Parameters**

|             |                  |
|-------------|------------------|
| <i>addr</i> | Start address.   |
| <i>len</i>  | Length in bytes. |

**Returns**

BLT\_NVM\_OKAY if successful, BLT\_NVM\_NOT\_IN\_RANGE if the address is not within the supported memory range, or BLT\_NVM\_ERROR if the erase operation failed.

**7.620.2.8 NvmInitHook()**

```
void NvmInitHook (
 void)
```

Callback that gets called at the start of the internal NVM driver initialization routine.

**Returns**

none.

**7.620.2.9 NvmReinitHook()**

```
void NvmReinitHook (
 void)
```

Callback that gets called at the start of a firmware update to reinitialize the NVM driver.

**Returns**

none.

**7.620.2.10 NvmWriteHook()**

```
blt_int8u NvmWriteHook (
 blt_addr addr,
 blt_int32u len,
 blt_int8u * data)
```

Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the data hasn't been written yet.

**Parameters**

|             |                             |
|-------------|-----------------------------|
| <i>addr</i> | Start address.              |
| <i>len</i>  | Length in bytes.            |
| <i>data</i> | Pointer to the data buffer. |

**Returns**

BLT\_NVM\_OKAY if successful, BLT\_NVM\_NOT\_IN\_RANGE if the address is not within the supported memory range, or BLT\_NVM\_ERROR if the write operation failed.

**7.620.2.11 UsbConnectHook()**

```
void UsbConnectHook (
 blt_bool connect)
```

Callback that gets called whenever the USB device should be connected to the USB bus.

**Parameters**

|                |                                                  |
|----------------|--------------------------------------------------|
| <i>connect</i> | BLT_TRUE to connect and BLT_FALSE to disconnect. |
|----------------|--------------------------------------------------|

**Returns**

none.

**7.620.2.12 UsbEnterLowPowerModeHook()**

```
void UsbEnterLowPowerModeHook (
 void)
```

Callback that gets called whenever the USB host requests the device to enter a low power mode.

**Returns**

none.

**7.620.2.13 UsbLeaveLowPowerModeHook()**

```
void UsbLeaveLowPowerModeHook (
 void)
```

Callback that gets called whenever the USB host requests the device to exit low power mode.

**Returns**

none.

#### 7.620.2.14 XcpGetSeedHook()

```
blt_int8u XcpGetSeedHook (
 blt_int8u resource,
 blt_int8u * seed)
```

Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the GET\_SEED command.

##### Parameters

|                 |                                                        |
|-----------------|--------------------------------------------------------|
| <i>resource</i> | Resource that the seed is requested for (XCP_RES_XXX). |
| <i>seed</i>     | Pointer to byte buffer where the seed will be stored.  |

##### Returns

Length of the seed in bytes.

#### 7.620.2.15 XcpVerifyKeyHook()

```
blt_int8u XcpVerifyKeyHook (
 blt_int8u resource,
 blt_int8u * key,
 blt_int8u len)
```

Called by the UNLOCK command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

##### Parameters

|                 |                                             |
|-----------------|---------------------------------------------|
| <i>resource</i> | resource to unlock (XCP_RES_XXX).           |
| <i>key</i>      | pointer to the byte buffer holding the key. |
| <i>len</i>      | length of the key in bytes.                 |

##### Returns

1 if the key was correct, 0 otherwise.

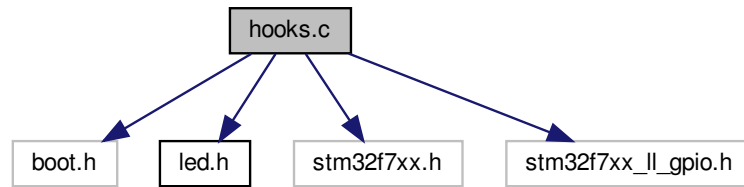
## 7.621 hooks.c File Reference

Bootloader callback source file.

```
#include "boot.h"
#include "led.h"
#include "stm32f7xx.h"
```

```
#include "stm32f7xx_ll_gpio.h"
```

Include dependency graph for ARMC7\_STM32F7\_Nucleo\_F746ZG\_Keil/Boot/hooks.c:



## Functions

- [blt\\_bool CpuUserProgramStartHook](#) (void)
 

*Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.*
- void [CopInitHook](#) (void)
 

*Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.*
- void [CopServiceHook](#) (void)
 

*Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.*
- void [UsbConnectHook](#) (blt\_bool connect)
 

*Callback that gets called whenever the USB device should be connected to the USB bus.*
- void [UsbEnterLowPowerModeHook](#) (void)
 

*Callback that gets called whenever the USB host requests the device to enter a low power mode.*
- void [UsbLeaveLowPowerModeHook](#) (void)
 

*Callback that gets called whenever the USB host requests the device to exit low power mode.*
- void [BackDoorInitHook](#) (void)
 

*Initializes the backdoor entry option.*
- [blt\\_bool BackDoorEntryHook](#) (void)
 

*Checks if a backdoor entry is requested.*
- void [NvmInitHook](#) (void)
 

*Callback that gets called at the start of the internal NVM driver initialization routine.*
- void [NvmReinitHook](#) (void)
 

*Callback that gets called at the start of a firmware update to reinitialize the NVM driver.*
- [blt\\_int8u NvmWriteHook](#) (blt\_addr addr, blt\_int32u len, blt\_int8u \*data)
 

*Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the data hasn't been written yet.*
- [blt\\_int8u NvmEraseHook](#) (blt\_addr addr, blt\_int32u len)
 

*Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the memory hasn't been erased yet.*
- [blt\\_bool NvmDoneHook](#) (void)
 

*Callback that gets called at the end of the NVM programming session.*
- [blt\\_int8u XcpGetSeedHook](#) (blt\_int8u resource, blt\_int8u \*seed)

*Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the GET\_SEED command.*

- `blt_int8u XcpVerifyKeyHook (blt_int8u resource, blt_int8u *key, blt_int8u len)`

*Called by the UNLOCK command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.*

## 7.621.1 Detailed Description

Bootloader callback source file.

## 7.621.2 Function Documentation

### 7.621.2.1 BackDoorEntryHook()

```
blt_bool BackDoorEntryHook (
 void)
```

Checks if a backdoor entry is requested.

#### Returns

BLT\_TRUE if the backdoor entry is requested, BLT\_FALSE otherwise.

### 7.621.2.2 BackDoorInitHook()

```
void BackDoorInitHook (
 void)
```

Initializes the backdoor entry option.

#### Returns

none.

### 7.621.2.3 CopInitHook()

```
void CopInitHook (
 void)
```

Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.

#### Returns

none.

#### 7.621.2.4 CopServiceHook()

```
void CopServiceHook (
 void)
```

Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.

##### Returns

none.

#### 7.621.2.5 CpuUserProgramStartHook()

```
blt_bool CpuUserProgramStartHook (
 void)
```

Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.

##### Returns

BLT\_TRUE if it is okay to start the user program, BLT\_FALSE to keep keep the bootloader active.

#### 7.621.2.6 NvmDoneHook()

```
blt_bool NvmDoneHook (
 void)
```

Callback that gets called at the end of the NVM programming session.

##### Returns

BLT\_TRUE is successful, BLT\_FALSE otherwise.

#### 7.621.2.7 NvmEraseHook()

```
blt_int8u NvmEraseHook (
 blt_addr addr,
 blt_int32u len)
```

Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the memory hasn't been erased yet.

**Parameters**

|             |                  |
|-------------|------------------|
| <i>addr</i> | Start address.   |
| <i>len</i>  | Length in bytes. |

**Returns**

BLT\_NVM\_OKAY if successful, BLT\_NVM\_NOT\_IN\_RANGE if the address is not within the supported memory range, or BLT\_NVM\_ERROR if the erase operation failed.

**7.621.2.8 NvmInitHook()**

```
void NvmInitHook (
 void)
```

Callback that gets called at the start of the internal NVM driver initialization routine.

**Returns**

none.

**7.621.2.9 NvmReinitHook()**

```
void NvmReinitHook (
 void)
```

Callback that gets called at the start of a firmware update to reinitialize the NVM driver.

**Returns**

none.

**7.621.2.10 NvmWriteHook()**

```
blt_int8u NvmWriteHook (
 blt_addr addr,
 blt_int32u len,
 blt_int8u * data)
```

Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the data hasn't been written yet.

**Parameters**

|             |                             |
|-------------|-----------------------------|
| <i>addr</i> | Start address.              |
| <i>len</i>  | Length in bytes.            |
| <i>data</i> | Pointer to the data buffer. |

**Returns**

BLT\_NVM\_OKAY if successful, BLT\_NVM\_NOT\_IN\_RANGE if the address is not within the supported memory range, or BLT\_NVM\_ERROR if the write operation failed.

**7.621.2.11 UsbConnectHook()**

```
void UsbConnectHook (
 blt_bool connect)
```

Callback that gets called whenever the USB device should be connected to the USB bus.

**Parameters**

|                |                                                  |
|----------------|--------------------------------------------------|
| <i>connect</i> | BLT_TRUE to connect and BLT_FALSE to disconnect. |
|----------------|--------------------------------------------------|

**Returns**

none.

**7.621.2.12 UsbEnterLowPowerModeHook()**

```
void UsbEnterLowPowerModeHook (
 void)
```

Callback that gets called whenever the USB host requests the device to enter a low power mode.

**Returns**

none.

**7.621.2.13 UsbLeaveLowPowerModeHook()**

```
void UsbLeaveLowPowerModeHook (
 void)
```

Callback that gets called whenever the USB host requests the device to exit low power mode.

**Returns**

none.



#### 7.621.2.14 XcpGetSeedHook()

```
blt_int8u XcpGetSeedHook (
 blt_int8u resource,
 blt_int8u * seed)
```

Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the GET\_SEED command.

##### Parameters

|                 |                                                        |
|-----------------|--------------------------------------------------------|
| <i>resource</i> | Resource that the seed if requested for (XCP_RES_XXX). |
| <i>seed</i>     | Pointer to byte buffer wher the seed will be stored.   |

##### Returns

Length of the seed in bytes.

#### 7.621.2.15 XcpVerifyKeyHook()

```
blt_int8u XcpVerifyKeyHook (
 blt_int8u resource,
 blt_int8u * key,
 blt_int8u len)
```

Called by the UNLOCK command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

##### Parameters

|                 |                                             |
|-----------------|---------------------------------------------|
| <i>resource</i> | resource to unlock (XCP_RES_XXX).           |
| <i>key</i>      | pointer to the byte buffer holding the key. |
| <i>len</i>      | length of the key in bytes.                 |

##### Returns

1 if the key was correct, 0 otherwise.

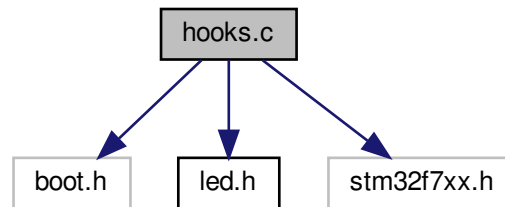
## 7.622 hooks.c File Reference

Bootloader callback source file.

```
#include "boot.h"
#include "led.h"
```

```
#include "stm32f7xx.h"
```

Include dependency graph for ARMCM7\_STM32F7\_Nucleo\_F767ZI\_CubeIDE/Boot/App/hooks.c:



## Functions

- [blt\\_bool CpuUserProgramStartHook](#) (void)  
*Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.*
- void [CopInitHook](#) (void)  
*Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.*
- void [CopServiceHook](#) (void)  
*Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.*
- void [BackDoorInitHook](#) (void)  
*Initializes the backdoor entry option.*
- [blt\\_bool BackDoorEntryHook](#) (void)  
*Checks if a backdoor entry is requested.*
- void [NvmInitHook](#) (void)  
*Callback that gets called at the start of the internal NVM driver initialization routine.*
- void [NvmReinitHook](#) (void)  
*Callback that gets called at the start of a firmware update to reinitialize the NVM driver.*
- [blt\\_int8u NvmWriteHook](#) ([blt\\_addr](#) addr, [blt\\_int32u](#) len, [blt\\_int8u](#) \*data)  
*Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the data hasn't been written yet.*
- [blt\\_int8u NvmEraseHook](#) ([blt\\_addr](#) addr, [blt\\_int32u](#) len)  
*Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the memory hasn't been erased yet.*
- [blt\\_bool NvmDoneHook](#) (void)  
*Callback that gets called at the end of the NVM programming session.*
- [blt\\_int8u XcpGetSeedHook](#) ([blt\\_int8u](#) resource, [blt\\_int8u](#) \*seed)  
*Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the GET\_SEED command.*
- [blt\\_int8u XcpVerifyKeyHook](#) ([blt\\_int8u](#) resource, [blt\\_int8u](#) \*key, [blt\\_int8u](#) len)  
*Called by the UNLOCK command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.*

## 7.622.1 Detailed Description

Bootloader callback source file.

## 7.622.2 Function Documentation

### 7.622.2.1 BackDoorEntryHook()

```
blt_bool BackDoorEntryHook (
 void)
```

Checks if a backdoor entry is requested.

#### Returns

BLT\_TRUE if the backdoor entry is requested, BLT\_FALSE otherwise.

### 7.622.2.2 BackDoorInitHook()

```
void BackDoorInitHook (
 void)
```

Initializes the backdoor entry option.

#### Returns

none.

### 7.622.2.3 CopInitHook()

```
void CopInitHook (
 void)
```

Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.

#### Returns

none.

#### 7.622.2.4 CopServiceHook()

```
void CopServiceHook (
 void)
```

Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.

##### Returns

none.

#### 7.622.2.5 CpuUserProgramStartHook()

```
blt_bool CpuUserProgramStartHook (
 void)
```

Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.

##### Returns

BLT\_TRUE if it is okay to start the user program, BLT\_FALSE to keep keep the bootloader active.

#### 7.622.2.6 NvmDoneHook()

```
blt_bool NvmDoneHook (
 void)
```

Callback that gets called at the end of the NVM programming session.

##### Returns

BLT\_TRUE is successful, BLT\_FALSE otherwise.

#### 7.622.2.7 NvmEraseHook()

```
blt_int8u NvmEraseHook (
 blt_addr addr,
 blt_int32u len)
```

Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the memory hasn't been erased yet.

## Parameters

|             |                  |
|-------------|------------------|
| <i>addr</i> | Start address.   |
| <i>len</i>  | Length in bytes. |

## Returns

BLT\_NVM\_OKAY if successful, BLT\_NVM\_NOT\_IN\_RANGE if the address is not within the supported memory range, or BLT\_NVM\_ERROR if the erase operation failed.

## 7.622.2.8 NvmInitHook()

```
void NvmInitHook (
 void)
```

Callback that gets called at the start of the internal NVM driver initialization routine.

## Returns

none.

## 7.622.2.9 NvmReinitHook()

```
void NvmReinitHook (
 void)
```

Callback that gets called at the start of a firmware update to reinitialize the NVM driver.

## Returns

none.

## 7.622.2.10 NvmWriteHook()

```
blt_int8u NvmWriteHook (
 blt_addr addr,
 blt_int32u len,
 blt_int8u * data)
```

Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the data hasn't been written yet.

**Parameters**

|             |                             |
|-------------|-----------------------------|
| <i>addr</i> | Start address.              |
| <i>len</i>  | Length in bytes.            |
| <i>data</i> | Pointer to the data buffer. |

**Returns**

BLT\_NVM\_OKAY if successful, BLT\_NVM\_NOT\_IN\_RANGE if the address is not within the supported memory range, or BLT\_NVM\_ERROR if the write operation failed.

**7.622.2.11 XcpGetSeedHook()**

```
blt_int8u XcpGetSeedHook (
 blt_int8u resource,
 blt_int8u * seed)
```

Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the GET\_SEED command.

**Parameters**

|                 |                                                        |
|-----------------|--------------------------------------------------------|
| <i>resource</i> | Resource that the seed is requested for (XCP_RES_XXX). |
| <i>seed</i>     | Pointer to byte buffer where the seed will be stored.  |

**Returns**

Length of the seed in bytes.

**7.622.2.12 XcpVerifyKeyHook()**

```
blt_int8u XcpVerifyKeyHook (
 blt_int8u resource,
 blt_int8u * key,
 blt_int8u len)
```

Called by the UNLOCK command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

**Parameters**

|                 |                                             |
|-----------------|---------------------------------------------|
| <i>resource</i> | resource to unlock (XCP_RES_XXX).           |
| <i>key</i>      | pointer to the byte buffer holding the key. |
| <i>len</i>      | length of the key in bytes.                 |

## Returns

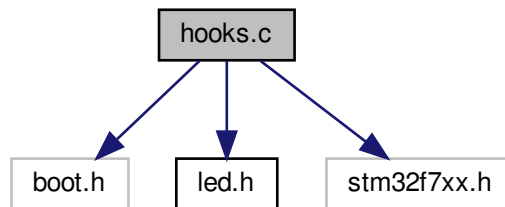
1 if the key was correct, 0 otherwise.

## 7.623 hooks.c File Reference

Bootloader callback source file.

```
#include "boot.h"
#include "led.h"
#include "stm32f7xx.h"
```

Include dependency graph for ARMCM7\_STM32F7\_Nucleo\_F767ZI\_GCC/Boot/hooks.c:



## Functions

- [blt\\_bool CpuUserProgramStartHook](#) (void)
 

*Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.*
- void [CopInitHook](#) (void)
 

*Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.*
- void [CopServiceHook](#) (void)
 

*Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.*
- void [BackDoorInitHook](#) (void)
 

*Initializes the backdoor entry option.*
- [blt\\_bool BackDoorEntryHook](#) (void)
 

*Checks if a backdoor entry is requested.*
- void [NvmInitHook](#) (void)
 

*Callback that gets called at the start of the internal NVM driver initialization routine.*
- void [NvmReinitHook](#) (void)
 

*Callback that gets called at the start of a firmware update to reinitialize the NVM driver.*
- [blt\\_int8u NvmWriteHook](#) ([blt\\_addr](#) addr, [blt\\_int32u](#) len, [blt\\_int8u](#) \*data)
 

*Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the data hasn't been written yet.*
- [blt\\_int8u NvmEraseHook](#) ([blt\\_addr](#) addr, [blt\\_int32u](#) len)

Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then `BLT_NVM_NOT_IN_RANGE` must be returned to indicate that the memory hasn't been erased yet.

- `blt_bool NvmDoneHook` (void)

Callback that gets called at the end of the NVM programming session.

- `blt_int8u XcpGetSeedHook` (`blt_int8u` resource, `blt_int8u *`seed)

Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the `GET_SEED` command.

- `blt_int8u XcpVerifyKeyHook` (`blt_int8u` resource, `blt_int8u *`key, `blt_int8u` len)

Called by the `UNLOCK` command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

### 7.623.1 Detailed Description

Bootloader callback source file.

### 7.623.2 Function Documentation

#### 7.623.2.1 BackDoorEntryHook()

```
blt_bool BackDoorEntryHook (
 void)
```

Checks if a backdoor entry is requested.

#### Returns

`BLT_TRUE` if the backdoor entry is requested, `BLT_FALSE` otherwise.

#### 7.623.2.2 BackDoorInitHook()

```
void BackDoorInitHook (
 void)
```

Initializes the backdoor entry option.

#### Returns

none.



### 7.623.2.3 CopInitHook()

```
void CopInitHook (
 void)
```

Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.

#### Returns

none.

### 7.623.2.4 CopServiceHook()

```
void CopServiceHook (
 void)
```

Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.

#### Returns

none.

### 7.623.2.5 CpuUserProgramStartHook()

```
blt_bool CpuUserProgramStartHook (
 void)
```

Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.

#### Returns

BLT\_TRUE if it is okay to start the user program, BLT\_FALSE to keep keep the bootloader active.

### 7.623.2.6 NvmDoneHook()

```
blt_bool NvmDoneHook (
 void)
```

Callback that gets called at the end of the NVM programming session.

#### Returns

BLT\_TRUE is successful, BLT\_FALSE otherwise.

#### 7.623.2.7 NvmEraseHook()

```
blt_int8u NvmEraseHook (
 blt_addr addr,
 blt_int32u len)
```

Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the memory hasn't been erased yet.

##### Parameters

|             |                  |
|-------------|------------------|
| <i>addr</i> | Start address.   |
| <i>len</i>  | Length in bytes. |

##### Returns

BLT\_NVM\_OKAY if successful, BLT\_NVM\_NOT\_IN\_RANGE if the address is not within the supported memory range, or BLT\_NVM\_ERROR if the erase operation failed.

#### 7.623.2.8 NvmInitHook()

```
void NvmInitHook (
 void)
```

Callback that gets called at the start of the internal NVM driver initialization routine.

##### Returns

none.

#### 7.623.2.9 NvmReinitHook()

```
void NvmReinitHook (
 void)
```

Callback that gets called at the start of a firmware update to reinitialize the NVM driver.

##### Returns

none.

#### 7.623.2.10 NvmWriteHook()

```
blt_int8u NvmWriteHook (
 blt_addr addr,
 blt_int32u len,
 blt_int8u * data)
```

Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the data hasn't been written yet.

## Parameters

|             |                             |
|-------------|-----------------------------|
| <i>addr</i> | Start address.              |
| <i>len</i>  | Length in bytes.            |
| <i>data</i> | Pointer to the data buffer. |

## Returns

BLT\_NVM\_OKAY if successful, BLT\_NVM\_NOT\_IN\_RANGE if the address is not within the supported memory range, or BLT\_NVM\_ERROR if the write operation failed.

## 7.623.2.11 XcpGetSeedHook()

```
blt_int8u XcpGetSeedHook (
 blt_int8u resource,
 blt_int8u * seed)
```

Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the GET\_SEED command.

## Parameters

|                 |                                                        |
|-----------------|--------------------------------------------------------|
| <i>resource</i> | Resource that the seed is requested for (XCP_RES_XXX). |
| <i>seed</i>     | Pointer to byte buffer where the seed will be stored.  |

## Returns

Length of the seed in bytes.

## 7.623.2.12 XcpVerifyKeyHook()

```
blt_int8u XcpVerifyKeyHook (
 blt_int8u resource,
 blt_int8u * key,
 blt_int8u len)
```

Called by the UNLOCK command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

## Parameters

|                 |                                             |
|-----------------|---------------------------------------------|
| <i>resource</i> | resource to unlock (XCP_RES_XXX).           |
| <i>key</i>      | pointer to the byte buffer holding the key. |
| <i>len</i>      | length of the key in bytes.                 |

## Returns

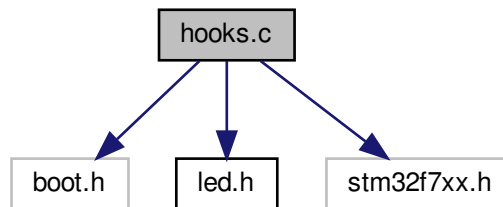
1 if the key was correct, 0 otherwise.

## 7.624 hooks.c File Reference

Bootloader callback source file.

```
#include "boot.h"
#include "led.h"
#include "stm32f7xx.h"
```

Include dependency graph for ARMCM7\_STM32F7\_Nucleo\_F767ZI\_IAR/Boot/hooks.c:



## Functions

- [blt\\_bool CpuUserProgramStartHook](#) (void)  
*Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.*
- void [CopInitHook](#) (void)  
*Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.*
- void [CopServiceHook](#) (void)  
*Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.*
- void [BackDoorInitHook](#) (void)  
*Initializes the backdoor entry option.*
- [blt\\_bool BackDoorEntryHook](#) (void)  
*Checks if a backdoor entry is requested.*
- void [NvmInitHook](#) (void)  
*Callback that gets called at the start of the internal NVM driver initialization routine.*
- void [NvmReinitHook](#) (void)  
*Callback that gets called at the start of a firmware update to reinitialize the NVM driver.*
- [blt\\_int8u NvmWriteHook](#) ([blt\\_addr](#) addr, [blt\\_int32u](#) len, [blt\\_int8u](#) \*data)  
*Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the data hasn't been written yet.*
- [blt\\_int8u NvmEraseHook](#) ([blt\\_addr](#) addr, [blt\\_int32u](#) len)

Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then `BLT_NVM_NOT_IN_RANGE` must be returned to indicate that the memory hasn't been erased yet.

- `blt_bool NvmDoneHook` (void)

Callback that gets called at the end of the NVM programming session.

- `blt_int8u XcpGetSeedHook` (`blt_int8u` resource, `blt_int8u *seed`)

Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the `GET_SEED` command.

- `blt_int8u XcpVerifyKeyHook` (`blt_int8u` resource, `blt_int8u *key`, `blt_int8u len`)

Called by the `UNLOCK` command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

## 7.624.1 Detailed Description

Bootloader callback source file.

## 7.624.2 Function Documentation

### 7.624.2.1 BackDoorEntryHook()

```
blt_bool BackDoorEntryHook (
 void)
```

Checks if a backdoor entry is requested.

#### Returns

`BLT_TRUE` if the backdoor entry is requested, `BLT_FALSE` otherwise.

### 7.624.2.2 BackDoorInitHook()

```
void BackDoorInitHook (
 void)
```

Initializes the backdoor entry option.

#### Returns

none.

#### 7.624.2.3 CopInitHook()

```
void CopInitHook (
 void)
```

Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.

##### Returns

none.

#### 7.624.2.4 CopServiceHook()

```
void CopServiceHook (
 void)
```

Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.

##### Returns

none.

#### 7.624.2.5 CpuUserProgramStartHook()

```
blt_bool CpuUserProgramStartHook (
 void)
```

Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.

##### Returns

BLT\_TRUE if it is okay to start the user program, BLT\_FALSE to keep keep the bootloader active.

#### 7.624.2.6 NvmDoneHook()

```
blt_bool NvmDoneHook (
 void)
```

Callback that gets called at the end of the NVM programming session.

##### Returns

BLT\_TRUE is successful, BLT\_FALSE otherwise.

### 7.624.2.7 NvmEraseHook()

```
blt_int8u NvmEraseHook (
 blt_addr addr,
 blt_int32u len)
```

Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the memory hasn't been erased yet.

#### Parameters

|             |                  |
|-------------|------------------|
| <i>addr</i> | Start address.   |
| <i>len</i>  | Length in bytes. |

#### Returns

BLT\_NVM\_OKAY if successful, BLT\_NVM\_NOT\_IN\_RANGE if the address is not within the supported memory range, or BLT\_NVM\_ERROR if the erase operation failed.

### 7.624.2.8 NvmInitHook()

```
void NvmInitHook (
 void)
```

Callback that gets called at the start of the internal NVM driver initialization routine.

#### Returns

none.

### 7.624.2.9 NvmReinitHook()

```
void NvmReinitHook (
 void)
```

Callback that gets called at the start of a firmware update to reinitialize the NVM driver.

#### Returns

none.

### 7.624.2.10 NvmWriteHook()

```
blt_int8u NvmWriteHook (
 blt_addr addr,
 blt_int32u len,
 blt_int8u * data)
```

Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the data hasn't been written yet.

**Parameters**

|             |                             |
|-------------|-----------------------------|
| <i>addr</i> | Start address.              |
| <i>len</i>  | Length in bytes.            |
| <i>data</i> | Pointer to the data buffer. |

**Returns**

BLT\_NVM\_OKAY if successful, BLT\_NVM\_NOT\_IN\_RANGE if the address is not within the supported memory range, or BLT\_NVM\_ERROR if the write operation failed.

**7.624.2.11 XcpGetSeedHook()**

```
blt_int8u XcpGetSeedHook (
 blt_int8u resource,
 blt_int8u * seed)
```

Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the GET\_SEED command.

**Parameters**

|                 |                                                        |
|-----------------|--------------------------------------------------------|
| <i>resource</i> | Resource that the seed is requested for (XCP_RES_XXX). |
| <i>seed</i>     | Pointer to byte buffer where the seed will be stored.  |

**Returns**

Length of the seed in bytes.

**7.624.2.12 XcpVerifyKeyHook()**

```
blt_int8u XcpVerifyKeyHook (
 blt_int8u resource,
 blt_int8u * key,
 blt_int8u len)
```

Called by the UNLOCK command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

**Parameters**

|                 |                                             |
|-----------------|---------------------------------------------|
| <i>resource</i> | resource to unlock (XCP_RES_XXX).           |
| <i>key</i>      | pointer to the byte buffer holding the key. |
| <i>len</i>      | length of the key in bytes.                 |



## Returns

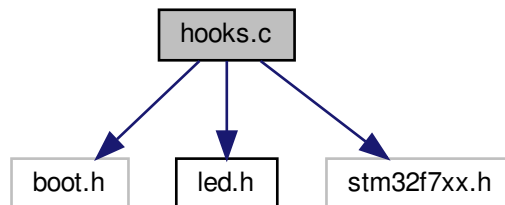
1 if the key was correct, 0 otherwise.

## 7.625 hooks.c File Reference

Bootloader callback source file.

```
#include "boot.h"
#include "led.h"
#include "stm32f7xx.h"
```

Include dependency graph for ARMCM7\_STM32F7\_Nucleo\_F767ZI\_Keil/Boot/hooks.c:



## Functions

- [blt\\_bool CpuUserProgramStartHook](#) (void)  
*Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.*
- void [CopInitHook](#) (void)  
*Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.*
- void [CopServiceHook](#) (void)  
*Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.*
- void [BackDoorInitHook](#) (void)  
*Initializes the backdoor entry option.*
- [blt\\_bool BackDoorEntryHook](#) (void)  
*Checks if a backdoor entry is requested.*
- void [NvmInitHook](#) (void)  
*Callback that gets called at the start of the internal NVM driver initialization routine.*
- void [NvmReinitHook](#) (void)  
*Callback that gets called at the start of a firmware update to reinitialize the NVM driver.*
- [blt\\_int8u NvmWriteHook](#) ([blt\\_addr](#) addr, [blt\\_int32u](#) len, [blt\\_int8u](#) \*data)  
*Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the data hasn't been written yet.*
- [blt\\_int8u NvmEraseHook](#) ([blt\\_addr](#) addr, [blt\\_int32u](#) len)

Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then `BLT_NVM_NOT_IN_RANGE` must be returned to indicate that the memory hasn't been erased yet.

- `blt_bool NvmDoneHook` (void)

Callback that gets called at the end of the NVM programming session.

- `blt_int8u XcpGetSeedHook` (`blt_int8u` resource, `blt_int8u *seed`)

Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the `GET_SEED` command.

- `blt_int8u XcpVerifyKeyHook` (`blt_int8u` resource, `blt_int8u *key`, `blt_int8u len`)

Called by the `UNLOCK` command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

### 7.625.1 Detailed Description

Bootloader callback source file.

### 7.625.2 Function Documentation

#### 7.625.2.1 BackDoorEntryHook()

```
blt_bool BackDoorEntryHook (
 void)
```

Checks if a backdoor entry is requested.

#### Returns

`BLT_TRUE` if the backdoor entry is requested, `BLT_FALSE` otherwise.

#### 7.625.2.2 BackDoorInitHook()

```
void BackDoorInitHook (
 void)
```

Initializes the backdoor entry option.

#### Returns

none.

### 7.625.2.3 CopInitHook()

```
void CopInitHook (
 void)
```

Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.

#### Returns

none.

### 7.625.2.4 CopServiceHook()

```
void CopServiceHook (
 void)
```

Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.

#### Returns

none.

### 7.625.2.5 CpuUserProgramStartHook()

```
blt_bool CpuUserProgramStartHook (
 void)
```

Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.

#### Returns

BLT\_TRUE if it is okay to start the user program, BLT\_FALSE to keep keep the bootloader active.

### 7.625.2.6 NvmDoneHook()

```
blt_bool NvmDoneHook (
 void)
```

Callback that gets called at the end of the NVM programming session.

#### Returns

BLT\_TRUE is successful, BLT\_FALSE otherwise.

#### 7.625.2.7 NvmEraseHook()

```
blt_int8u NvmEraseHook (
 blt_addr addr,
 blt_int32u len)
```

Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the memory hasn't been erased yet.

##### Parameters

|             |                  |
|-------------|------------------|
| <i>addr</i> | Start address.   |
| <i>len</i>  | Length in bytes. |

##### Returns

BLT\_NVM\_OKAY if successful, BLT\_NVM\_NOT\_IN\_RANGE if the address is not within the supported memory range, or BLT\_NVM\_ERROR if the erase operation failed.

#### 7.625.2.8 NvmInitHook()

```
void NvmInitHook (
 void)
```

Callback that gets called at the start of the internal NVM driver initialization routine.

##### Returns

none.

#### 7.625.2.9 NvmReinitHook()

```
void NvmReinitHook (
 void)
```

Callback that gets called at the start of a firmware update to reinitialize the NVM driver.

##### Returns

none.

#### 7.625.2.10 NvmWriteHook()

```
blt_int8u NvmWriteHook (
 blt_addr addr,
 blt_int32u len,
 blt_int8u * data)
```

Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the data hasn't been written yet.

**Parameters**

|             |                             |
|-------------|-----------------------------|
| <i>addr</i> | Start address.              |
| <i>len</i>  | Length in bytes.            |
| <i>data</i> | Pointer to the data buffer. |

**Returns**

BLT\_NVM\_OKAY if successful, BLT\_NVM\_NOT\_IN\_RANGE if the address is not within the supported memory range, or BLT\_NVM\_ERROR if the write operation failed.

**7.625.2.11 XcpGetSeedHook()**

```
blt_int8u XcpGetSeedHook (
 blt_int8u resource,
 blt_int8u * seed)
```

Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the GET\_SEED command.

**Parameters**

|                 |                                                        |
|-----------------|--------------------------------------------------------|
| <i>resource</i> | Resource that the seed is requested for (XCP_RES_XXX). |
| <i>seed</i>     | Pointer to byte buffer where the seed will be stored.  |

**Returns**

Length of the seed in bytes.

**7.625.2.12 XcpVerifyKeyHook()**

```
blt_int8u XcpVerifyKeyHook (
 blt_int8u resource,
 blt_int8u * key,
 blt_int8u len)
```

Called by the UNLOCK command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

**Parameters**

|                 |                                             |
|-----------------|---------------------------------------------|
| <i>resource</i> | resource to unlock (XCP_RES_XXX).           |
| <i>key</i>      | pointer to the byte buffer holding the key. |
| <i>len</i>      | length of the key in bytes.                 |

## Returns

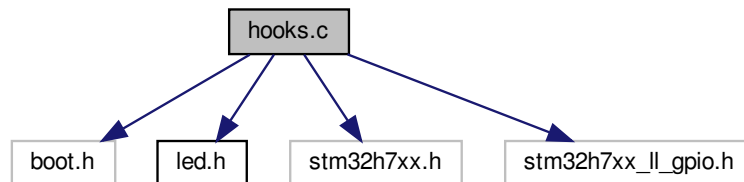
1 if the key was correct, 0 otherwise.

## 7.626 hooks.c File Reference

Bootloader callback source file.

```
#include "boot.h"
#include "led.h"
#include "stm32h7xx.h"
#include "stm32h7xx_ll_gpio.h"
```

Include dependency graph for ARMCM7\_STM32H7\_Nucleo\_H743ZI\_CubeIDE/Boot/App/hooks.c:



## Functions

- [blt\\_bool CpuUserProgramStartHook](#) (void)  
*Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.*
- void [CopInitHook](#) (void)  
*Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.*
- void [CopServiceHook](#) (void)  
*Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.*
- void [UsbConnectHook](#) ([blt\\_bool](#) connect)  
*Callback that gets called whenever the USB device should be connected to the USB bus.*
- void [UsbEnterLowPowerModeHook](#) (void)  
*Callback that gets called whenever the USB host requests the device to enter a low power mode.*
- void [UsbLeaveLowPowerModeHook](#) (void)  
*Callback that gets called whenever the USB host requests the device to exit low power mode.*
- void [BackDoorInitHook](#) (void)  
*Initializes the backdoor entry option.*
- [blt\\_bool BackDoorEntryHook](#) (void)  
*Checks if a backdoor entry is requested.*
- void [NvmInitHook](#) (void)  
*Callback that gets called at the start of the internal NVM driver initialization routine.*
- void [NvmReinitHook](#) (void)

*Callback that gets called at the start of a firmware update to reinitialize the NVM driver.*

- [blt\\_int8u NvmWriteHook](#) ([blt\\_addr](#) addr, [blt\\_int32u](#) len, [blt\\_int8u](#) \*data)

*Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the data hasn't been written yet.*

- [blt\\_int8u NvmEraseHook](#) ([blt\\_addr](#) addr, [blt\\_int32u](#) len)

*Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the memory hasn't been erased yet.*

- [blt\\_bool NvmDoneHook](#) (void)

*Callback that gets called at the end of the NVM programming session.*

- [blt\\_int8u XcpGetSeedHook](#) ([blt\\_int8u](#) resource, [blt\\_int8u](#) \*seed)

*Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the GET\_SEED command.*

- [blt\\_int8u XcpVerifyKeyHook](#) ([blt\\_int8u](#) resource, [blt\\_int8u](#) \*key, [blt\\_int8u](#) len)

*Called by the UNLOCK command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.*

## 7.626.1 Detailed Description

Bootloader callback source file.

## 7.626.2 Function Documentation

### 7.626.2.1 BackDoorEntryHook()

```
blt_bool BackDoorEntryHook (
 void)
```

Checks if a backdoor entry is requested.

#### Returns

BLT\_TRUE if the backdoor entry is requested, BLT\_FALSE otherwise.

### 7.626.2.2 BackDoorInitHook()

```
void BackDoorInitHook (
 void)
```

Initializes the backdoor entry option.

#### Returns

none.

### 7.626.2.3 CopInitHook()

```
void CopInitHook (
 void)
```

Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.

#### Returns

none.

### 7.626.2.4 CopServiceHook()

```
void CopServiceHook (
 void)
```

Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.

#### Returns

none.

### 7.626.2.5 CpuUserProgramStartHook()

```
blt_bool CpuUserProgramStartHook (
 void)
```

Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.

#### Returns

BLT\_TRUE if it is okay to start the user program, BLT\_FALSE to keep keep the bootloader active.

### 7.626.2.6 NvmDoneHook()

```
blt_bool NvmDoneHook (
 void)
```

Callback that gets called at the end of the NVM programming session.

#### Returns

BLT\_TRUE is successful, BLT\_FALSE otherwise.



### 7.626.2.7 NvmEraseHook()

```
blt_int8u NvmEraseHook (
 blt_addr addr,
 blt_int32u len)
```

Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the memory hasn't been erased yet.

#### Parameters

|             |                  |
|-------------|------------------|
| <i>addr</i> | Start address.   |
| <i>len</i>  | Length in bytes. |

#### Returns

BLT\_NVM\_OKAY if successful, BLT\_NVM\_NOT\_IN\_RANGE if the address is not within the supported memory range, or BLT\_NVM\_ERROR if the erase operation failed.

### 7.626.2.8 NvmInitHook()

```
void NvmInitHook (
 void)
```

Callback that gets called at the start of the internal NVM driver initialization routine.

#### Returns

none.

### 7.626.2.9 NvmReinitHook()

```
void NvmReinitHook (
 void)
```

Callback that gets called at the start of a firmware update to reinitialize the NVM driver.

#### Returns

none.

### 7.626.2.10 NvmWriteHook()

```
blt_int8u NvmWriteHook (
 blt_addr addr,
 blt_int32u len,
 blt_int8u * data)
```

Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the data hasn't been written yet.

**Parameters**

|             |                             |
|-------------|-----------------------------|
| <i>addr</i> | Start address.              |
| <i>len</i>  | Length in bytes.            |
| <i>data</i> | Pointer to the data buffer. |

**Returns**

BLT\_NVM\_OKAY if successful, BLT\_NVM\_NOT\_IN\_RANGE if the address is not within the supported memory range, or BLT\_NVM\_ERROR if the write operation failed.

**7.626.2.11 UsbConnectHook()**

```
void UsbConnectHook (
 blt_bool connect)
```

Callback that gets called whenever the USB device should be connected to the USB bus.

**Parameters**

|                |                                                  |
|----------------|--------------------------------------------------|
| <i>connect</i> | BLT_TRUE to connect and BLT_FALSE to disconnect. |
|----------------|--------------------------------------------------|

**Returns**

none.

**7.626.2.12 UsbEnterLowPowerModeHook()**

```
void UsbEnterLowPowerModeHook (
 void)
```

Callback that gets called whenever the USB host requests the device to enter a low power mode.

**Returns**

none.

**7.626.2.13 UsbLeaveLowPowerModeHook()**

```
void UsbLeaveLowPowerModeHook (
 void)
```

Callback that gets called whenever the USB host requests the device to exit low power mode.

**Returns**

none.

#### 7.626.2.14 XcpGetSeedHook()

```
blt_int8u XcpGetSeedHook (
 blt_int8u resource,
 blt_int8u * seed)
```

Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the GET\_SEED command.

##### Parameters

|                 |                                                        |
|-----------------|--------------------------------------------------------|
| <i>resource</i> | Resource that the seed is requested for (XCP_RES_XXX). |
| <i>seed</i>     | Pointer to byte buffer where the seed will be stored.  |

##### Returns

Length of the seed in bytes.

#### 7.626.2.15 XcpVerifyKeyHook()

```
blt_int8u XcpVerifyKeyHook (
 blt_int8u resource,
 blt_int8u * key,
 blt_int8u len)
```

Called by the UNLOCK command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

##### Parameters

|                 |                                             |
|-----------------|---------------------------------------------|
| <i>resource</i> | resource to unlock (XCP_RES_XXX).           |
| <i>key</i>      | pointer to the byte buffer holding the key. |
| <i>len</i>      | length of the key in bytes.                 |

##### Returns

1 if the key was correct, 0 otherwise.

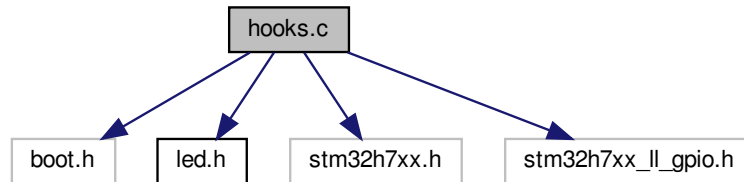
## 7.627 hooks.c File Reference

Bootloader callback source file.

```
#include "boot.h"
#include "led.h"
#include "stm32h7xx.h"
```

```
#include "stm32h7xx_ll_gpio.h"
```

Include dependency graph for ARMC7\_STM32H7\_Nucleo\_H743ZI\_GCC/Boot/hooks.c:



## Functions

- [blt\\_bool CpuUserProgramStartHook](#) (void)
 

*Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.*
- void [CopInitHook](#) (void)
 

*Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.*
- void [CopServiceHook](#) (void)
 

*Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.*
- void [UsbConnectHook](#) (blt\_bool connect)
 

*Callback that gets called whenever the USB device should be connected to the USB bus.*
- void [UsbEnterLowPowerModeHook](#) (void)
 

*Callback that gets called whenever the USB host requests the device to enter a low power mode.*
- void [UsbLeaveLowPowerModeHook](#) (void)
 

*Callback that gets called whenever the USB host requests the device to exit low power mode.*
- void [BackDoorInitHook](#) (void)
 

*Initializes the backdoor entry option.*
- [blt\\_bool BackDoorEntryHook](#) (void)
 

*Checks if a backdoor entry is requested.*
- void [NvmInitHook](#) (void)
 

*Callback that gets called at the start of the internal NVM driver initialization routine.*
- void [NvmReinitHook](#) (void)
 

*Callback that gets called at the start of a firmware update to reinitialize the NVM driver.*
- [blt\\_int8u NvmWriteHook](#) (blt\_addr addr, blt\_int32u len, blt\_int8u \*data)
 

*Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the data hasn't been written yet.*
- [blt\\_int8u NvmEraseHook](#) (blt\_addr addr, blt\_int32u len)
 

*Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the memory hasn't been erased yet.*
- [blt\\_bool NvmDoneHook](#) (void)
 

*Callback that gets called at the end of the NVM programming session.*
- [blt\\_int8u XcpGetSeedHook](#) (blt\_int8u resource, blt\_int8u \*seed)

*Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the GET\_SEED command.*

- `blt_int8u XcpVerifyKeyHook (blt_int8u resource, blt_int8u *key, blt_int8u len)`

*Called by the UNLOCK command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.*

## 7.627.1 Detailed Description

Bootloader callback source file.

## 7.627.2 Function Documentation

### 7.627.2.1 BackDoorEntryHook()

```
blt_bool BackDoorEntryHook (
 void)
```

Checks if a backdoor entry is requested.

#### Returns

BLT\_TRUE if the backdoor entry is requested, BLT\_FALSE otherwise.

### 7.627.2.2 BackDoorInitHook()

```
void BackDoorInitHook (
 void)
```

Initializes the backdoor entry option.

#### Returns

none.

### 7.627.2.3 CopInitHook()

```
void CopInitHook (
 void)
```

Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.

#### Returns

none.

#### 7.627.2.4 CopServiceHook()

```
void CopServiceHook (
 void)
```

Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.

##### Returns

none.

#### 7.627.2.5 CpuUserProgramStartHook()

```
blt_bool CpuUserProgramStartHook (
 void)
```

Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.

##### Returns

BLT\_TRUE if it is okay to start the user program, BLT\_FALSE to keep keep the bootloader active.

#### 7.627.2.6 NvmDoneHook()

```
blt_bool NvmDoneHook (
 void)
```

Callback that gets called at the end of the NVM programming session.

##### Returns

BLT\_TRUE is successful, BLT\_FALSE otherwise.

#### 7.627.2.7 NvmEraseHook()

```
blt_int8u NvmEraseHook (
 blt_addr addr,
 blt_int32u len)
```

Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the memory hasn't been erased yet.

**Parameters**

|             |                  |
|-------------|------------------|
| <i>addr</i> | Start address.   |
| <i>len</i>  | Length in bytes. |

**Returns**

BLT\_NVM\_OKAY if successful, BLT\_NVM\_NOT\_IN\_RANGE if the address is not within the supported memory range, or BLT\_NVM\_ERROR if the erase operation failed.

**7.627.2.8 NvmInitHook()**

```
void NvmInitHook (
 void)
```

Callback that gets called at the start of the internal NVM driver initialization routine.

**Returns**

none.

**7.627.2.9 NvmReinitHook()**

```
void NvmReinitHook (
 void)
```

Callback that gets called at the start of a firmware update to reinitialize the NVM driver.

**Returns**

none.

**7.627.2.10 NvmWriteHook()**

```
blt_int8u NvmWriteHook (
 blt_addr addr,
 blt_int32u len,
 blt_int8u * data)
```

Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the data hasn't been written yet.

**Parameters**

|             |                             |
|-------------|-----------------------------|
| <i>addr</i> | Start address.              |
| <i>len</i>  | Length in bytes.            |
| <i>data</i> | Pointer to the data buffer. |

**Returns**

BLT\_NVM\_OKAY if successful, BLT\_NVM\_NOT\_IN\_RANGE if the address is not within the supported memory range, or BLT\_NVM\_ERROR if the write operation failed.

**7.627.2.11 UsbConnectHook()**

```
void UsbConnectHook (
 blt_bool connect)
```

Callback that gets called whenever the USB device should be connected to the USB bus.

**Parameters**

|                |                                                  |
|----------------|--------------------------------------------------|
| <i>connect</i> | BLT_TRUE to connect and BLT_FALSE to disconnect. |
|----------------|--------------------------------------------------|

**Returns**

none.

**7.627.2.12 UsbEnterLowPowerModeHook()**

```
void UsbEnterLowPowerModeHook (
 void)
```

Callback that gets called whenever the USB host requests the device to enter a low power mode.

**Returns**

none.

**7.627.2.13 UsbLeaveLowPowerModeHook()**

```
void UsbLeaveLowPowerModeHook (
 void)
```

Callback that gets called whenever the USB host requests the device to exit low power mode.

**Returns**

none.



#### 7.627.2.14 XcpGetSeedHook()

```
blt_int8u XcpGetSeedHook (
 blt_int8u resource,
 blt_int8u * seed)
```

Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the GET\_SEED command.

##### Parameters

|                 |                                                        |
|-----------------|--------------------------------------------------------|
| <i>resource</i> | Resource that the seed is requested for (XCP_RES_XXX). |
| <i>seed</i>     | Pointer to byte buffer where the seed will be stored.  |

##### Returns

Length of the seed in bytes.

#### 7.627.2.15 XcpVerifyKeyHook()

```
blt_int8u XcpVerifyKeyHook (
 blt_int8u resource,
 blt_int8u * key,
 blt_int8u len)
```

Called by the UNLOCK command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

##### Parameters

|                 |                                             |
|-----------------|---------------------------------------------|
| <i>resource</i> | resource to unlock (XCP_RES_XXX).           |
| <i>key</i>      | pointer to the byte buffer holding the key. |
| <i>len</i>      | length of the key in bytes.                 |

##### Returns

1 if the key was correct, 0 otherwise.

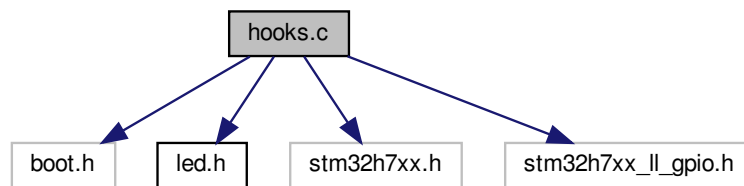
## 7.628 hooks.c File Reference

Bootloader callback source file.

```
#include "boot.h"
#include "led.h"
#include "stm32h7xx.h"
```

```
#include "stm32h7xx_ll_gpio.h"
```

Include dependency graph for ARMC7\_STM32H7\_Nucleo\_H743ZI\_IAR/Boot/hooks.c:



## Functions

- [blt\\_bool CpuUserProgramStartHook](#) (void)
 

*Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.*
- void [CopInitHook](#) (void)
 

*Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.*
- void [CopServiceHook](#) (void)
 

*Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.*
- void [UsbConnectHook](#) ([blt\\_bool](#) connect)
 

*Callback that gets called whenever the USB device should be connected to the USB bus.*
- void [UsbEnterLowPowerModeHook](#) (void)
 

*Callback that gets called whenever the USB host requests the device to enter a low power mode.*
- void [UsbLeaveLowPowerModeHook](#) (void)
 

*Callback that gets called whenever the USB host requests the device to exit low power mode.*
- void [BackDoorInitHook](#) (void)
 

*Initializes the backdoor entry option.*
- [blt\\_bool BackDoorEntryHook](#) (void)
 

*Checks if a backdoor entry is requested.*
- void [NvmInitHook](#) (void)
 

*Callback that gets called at the start of the internal NVM driver initialization routine.*
- void [NvmReinitHook](#) (void)
 

*Callback that gets called at the start of a firmware update to reinitialize the NVM driver.*
- [blt\\_int8u NvmWriteHook](#) ([blt\\_addr](#) addr, [blt\\_int32u](#) len, [blt\\_int8u](#) \*data)
 

*Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the data hasn't been written yet.*
- [blt\\_int8u NvmEraseHook](#) ([blt\\_addr](#) addr, [blt\\_int32u](#) len)
 

*Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the memory hasn't been erased yet.*
- [blt\\_bool NvmDoneHook](#) (void)
 

*Callback that gets called at the end of the NVM programming session.*
- [blt\\_int8u XcpGetSeedHook](#) ([blt\\_int8u](#) resource, [blt\\_int8u](#) \*seed)

*Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the GET\_SEED command.*

- `blt_int8u XcpVerifyKeyHook (blt_int8u resource, blt_int8u *key, blt_int8u len)`

*Called by the UNLOCK command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.*

## 7.628.1 Detailed Description

Bootloader callback source file.

## 7.628.2 Function Documentation

### 7.628.2.1 BackDoorEntryHook()

```
blt_bool BackDoorEntryHook (
 void)
```

Checks if a backdoor entry is requested.

#### Returns

BLT\_TRUE if the backdoor entry is requested, BLT\_FALSE otherwise.

### 7.628.2.2 BackDoorInitHook()

```
void BackDoorInitHook (
 void)
```

Initializes the backdoor entry option.

#### Returns

none.

### 7.628.2.3 CopInitHook()

```
void CopInitHook (
 void)
```

Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.

#### Returns

none.

#### 7.628.2.4 CopServiceHook()

```
void CopServiceHook (
 void)
```

Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.

##### Returns

none.

#### 7.628.2.5 CpuUserProgramStartHook()

```
blt_bool CpuUserProgramStartHook (
 void)
```

Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.

##### Returns

BLT\_TRUE if it is okay to start the user program, BLT\_FALSE to keep keep the bootloader active.

#### 7.628.2.6 NvmDoneHook()

```
blt_bool NvmDoneHook (
 void)
```

Callback that gets called at the end of the NVM programming session.

##### Returns

BLT\_TRUE is successful, BLT\_FALSE otherwise.

#### 7.628.2.7 NvmEraseHook()

```
blt_int8u NvmEraseHook (
 blt_addr addr,
 blt_int32u len)
```

Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the memory hasn't been erased yet.

**Parameters**

|             |                  |
|-------------|------------------|
| <i>addr</i> | Start address.   |
| <i>len</i>  | Length in bytes. |

**Returns**

BLT\_NVM\_OKAY if successful, BLT\_NVM\_NOT\_IN\_RANGE if the address is not within the supported memory range, or BLT\_NVM\_ERROR if the erase operation failed.

**7.628.2.8 NvmInitHook()**

```
void NvmInitHook (
 void)
```

Callback that gets called at the start of the internal NVM driver initialization routine.

**Returns**

none.

**7.628.2.9 NvmReinitHook()**

```
void NvmReinitHook (
 void)
```

Callback that gets called at the start of a firmware update to reinitialize the NVM driver.

**Returns**

none.

**7.628.2.10 NvmWriteHook()**

```
blt_int8u NvmWriteHook (
 blt_addr addr,
 blt_int32u len,
 blt_int8u * data)
```

Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the data hasn't been written yet.

**Parameters**

|             |                             |
|-------------|-----------------------------|
| <i>addr</i> | Start address.              |
| <i>len</i>  | Length in bytes.            |
| <i>data</i> | Pointer to the data buffer. |

**Returns**

BLT\_NVM\_OKAY if successful, BLT\_NVM\_NOT\_IN\_RANGE if the address is not within the supported memory range, or BLT\_NVM\_ERROR if the write operation failed.

**7.628.2.11 UsbConnectHook()**

```
void UsbConnectHook (
 blt_bool connect)
```

Callback that gets called whenever the USB device should be connected to the USB bus.

**Parameters**

|                |                                                  |
|----------------|--------------------------------------------------|
| <i>connect</i> | BLT_TRUE to connect and BLT_FALSE to disconnect. |
|----------------|--------------------------------------------------|

**Returns**

none.

**7.628.2.12 UsbEnterLowPowerModeHook()**

```
void UsbEnterLowPowerModeHook (
 void)
```

Callback that gets called whenever the USB host requests the device to enter a low power mode.

**Returns**

none.

**7.628.2.13 UsbLeaveLowPowerModeHook()**

```
void UsbLeaveLowPowerModeHook (
 void)
```

Callback that gets called whenever the USB host requests the device to exit low power mode.

**Returns**

none.

#### 7.628.2.14 XcpGetSeedHook()

```
blt_int8u XcpGetSeedHook (
 blt_int8u resource,
 blt_int8u * seed)
```

Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the GET\_SEED command.

##### Parameters

|                 |                                                        |
|-----------------|--------------------------------------------------------|
| <i>resource</i> | Resource that the seed is requested for (XCP_RES_XXX). |
| <i>seed</i>     | Pointer to byte buffer where the seed will be stored.  |

##### Returns

Length of the seed in bytes.

#### 7.628.2.15 XcpVerifyKeyHook()

```
blt_int8u XcpVerifyKeyHook (
 blt_int8u resource,
 blt_int8u * key,
 blt_int8u len)
```

Called by the UNLOCK command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

##### Parameters

|                 |                                             |
|-----------------|---------------------------------------------|
| <i>resource</i> | resource to unlock (XCP_RES_XXX).           |
| <i>key</i>      | pointer to the byte buffer holding the key. |
| <i>len</i>      | length of the key in bytes.                 |

##### Returns

1 if the key was correct, 0 otherwise.

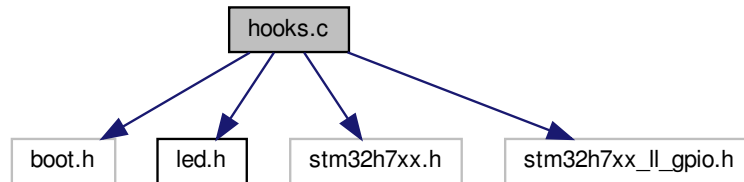
## 7.629 hooks.c File Reference

Bootloader callback source file.

```
#include "boot.h"
#include "led.h"
#include "stm32h7xx.h"
```

```
#include "stm32h7xx_ll_gpio.h"
```

Include dependency graph for ARMC7\_STM32H7\_Nucleo\_H743ZI\_Keil/Boot/hooks.c:



## Functions

- [blt\\_bool CpuUserProgramStartHook](#) (void)  
*Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.*
- void [CopInitHook](#) (void)  
*Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.*
- void [CopServiceHook](#) (void)  
*Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.*
- void [UsbConnectHook](#) (blt\_bool connect)  
*Callback that gets called whenever the USB device should be connected to the USB bus.*
- void [UsbEnterLowPowerModeHook](#) (void)  
*Callback that gets called whenever the USB host requests the device to enter a low power mode.*
- void [UsbLeaveLowPowerModeHook](#) (void)  
*Callback that gets called whenever the USB host requests the device to exit low power mode.*
- void [BackDoorInitHook](#) (void)  
*Initializes the backdoor entry option.*
- [blt\\_bool BackDoorEntryHook](#) (void)  
*Checks if a backdoor entry is requested.*
- void [NvmInitHook](#) (void)  
*Callback that gets called at the start of the internal NVM driver initialization routine.*
- void [NvmReinitHook](#) (void)  
*Callback that gets called at the start of a firmware update to reinitialize the NVM driver.*
- [blt\\_int8u NvmWriteHook](#) (blt\_addr addr, blt\_int32u len, blt\_int8u \*data)  
*Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the data hasn't been written yet.*
- [blt\\_int8u NvmEraseHook](#) (blt\_addr addr, blt\_int32u len)  
*Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the memory hasn't been erased yet.*
- [blt\\_bool NvmDoneHook](#) (void)  
*Callback that gets called at the end of the NVM programming session.*
- [blt\\_int8u XcpGetSeedHook](#) (blt\_int8u resource, blt\_int8u \*seed)



*Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the GET\_SEED command.*

- `blt_int8u XcpVerifyKeyHook (blt_int8u resource, blt_int8u *key, blt_int8u len)`

*Called by the UNLOCK command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.*

## 7.629.1 Detailed Description

Bootloader callback source file.

## 7.629.2 Function Documentation

### 7.629.2.1 BackDoorEntryHook()

```
blt_bool BackDoorEntryHook (
 void)
```

Checks if a backdoor entry is requested.

#### Returns

BLT\_TRUE if the backdoor entry is requested, BLT\_FALSE otherwise.

### 7.629.2.2 BackDoorInitHook()

```
void BackDoorInitHook (
 void)
```

Initializes the backdoor entry option.

#### Returns

none.

### 7.629.2.3 CopInitHook()

```
void CopInitHook (
 void)
```

Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.

#### Returns

none.

#### 7.629.2.4 CopServiceHook()

```
void CopServiceHook (
 void)
```

Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.

##### Returns

none.

#### 7.629.2.5 CpuUserProgramStartHook()

```
blt_bool CpuUserProgramStartHook (
 void)
```

Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.

##### Returns

BLT\_TRUE if it is okay to start the user program, BLT\_FALSE to keep keep the bootloader active.

#### 7.629.2.6 NvmDoneHook()

```
blt_bool NvmDoneHook (
 void)
```

Callback that gets called at the end of the NVM programming session.

##### Returns

BLT\_TRUE is successful, BLT\_FALSE otherwise.

#### 7.629.2.7 NvmEraseHook()

```
blt_int8u NvmEraseHook (
 blt_addr addr,
 blt_int32u len)
```

Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the memory hasn't been erased yet.

## Parameters

|             |                  |
|-------------|------------------|
| <i>addr</i> | Start address.   |
| <i>len</i>  | Length in bytes. |

## Returns

BLT\_NVM\_OKAY if successful, BLT\_NVM\_NOT\_IN\_RANGE if the address is not within the supported memory range, or BLT\_NVM\_ERROR if the erase operation failed.

## 7.629.2.8 NvmInitHook()

```
void NvmInitHook (
 void)
```

Callback that gets called at the start of the internal NVM driver initialization routine.

## Returns

none.

## 7.629.2.9 NvmReinitHook()

```
void NvmReinitHook (
 void)
```

Callback that gets called at the start of a firmware update to reinitialize the NVM driver.

## Returns

none.

## 7.629.2.10 NvmWriteHook()

```
blt_int8u NvmWriteHook (
 blt_addr addr,
 blt_int32u len,
 blt_int8u * data)
```

Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the data hasn't been written yet.

**Parameters**

|             |                             |
|-------------|-----------------------------|
| <i>addr</i> | Start address.              |
| <i>len</i>  | Length in bytes.            |
| <i>data</i> | Pointer to the data buffer. |

**Returns**

BLT\_NVM\_OKAY if successful, BLT\_NVM\_NOT\_IN\_RANGE if the address is not within the supported memory range, or BLT\_NVM\_ERROR if the write operation failed.

**7.629.2.11 UsbConnectHook()**

```
void UsbConnectHook (
 blt_bool connect)
```

Callback that gets called whenever the USB device should be connected to the USB bus.

**Parameters**

|                |                                                  |
|----------------|--------------------------------------------------|
| <i>connect</i> | BLT_TRUE to connect and BLT_FALSE to disconnect. |
|----------------|--------------------------------------------------|

**Returns**

none.

**7.629.2.12 UsbEnterLowPowerModeHook()**

```
void UsbEnterLowPowerModeHook (
 void)
```

Callback that gets called whenever the USB host requests the device to enter a low power mode.

**Returns**

none.

**7.629.2.13 UsbLeaveLowPowerModeHook()**

```
void UsbLeaveLowPowerModeHook (
 void)
```

Callback that gets called whenever the USB host requests the device to exit low power mode.

**Returns**

none.

#### 7.629.2.14 XcpGetSeedHook()

```
blt_int8u XcpGetSeedHook (
 blt_int8u resource,
 blt_int8u * seed)
```

Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the GET\_SEED command.

##### Parameters

|                 |                                                        |
|-----------------|--------------------------------------------------------|
| <i>resource</i> | Resource that the seed if requested for (XCP_RES_XXX). |
| <i>seed</i>     | Pointer to byte buffer wher the seed will be stored.   |

##### Returns

Length of the seed in bytes.

#### 7.629.2.15 XcpVerifyKeyHook()

```
blt_int8u XcpVerifyKeyHook (
 blt_int8u resource,
 blt_int8u * key,
 blt_int8u len)
```

Called by the UNLOCK command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

##### Parameters

|                 |                                             |
|-----------------|---------------------------------------------|
| <i>resource</i> | resource to unlock (XCP_RES_XXX).           |
| <i>key</i>      | pointer to the byte buffer holding the key. |
| <i>len</i>      | length of the key in bytes.                 |

##### Returns

1 if the key was correct, 0 otherwise.

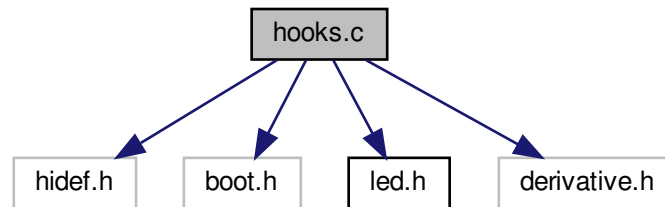
## 7.630 hooks.c File Reference

Bootloader callback source file.

```
#include <hidef.h>
#include "boot.h"
#include "led.h"
```

```
#include "derivative.h"
```

Include dependency graph for HCS12\_DevKit\_S12G128\_CodeWarrior/Boot/hooks.c:



## Functions

- [blt\\_bool CpuUserProgramStartHook](#) (void)  
*Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.*
- void [CopInitHook](#) (void)  
*Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.*
- void [CopServiceHook](#) (void)  
*Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.*
- void [BackDoorInitHook](#) (void)  
*Initializes the backdoor entry option.*
- [blt\\_bool BackDoorEntryHook](#) (void)  
*Checks if a backdoor entry is requested.*
- void [NvmInitHook](#) (void)  
*Callback that gets called at the start of the internal NVM driver initialization routine.*
- void [NvmReinitHook](#) (void)  
*Callback that gets called at the start of a firmware update to reinitialize the NVM driver.*
- [blt\\_int8u NvmWriteHook](#) ([blt\\_addr](#) addr, [blt\\_int32u](#) len, [blt\\_int8u](#) \*data)  
*Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the data hasn't been written yet.*
- [blt\\_int8u NvmEraseHook](#) ([blt\\_addr](#) addr, [blt\\_int32u](#) len)  
*Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the memory hasn't been erased yet.*
- [blt\\_bool NvmDoneHook](#) (void)  
*Callback that gets called at the end of the NVM programming session.*
- [blt\\_int8u XcpGetSeedHook](#) ([blt\\_int8u](#) resource, [blt\\_int8u](#) \*seed)  
*Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the GET\_SEED command.*
- [blt\\_int8u XcpVerifyKeyHook](#) ([blt\\_int8u](#) resource, [blt\\_int8u](#) \*key, [blt\\_int8u](#) len)  
*Called by the UNLOCK command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.*

## 7.630.1 Detailed Description

Bootloader callback source file.

## 7.630.2 Function Documentation

### 7.630.2.1 BackDoorEntryHook()

```
blt_bool BackDoorEntryHook (
 void)
```

Checks if a backdoor entry is requested.

#### Returns

BLT\_TRUE if the backdoor entry is requested, BLT\_FALSE otherwise.

### 7.630.2.2 BackDoorInitHook()

```
void BackDoorInitHook (
 void)
```

Initializes the backdoor entry option.

#### Returns

none.

### 7.630.2.3 CopInitHook()

```
void CopInitHook (
 void)
```

Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.

#### Returns

none.

#### 7.630.2.4 CopServiceHook()

```
void CopServiceHook (
 void)
```

Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.

##### Returns

none.

#### 7.630.2.5 CpuUserProgramStartHook()

```
blt_bool CpuUserProgramStartHook (
 void)
```

Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.

##### Returns

BLT\_TRUE if it is okay to start the user program, BLT\_FALSE to keep keep the bootloader active.

#### 7.630.2.6 NvmDoneHook()

```
blt_bool NvmDoneHook (
 void)
```

Callback that gets called at the end of the NVM programming session.

##### Returns

BLT\_TRUE is successful, BLT\_FALSE otherwise.

#### 7.630.2.7 NvmEraseHook()

```
blt_int8u NvmEraseHook (
 blt_addr addr,
 blt_int32u len)
```

Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the memory hasn't been erased yet.



## Parameters

|             |                  |
|-------------|------------------|
| <i>addr</i> | Start address.   |
| <i>len</i>  | Length in bytes. |

## Returns

BLT\_NVM\_OKAY if successful, BLT\_NVM\_NOT\_IN\_RANGE if the address is not within the supported memory range, or BLT\_NVM\_ERROR if the erase operation failed.

## 7.630.2.8 NvmInitHook()

```
void NvmInitHook (
 void)
```

Callback that gets called at the start of the internal NVM driver initialization routine.

## Returns

none.

## 7.630.2.9 NvmReinitHook()

```
void NvmReinitHook (
 void)
```

Callback that gets called at the start of a firmware update to reinitialize the NVM driver.

## Returns

none.

## 7.630.2.10 NvmWriteHook()

```
blt_int8u NvmWriteHook (
 blt_addr addr,
 blt_int32u len,
 blt_int8u * data)
```

Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the data hasn't been written yet.

**Parameters**

|             |                             |
|-------------|-----------------------------|
| <i>addr</i> | Start address.              |
| <i>len</i>  | Length in bytes.            |
| <i>data</i> | Pointer to the data buffer. |

**Returns**

BLT\_NVM\_OKAY if successful, BLT\_NVM\_NOT\_IN\_RANGE if the address is not within the supported memory range, or BLT\_NVM\_ERROR if the write operation failed.

**7.630.2.11 XcpGetSeedHook()**

```
blt_int8u XcpGetSeedHook (
 blt_int8u resource,
 blt_int8u * seed)
```

Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the GET\_SEED command.

**Parameters**

|                 |                                                        |
|-----------------|--------------------------------------------------------|
| <i>resource</i> | Resource that the seed is requested for (XCP_RES_XXX). |
| <i>seed</i>     | Pointer to byte buffer where the seed will be stored.  |

**Returns**

Length of the seed in bytes.

**7.630.2.12 XcpVerifyKeyHook()**

```
blt_int8u XcpVerifyKeyHook (
 blt_int8u resource,
 blt_int8u * key,
 blt_int8u len)
```

Called by the UNLOCK command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

**Parameters**

|                 |                                             |
|-----------------|---------------------------------------------|
| <i>resource</i> | resource to unlock (XCP_RES_XXX).           |
| <i>key</i>      | pointer to the byte buffer holding the key. |
| <i>len</i>      | length of the key in bytes.                 |

## Returns

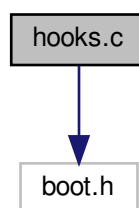
1 if the key was correct, 0 otherwise.

## 7.631 hooks.c File Reference

Bootloader callback source file.

```
#include "boot.h"
```

Include dependency graph for HCS12\_Evbplus\_Dragon12p\_CodeWarrior/Boot/hooks.c:



### Functions

- void [BackDoorInitHook](#) (void)  
*Initializes the backdoor entry option.*
- [blt\\_bool](#) [BackDoorEntryHook](#) (void)  
*Checks if a backdoor entry is requested.*
- [blt\\_bool](#) [CpuUserProgramStartHook](#) (void)  
*Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.*
- void [NvmInitHook](#) (void)  
*Callback that gets called at the start of the internal NVM driver initialization routine.*
- void [NvmReinitHook](#) (void)  
*Callback that gets called at the start of a firmware update to reinitialize the NVM driver.*
- [blt\\_int8u](#) [NvmWriteHook](#) ([blt\\_addr](#) addr, [blt\\_int32u](#) len, [blt\\_int8u](#) \*data)  
*Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then `BLT_NVM_NOT_IN_RANGE` must be returned to indicate that the data hasn't been written yet.*
- [blt\\_int8u](#) [NvmEraseHook](#) ([blt\\_addr](#) addr, [blt\\_int32u](#) len)  
*Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then `BLT_NVM_NOT_IN_RANGE` must be returned to indicate that the memory hasn't been erased yet.*
- [blt\\_bool](#) [NvmDoneHook](#) (void)  
*Callback that gets called at the end of the NVM programming session.*
- void [CopInitHook](#) (void)  
*Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.*

- void [CopServiceHook](#) (void)

*Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.*

- [blt\\_int8u XcpGetSeedHook](#) (blt\_int8u resource, blt\_int8u \*seed)

*Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the GET\_SEED command.*

- [blt\\_int8u XcpVerifyKeyHook](#) (blt\_int8u resource, blt\_int8u \*key, blt\_int8u len)

*Called by the UNLOCK command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.*

### 7.631.1 Detailed Description

Bootloader callback source file.

### 7.631.2 Function Documentation

#### 7.631.2.1 BackDoorEntryHook()

```
blt_bool BackDoorEntryHook (
 void)
```

Checks if a backdoor entry is requested.

#### Returns

BLT\_TRUE if the backdoor entry is requested, BLT\_FALSE otherwise.

Referenced by BackDoorInit().

#### 7.631.2.2 BackDoorInitHook()

```
void BackDoorInitHook (
 void)
```

Initializes the backdoor entry option.

#### Returns

none.

Referenced by BackDoorInit().

### 7.631.2.3 CopInitHook()

```
void CopInitHook (
 void)
```

Callback that gets called at the end of the internal COP driver initialization routine. It can be used to configure and enable the watchdog.

#### Returns

none.

Referenced by CopInit().

### 7.631.2.4 CopServiceHook()

```
void CopServiceHook (
 void)
```

Callback that gets called at the end of the internal COP driver service routine. This gets called upon initialization and during potential long lasting loops and routine. It can be used to service the watchdog to prevent a watchdog reset.

#### Returns

none.

Referenced by CopService().

### 7.631.2.5 CpuUserProgramStartHook()

```
blt_bool CpuUserProgramStartHook (
 void)
```

Callback that gets called when the bootloader is about to exit and hand over control to the user program. This is the last moment that some final checking can be performed and if necessary prevent the bootloader from activating the user program.

#### Returns

BLT\_TRUE if it is okay to start the user program, BLT\_FALSE to keep keep the bootloader active.

Referenced by CpuStartUserProgram().

#### 7.631.2.6 NvmDoneHook()

```
blt_bool NvmDoneHook (
 void)
```

Callback that gets called at the end of the NVM programming session.

##### Returns

BLT\_TRUE is successful, BLT\_FALSE otherwise.

Referenced by NvmDone().

#### 7.631.2.7 NvmEraseHook()

```
blt_int8u NvmEraseHook (
 blt_addr addr,
 blt_int32u len)
```

Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the memory hasn't been erased yet.

##### Parameters

|             |                  |
|-------------|------------------|
| <i>addr</i> | Start address.   |
| <i>len</i>  | Length in bytes. |

##### Returns

BLT\_NVM\_OKAY if successful, BLT\_NVM\_NOT\_IN\_RANGE if the address is not within the supported memory range, or BLT\_NVM\_ERROR is the erase operation failed.

Referenced by NvmErase().

#### 7.631.2.8 NvmInitHook()

```
void NvmInitHook (
 void)
```

Callback that gets called at the start of the internal NVM driver initialization routine.

##### Returns

none.

Referenced by NvmInit().

### 7.631.2.9 NvmReinitHook()

```
void NvmReinitHook (
 void)
```

Callback that gets called at the start of a firmware update to reinitialize the NVM driver.

#### Returns

none.

Referenced by NvmReinit().

### 7.631.2.10 NvmWriteHook()

```
blt_int8u NvmWriteHook (
 blt_addr addr,
 blt_int32u len,
 blt_int8u * data)
```

Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the data hasn't been written yet.

#### Parameters

|             |                             |
|-------------|-----------------------------|
| <i>addr</i> | Start address.              |
| <i>len</i>  | Length in bytes.            |
| <i>data</i> | Pointer to the data buffer. |

#### Returns

BLT\_NVM\_OKAY if successful, BLT\_NVM\_NOT\_IN\_RANGE if the address is not within the supported memory range, or BLT\_NVM\_ERROR if the write operation failed.

Referenced by NvmWrite().

### 7.631.2.11 XcpGetSeedHook()

```
blt_int8u XcpGetSeedHook (
 blt_int8u resource,
 blt_int8u * seed)
```

Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the GET\_SEED command.

**Parameters**

|                 |                                                        |
|-----------------|--------------------------------------------------------|
| <i>resource</i> | Resource that the seed is requested for (XCP_RES_XXX). |
| <i>seed</i>     | Pointer to byte buffer where the seed will be stored.  |

**Returns**

Length of the seed in bytes.

Referenced by XcpGetSeed().

**7.631.2.12 XcpVerifyKeyHook()**

```
blt_int8u XcpVerifyKeyHook (
 blt_int8u resource,
 blt_int8u * key,
 blt_int8u len)
```

Called by the UNLOCK command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

**Parameters**

|                 |                                             |
|-----------------|---------------------------------------------|
| <i>resource</i> | resource to unlock (XCP_RES_XXX).           |
| <i>key</i>      | pointer to the byte buffer holding the key. |
| <i>len</i>      | length of the key in bytes.                 |

**Returns**

1 if the key was correct, 0 otherwise.

Referenced by XcpVerifyKey().

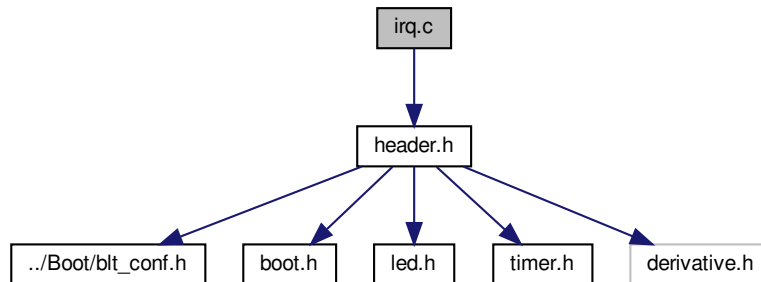
**7.632 irq.c File Reference**

IRQ driver source file.



```
#include "header.h"
```

Include dependency graph for irq.c:



## Functions

- void [IrqInterruptEnable](#) (void)  
*Enables the generation IRQ interrupts. Typically called once during software startup after completion of the initialization.*
- void [IrqInterruptDisable](#) (void)  
*Disables the generation IRQ interrupts and stores information on whether or not the interrupts were already disabled before explicitly disabling them with this function. Normally used as a pair together with [IrqInterruptRestore](#) during a critical section.*
- void [IrqInterruptRestore](#) (void)  
*Restore the generation IRQ interrupts to the setting it had prior to calling [IrqInterruptDisable](#). Normally used as a pair together with [IrqInterruptDisable](#) during a critical section.*

## Variables

- static volatile unsigned long [irqNesting](#) =0  
*IRQ nesting counter .*
- static volatile unsigned char [irqCCRregSave](#)  
*Copy of CCR register.*

### 7.632.1 Detailed Description

IRQ driver source file.

### 7.632.2 Function Documentation

#### 7.632.2.1 IrqInterruptDisable()

```
void IrqInterruptDisable (
 void)
```

Disables the generation IRQ interrupts and stores information on whether or not the interrupts were already disabled before explicitly disabling them with this function. Normally used as a pair together with IrqInterruptRestore during a critical section.

##### Returns

none.

#### 7.632.2.2 IrqInterruptEnable()

```
void IrqInterruptEnable (
 void)
```

Enables the generation IRQ interrupts. Typically called once during software startup after completion of the initialization.

##### Returns

none.

#### 7.632.2.3 IrqInterruptRestore()

```
void IrqInterruptRestore (
 void)
```

Restore the generation IRQ interrupts to the setting it had prior to calling IrqInterruptDisable. Normally used as a pair together with IrqInterruptDisable during a critical section.

##### Returns

none.

### 7.633 irq.h File Reference

IRQ driver header file.

## Functions

- void [IrqInterruptEnable](#) (void)  
*Enables the generation IRQ interrupts. Typically called once during software startup after completion of the initialization.*
- void [IrqInterruptDisable](#) (void)  
*Disables the generation IRQ interrupts and stores information on whether or not the interrupts were already disabled before explicitly disabling them with this function. Normally used as a pair together with [IrqInterruptRestore](#) during a critical section.*
- void [IrqInterruptRestore](#) (void)  
*Restore the generation IRQ interrupts to the setting it had prior to calling [IrqInterruptDisable](#). Normally used as a pair together with [IrqInterruptDisable](#) during a critical section.*

### 7.633.1 Detailed Description

IRQ driver header file.

### 7.633.2 Function Documentation

#### 7.633.2.1 IrqInterruptDisable()

```
void IrqInterruptDisable (
 void)
```

Disables the generation IRQ interrupts and stores information on whether or not the interrupts were already disabled before explicitly disabling them with this function. Normally used as a pair together with [IrqInterruptRestore](#) during a critical section.

#### Returns

none.

#### 7.633.2.2 IrqInterruptEnable()

```
void IrqInterruptEnable (
 void)
```

Enables the generation IRQ interrupts. Typically called once during software startup after completion of the initialization.

#### Returns

none.

### 7.633.2.3 IrqInterruptRestore()

```
void IrqInterruptRestore (
 void)
```

Restore the generation IRQ interrupts to the setting it had prior to calling IrqInterruptDisable. Normally used as a pair together with IrqInterruptDisable during a critical section.

#### Returns

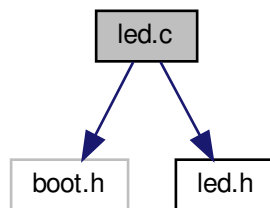
none.

## 7.634 led.c File Reference

LED driver source file.

```
#include "boot.h"
#include "led.h"
```

Include dependency graph for \_template/Boot/led.c:



## Functions

- void `LedBlinkInit` (`blt_int16u` interval\_ms)  
*Initializes the LED blink driver.*
- void `LedBlinkTask` (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void `LedBlinkExit` (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

## Variables

- static `blt_int16u` `ledBlinkIntervalMs`  
*Holds the desired LED blink interval time.*

### 7.634.1 Detailed Description

LED driver source file.

### 7.634.2 Function Documentation

#### 7.634.2.1 LedBlinkExit()

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

##### Returns

none.

Referenced by CpuUserProgramStartHook().

#### 7.634.2.2 LedBlinkInit()

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

##### Parameters

|                    |                                                                |
|--------------------|----------------------------------------------------------------|
| <i>interval_ms</i> | Specifies the desired LED blink interval time in milliseconds. |
|--------------------|----------------------------------------------------------------|

##### Returns

none.

Referenced by CopInitHook().

#### 7.634.2.3 LedBlinkTask()

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

**Returns**

none.

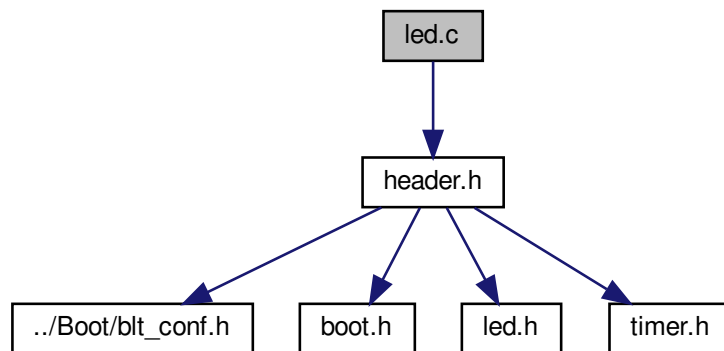
Referenced by CopServiceHook().

## 7.635 led.c File Reference

LED driver source file.

```
#include "header.h"
```

Include dependency graph for \_template/Prog/led.c:

**Macros**

- `#define LED_TOGGLE_MS (500)`  
*Toggle interval time in milliseconds.*

**Functions**

- void `LedInit` (void)  
*Initializes the LED.*
- void `LedToggle` (void)  
*Toggles the LED at a fixed time interval.*

### 7.635.1 Detailed Description

LED driver source file.

## 7.635.2 Function Documentation

### 7.635.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

#### Returns

none.

Referenced by Applnit(), and Init().

### 7.635.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

#### Returns

none.

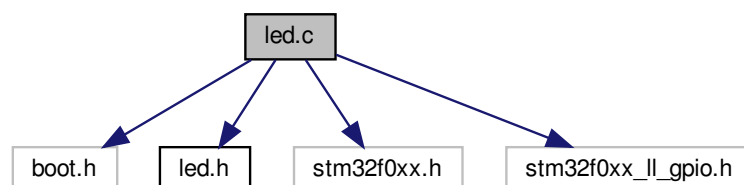
Referenced by AppTask(), and main().

## 7.636 led.c File Reference

LED driver source file.

```
#include "boot.h"
#include "led.h"
#include "stm32f0xx.h"
#include "stm32f0xx_ll_gpio.h"
```

Include dependency graph for ARMC00\_STM32F0\_Discovery\_STM32F051\_CubeIDE/Boot/App/led.c:



## Functions

- void `LedBlinkInit` (`blt_int16u` interval\_ms)  
*Initializes the LED blink driver.*
- void `LedBlinkTask` (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void `LedBlinkExit` (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

## Variables

- static `blt_int16u` `ledBlinkIntervalMs`  
*Holds the desired LED blink interval time.*

### 7.636.1 Detailed Description

LED driver source file.

### 7.636.2 Function Documentation

#### 7.636.2.1 `LedBlinkExit()`

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

#### Returns

none.

#### 7.636.2.2 `LedBlinkInit()`

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

#### Parameters

|                          |                                                                |
|--------------------------|----------------------------------------------------------------|
| <code>interval_ms</code> | Specifies the desired LED blink interval time in milliseconds. |
|--------------------------|----------------------------------------------------------------|



**Returns**

none.

**7.636.2.3 LedBlinkTask()**

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

**Returns**

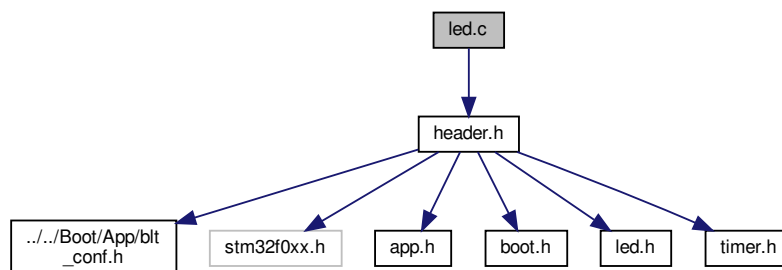
none.

**7.637 led.c File Reference**

LED driver source file.

```
#include "header.h"
```

Include dependency graph for ARMCM0\_STM32F0\_Discovery\_STM32F051\_CubeIDE/Prog/App/led.c:

**Macros**

- `#define LED_TOGGLE_MS (500)`  
*Toggle interval time in milliseconds.*

**Functions**

- `void LedInit (void)`  
*Initializes the LED.*
- `void LedToggle (void)`  
*Toggles the LED at a fixed time interval.*

### 7.637.1 Detailed Description

LED driver source file.

### 7.637.2 Function Documentation

#### 7.637.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

##### Returns

none.

#### 7.637.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

##### Returns

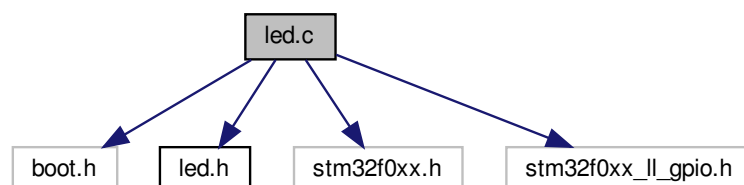
none.

## 7.638 led.c File Reference

LED driver source file.

```
#include "boot.h"
#include "led.h"
#include "stm32f0xx.h"
#include "stm32f0xx_ll_gpio.h"
```

Include dependency graph for ARMC00\_STM32F0\_Discovery\_STM32F051\_GCC/Boot/led.c:



## Functions

- void [LedBlinkInit](#) ([blt\\_int16u](#) interval\_ms)  
*Initializes the LED blink driver.*
- void [LedBlinkTask](#) (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void [LedBlinkExit](#) (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

## Variables

- static [blt\\_int16u](#) [ledBlinkIntervalMs](#)  
*Holds the desired LED blink interval time.*

### 7.638.1 Detailed Description

LED driver source file.

### 7.638.2 Function Documentation

#### 7.638.2.1 LedBlinkExit()

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

#### Returns

none.

#### 7.638.2.2 LedBlinkInit()

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

#### Parameters

|                    |                                                                |
|--------------------|----------------------------------------------------------------|
| <i>interval_ms</i> | Specifies the desired LED blink interval time in milliseconds. |
|--------------------|----------------------------------------------------------------|

**Returns**

none.

**7.638.2.3 LedBlinkTask()**

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

**Returns**

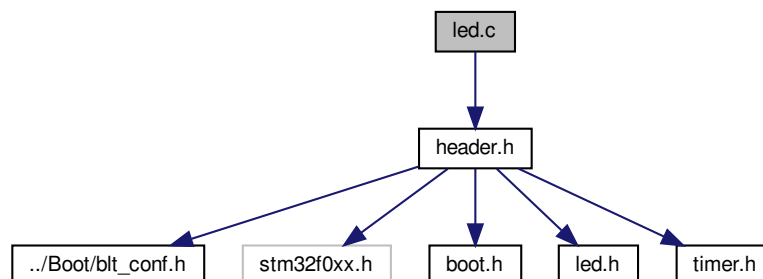
none.

**7.639 led.c File Reference**

LED driver source file.

```
#include "header.h"
```

Include dependency graph for ARMCM0\_STM32F0\_Discovery\_STM32F051\_GCC/Prog/led.c:

**Macros**

- `#define LED_TOGGLE_MS (500)`  
*Toggle interval time in milliseconds.*

**Functions**

- void `LedInit` (void)  
*Initializes the LED.*
- void `LedToggle` (void)  
*Toggles the LED at a fixed time interval.*

### 7.639.1 Detailed Description

LED driver source file.

### 7.639.2 Function Documentation

#### 7.639.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

##### Returns

none.

#### 7.639.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

##### Returns

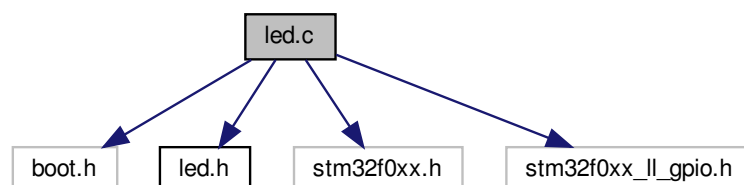
none.

## 7.640 led.c File Reference

LED driver source file.

```
#include "boot.h"
#include "led.h"
#include "stm32f0xx.h"
#include "stm32f0xx_ll_gpio.h"
```

Include dependency graph for ARMC00\_STM32F0\_Discovery\_STM32F051\_IAR/Boot/led.c:



## Functions

- void `LedBlinkInit` (`blt_int16u` interval\_ms)  
*Initializes the LED blink driver.*
- void `LedBlinkTask` (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void `LedBlinkExit` (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

## Variables

- static `blt_int16u` `ledBlinkIntervalMs`  
*Holds the desired LED blink interval time.*

### 7.640.1 Detailed Description

LED driver source file.

### 7.640.2 Function Documentation

#### 7.640.2.1 `LedBlinkExit()`

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

#### Returns

none.

#### 7.640.2.2 `LedBlinkInit()`

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

#### Parameters

|                          |                                                                |
|--------------------------|----------------------------------------------------------------|
| <code>interval_ms</code> | Specifies the desired LED blink interval time in milliseconds. |
|--------------------------|----------------------------------------------------------------|

**Returns**

none.

**7.640.2.3 LedBlinkTask()**

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

**Returns**

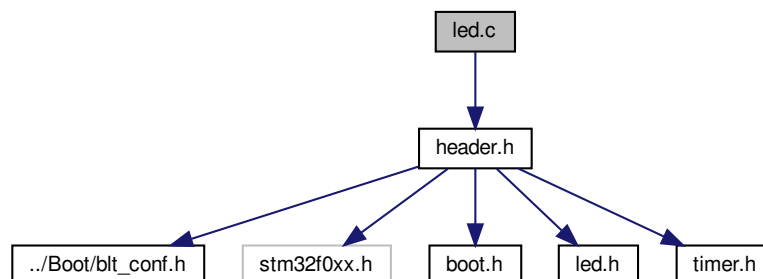
none.

**7.641 led.c File Reference**

LED driver source file.

```
#include "header.h"
```

Include dependency graph for ARMCM0\_STM32F0\_Discovery\_STM32F051\_IAR/Prog/led.c:

**Macros**

- `#define LED_TOGGLE_MS (500)`  
*Toggle interval time in milliseconds.*

**Functions**

- `void LedInit (void)`  
*Initializes the LED.*
- `void LedToggle (void)`  
*Toggles the LED at a fixed time interval.*

### 7.641.1 Detailed Description

LED driver source file.

### 7.641.2 Function Documentation

#### 7.641.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

##### Returns

none.

#### 7.641.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

##### Returns

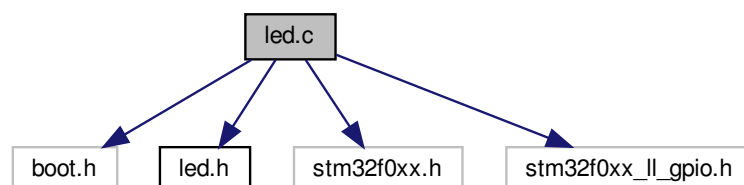
none.

## 7.642 led.c File Reference

LED driver source file.

```
#include "boot.h"
#include "led.h"
#include "stm32f0xx.h"
#include "stm32f0xx_ll_gpio.h"
```

Include dependency graph for ARMC00\_STM32F0\_Discovery\_STM32F051\_Keil/Boot/led.c:





## Functions

- void [LedBlinkInit](#) ([blt\\_int16u](#) interval\_ms)  
*Initializes the LED blink driver.*
- void [LedBlinkTask](#) (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void [LedBlinkExit](#) (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

## Variables

- static [blt\\_int16u](#) [ledBlinkIntervalMs](#)  
*Holds the desired LED blink interval time.*

### 7.642.1 Detailed Description

LED driver source file.

### 7.642.2 Function Documentation

#### 7.642.2.1 LedBlinkExit()

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

#### Returns

none.

#### 7.642.2.2 LedBlinkInit()

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

#### Parameters

|                    |                                                                |
|--------------------|----------------------------------------------------------------|
| <i>interval_ms</i> | Specifies the desired LED blink interval time in milliseconds. |
|--------------------|----------------------------------------------------------------|

**Returns**

none.

**7.642.2.3 LedBlinkTask()**

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

**Returns**

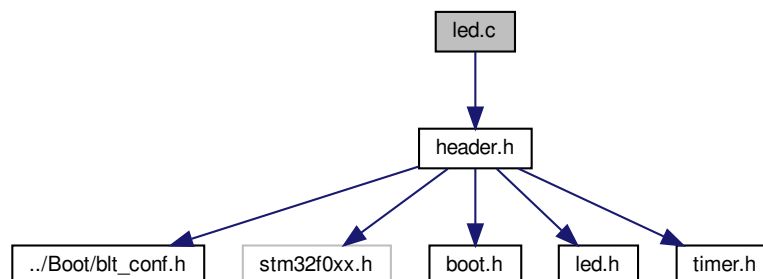
none.

**7.643 led.c File Reference**

LED driver source file.

```
#include "header.h"
```

Include dependency graph for ARMCM0\_STM32F0\_Discovery\_STM32F051\_Keil/Prog/led.c:

**Macros**

- `#define LED_TOGGLE_MS (500)`  
*Toggle interval time in milliseconds.*

**Functions**

- void `LedInit` (void)  
*Initializes the LED.*
- void `LedToggle` (void)  
*Toggles the LED at a fixed time interval.*

### 7.643.1 Detailed Description

LED driver source file.

### 7.643.2 Function Documentation

#### 7.643.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

##### Returns

none.

#### 7.643.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

##### Returns

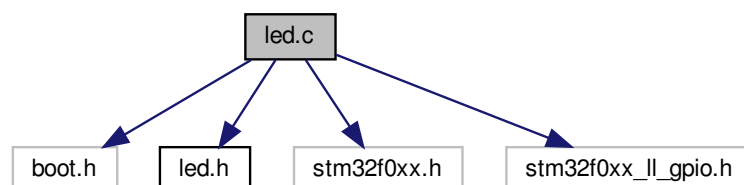
none.

## 7.644 led.c File Reference

LED driver source file.

```
#include "boot.h"
#include "led.h"
#include "stm32f0xx.h"
#include "stm32f0xx_ll_gpio.h"
```

Include dependency graph for ARMC00\_STM32F0\_Nucleo\_F091RC\_CubeIDE/Boot/App/led.c:



## Functions

- void [LedBlinkInit](#) ([blt\\_int16u](#) interval\_ms)  
*Initializes the LED blink driver.*
- void [LedBlinkTask](#) (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void [LedBlinkExit](#) (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

## Variables

- static [blt\\_int16u](#) [ledBlinkIntervalMs](#)  
*Holds the desired LED blink interval time.*

### 7.644.1 Detailed Description

LED driver source file.

### 7.644.2 Function Documentation

#### 7.644.2.1 LedBlinkExit()

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

#### Returns

none.

#### 7.644.2.2 LedBlinkInit()

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

#### Parameters

|                    |                                                                |
|--------------------|----------------------------------------------------------------|
| <i>interval_ms</i> | Specifies the desired LED blink interval time in milliseconds. |
|--------------------|----------------------------------------------------------------|

**Returns**

none.

**7.644.2.3 LedBlinkTask()**

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

**Returns**

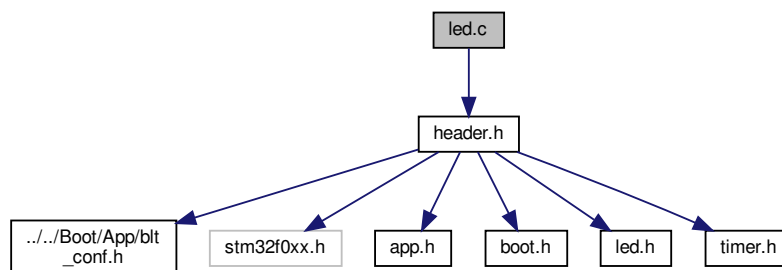
none.

**7.645 led.c File Reference**

LED driver source file.

```
#include "header.h"
```

Include dependency graph for ARMC00\_STM32F0\_Nucleo\_F091RC\_CubeIDE/Prog/App/led.c:

**Macros**

- `#define LED_TOGGLE_MS (500)`  
*Toggle interval time in milliseconds.*

**Functions**

- `void LedInit (void)`  
*Initializes the LED.*
- `void LedToggle (void)`  
*Toggles the LED at a fixed time interval.*

### 7.645.1 Detailed Description

LED driver source file.

### 7.645.2 Function Documentation

#### 7.645.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

##### Returns

none.

#### 7.645.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

##### Returns

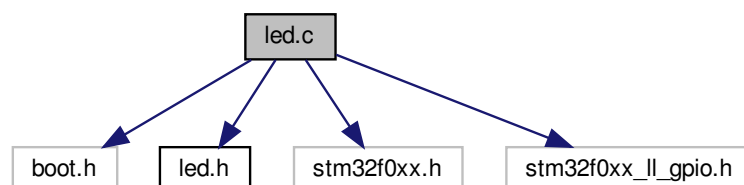
none.

## 7.646 led.c File Reference

LED driver source file.

```
#include "boot.h"
#include "led.h"
#include "stm32f0xx.h"
#include "stm32f0xx_ll_gpio.h"
```

Include dependency graph for ARMC00\_STM32F0\_Nucleo\_F091RC\_GCC/Boot/led.c:



## Functions

- void [LedBlinkInit](#) ([blt\\_int16u](#) interval\_ms)  
*Initializes the LED blink driver.*
- void [LedBlinkTask](#) (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void [LedBlinkExit](#) (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

## Variables

- static [blt\\_int16u](#) [ledBlinkIntervalMs](#)  
*Holds the desired LED blink interval time.*

### 7.646.1 Detailed Description

LED driver source file.

### 7.646.2 Function Documentation

#### 7.646.2.1 LedBlinkExit()

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

#### Returns

none.

#### 7.646.2.2 LedBlinkInit()

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

#### Parameters

|                    |                                                                |
|--------------------|----------------------------------------------------------------|
| <i>interval_ms</i> | Specifies the desired LED blink interval time in milliseconds. |
|--------------------|----------------------------------------------------------------|

**Returns**

none.

**7.646.2.3 LedBlinkTask()**

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

**Returns**

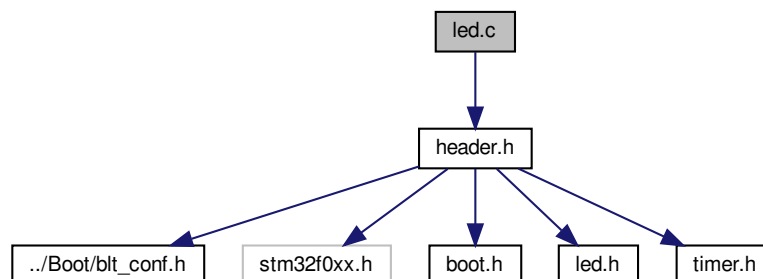
none.

**7.647 led.c File Reference**

LED driver source file.

```
#include "header.h"
```

Include dependency graph for ARMCM0\_STM32F0\_Nucleo\_F091RC\_GCC/Prog/led.c:

**Macros**

- `#define LED_TOGGLE_MS (500)`  
*Toggle interval time in milliseconds.*

**Functions**

- void `LedInit` (void)  
*Initializes the LED.*
- void `LedToggle` (void)  
*Toggles the LED at a fixed time interval.*



### 7.647.1 Detailed Description

LED driver source file.

### 7.647.2 Function Documentation

#### 7.647.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

**Returns**

none.

#### 7.647.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

**Returns**

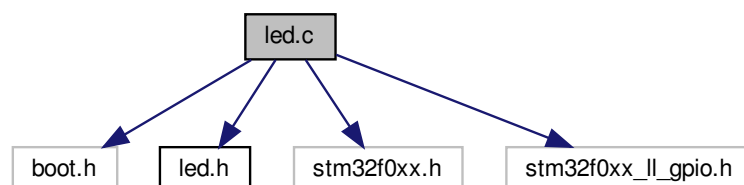
none.

## 7.648 led.c File Reference

LED driver source file.

```
#include "boot.h"
#include "led.h"
#include "stm32f0xx.h"
#include "stm32f0xx_ll_gpio.h"
```

Include dependency graph for ARMC00\_STM32F0\_Nucleo\_F091RC\_IAR/Boot/led.c:



## Functions

- void `LedBlinkInit` (`blt_int16u` interval\_ms)  
*Initializes the LED blink driver.*
- void `LedBlinkTask` (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void `LedBlinkExit` (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

## Variables

- static `blt_int16u` `ledBlinkIntervalMs`  
*Holds the desired LED blink interval time.*

### 7.648.1 Detailed Description

LED driver source file.

### 7.648.2 Function Documentation

#### 7.648.2.1 `LedBlinkExit()`

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

#### Returns

none.

#### 7.648.2.2 `LedBlinkInit()`

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

#### Parameters

|                          |                                                                |
|--------------------------|----------------------------------------------------------------|
| <code>interval_ms</code> | Specifies the desired LED blink interval time in milliseconds. |
|--------------------------|----------------------------------------------------------------|

**Returns**

none.

**7.648.2.3 LedBlinkTask()**

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

**Returns**

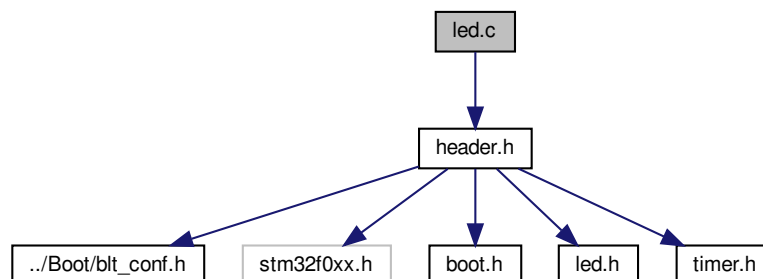
none.

**7.649 led.c File Reference**

LED driver source file.

```
#include "header.h"
```

Include dependency graph for ARMCM0\_STM32F0\_Nucleo\_F091RC\_IAR/Prog/led.c:

**Macros**

- `#define LED_TOGGLE_MS (500)`  
*Toggle interval time in milliseconds.*

**Functions**

- void `LedInit` (void)  
*Initializes the LED.*
- void `LedToggle` (void)  
*Toggles the LED at a fixed time interval.*

### 7.649.1 Detailed Description

LED driver source file.

### 7.649.2 Function Documentation

#### 7.649.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

##### Returns

none.

#### 7.649.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

##### Returns

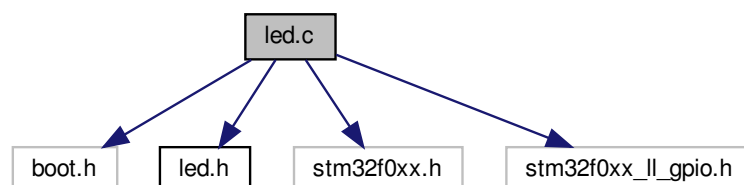
none.

## 7.650 led.c File Reference

LED driver source file.

```
#include "boot.h"
#include "led.h"
#include "stm32f0xx.h"
#include "stm32f0xx_ll_gpio.h"
```

Include dependency graph for ARMC00\_STM32F0\_Nucleo\_F091RC\_Keil/Boot/led.c:



## Functions

- void [LedBlinkInit](#) ([blt\\_int16u](#) interval\_ms)  
*Initializes the LED blink driver.*
- void [LedBlinkTask](#) (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void [LedBlinkExit](#) (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

## Variables

- static [blt\\_int16u](#) [ledBlinkIntervalMs](#)  
*Holds the desired LED blink interval time.*

### 7.650.1 Detailed Description

LED driver source file.

### 7.650.2 Function Documentation

#### 7.650.2.1 LedBlinkExit()

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

#### Returns

none.

#### 7.650.2.2 LedBlinkInit()

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

#### Parameters

|                    |                                                                |
|--------------------|----------------------------------------------------------------|
| <i>interval_ms</i> | Specifies the desired LED blink interval time in milliseconds. |
|--------------------|----------------------------------------------------------------|

**Returns**

none.

**7.650.2.3 LedBlinkTask()**

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

**Returns**

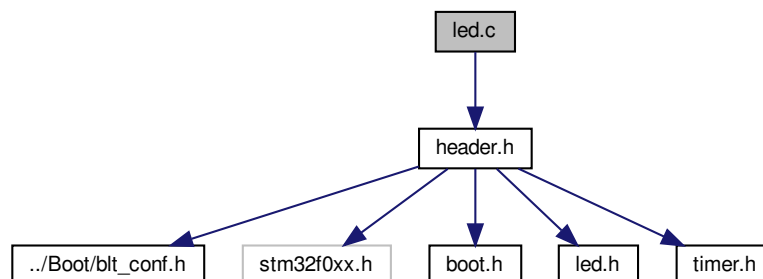
none.

**7.651 led.c File Reference**

LED driver source file.

```
#include "header.h"
```

Include dependency graph for ARMCM0\_STM32F0\_Nucleo\_F091RC\_Keil/Prog/led.c:

**Macros**

- `#define LED_TOGGLE_MS (500)`  
*Toggle interval time in milliseconds.*

**Functions**

- void `LedInit` (void)  
*Initializes the LED.*
- void `LedToggle` (void)  
*Toggles the LED at a fixed time interval.*

### 7.651.1 Detailed Description

LED driver source file.

### 7.651.2 Function Documentation

#### 7.651.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

##### Returns

none.

#### 7.651.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

##### Returns

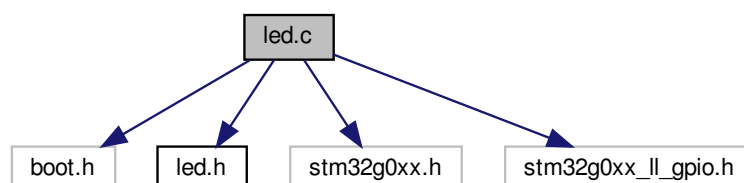
none.

## 7.652 led.c File Reference

LED driver source file.

```
#include "boot.h"
#include "led.h"
#include "stm32g0xx.h"
#include "stm32g0xx_ll_gpio.h"
```

Include dependency graph for ARMCM0\_STM32G0\_Nucleo\_G071RB\_CubeIDE/Boot/App/led.c:



## Functions

- void [LedBlinkInit](#) ([blt\\_int16u](#) interval\_ms)  
*Initializes the LED blink driver.*
- void [LedBlinkTask](#) (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void [LedBlinkExit](#) (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

## Variables

- static [blt\\_int16u](#) [ledBlinkIntervalMs](#)  
*Holds the desired LED blink interval time.*

### 7.652.1 Detailed Description

LED driver source file.

### 7.652.2 Function Documentation

#### 7.652.2.1 LedBlinkExit()

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

#### Returns

none.

#### 7.652.2.2 LedBlinkInit()

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

#### Parameters

|                    |                                                                |
|--------------------|----------------------------------------------------------------|
| <i>interval_ms</i> | Specifies the desired LED blink interval time in milliseconds. |
|--------------------|----------------------------------------------------------------|



**Returns**

none.

**7.652.2.3 LedBlinkTask()**

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

**Returns**

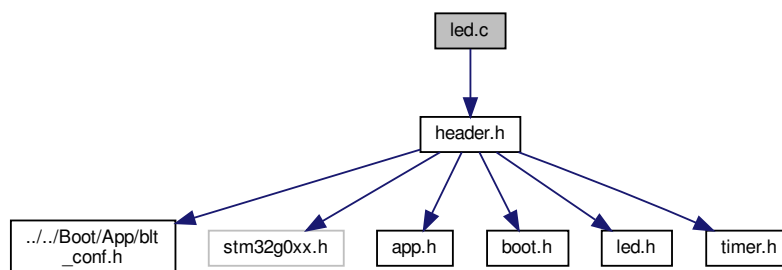
none.

**7.653 led.c File Reference**

LED driver source file.

```
#include "header.h"
```

Include dependency graph for ARMC00\_STM32G0\_Nucleo\_G071RB\_CubeIDE/Prog/App/led.c:

**Macros**

- `#define LED_TOGGLE_MS (500)`  
*Toggle interval time in milliseconds.*

**Functions**

- `void LedInit (void)`  
*Initializes the LED.*
- `void LedToggle (void)`  
*Toggles the LED at a fixed time interval.*

### 7.653.1 Detailed Description

LED driver source file.

### 7.653.2 Function Documentation

#### 7.653.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

##### Returns

none.

#### 7.653.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

##### Returns

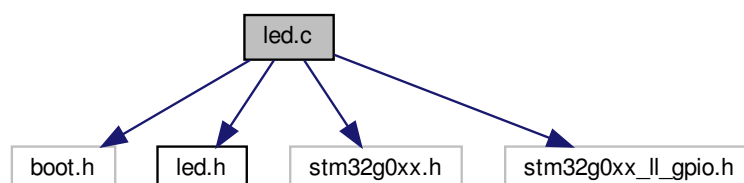
none.

## 7.654 led.c File Reference

LED driver source file.

```
#include "boot.h"
#include "led.h"
#include "stm32g0xx.h"
#include "stm32g0xx_ll_gpio.h"
```

Include dependency graph for ARMCM0\_STM32G0\_Nucleo\_G071RB\_GCC/Boot/led.c:



## Functions

- void [LedBlinkInit](#) ([blt\\_int16u](#) interval\_ms)  
*Initializes the LED blink driver.*
- void [LedBlinkTask](#) (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void [LedBlinkExit](#) (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

## Variables

- static [blt\\_int16u](#) [ledBlinkIntervalMs](#)  
*Holds the desired LED blink interval time.*

### 7.654.1 Detailed Description

LED driver source file.

### 7.654.2 Function Documentation

#### 7.654.2.1 LedBlinkExit()

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

#### Returns

none.

#### 7.654.2.2 LedBlinkInit()

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

#### Parameters

|                    |                                                                |
|--------------------|----------------------------------------------------------------|
| <i>interval_ms</i> | Specifies the desired LED blink interval time in milliseconds. |
|--------------------|----------------------------------------------------------------|

**Returns**

none.

**7.654.2.3 LedBlinkTask()**

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

**Returns**

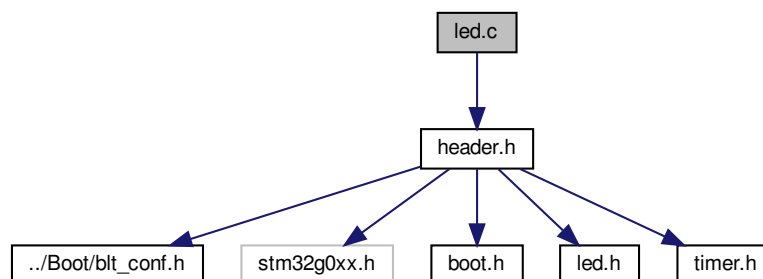
none.

**7.655 led.c File Reference**

LED driver source file.

```
#include "header.h"
```

Include dependency graph for ARMCM0\_STM32G0\_Nucleo\_G071RB\_GCC/Prog/led.c:

**Macros**

- `#define LED_TOGGLE_MS (500)`  
*Toggle interval time in milliseconds.*

**Functions**

- void `LedInit` (void)  
*Initializes the LED.*
- void `LedToggle` (void)  
*Toggles the LED at a fixed time interval.*

### 7.655.1 Detailed Description

LED driver source file.

### 7.655.2 Function Documentation

#### 7.655.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

##### Returns

none.

#### 7.655.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

##### Returns

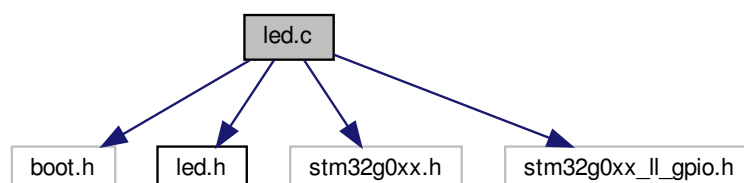
none.

## 7.656 led.c File Reference

LED driver source file.

```
#include "boot.h"
#include "led.h"
#include "stm32g0xx.h"
#include "stm32g0xx_ll_gpio.h"
```

Include dependency graph for ARMCM0\_STM32G0\_Nucleo\_G071RB\_IAR/Boot/led.c:



## Functions

- void `LedBlinkInit` (`blt_int16u` interval\_ms)  
*Initializes the LED blink driver.*
- void `LedBlinkTask` (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void `LedBlinkExit` (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

## Variables

- static `blt_int16u` `ledBlinkIntervalMs`  
*Holds the desired LED blink interval time.*

### 7.656.1 Detailed Description

LED driver source file.

### 7.656.2 Function Documentation

#### 7.656.2.1 LedBlinkExit()

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

#### Returns

none.

#### 7.656.2.2 LedBlinkInit()

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

#### Parameters

|                          |                                                                |
|--------------------------|----------------------------------------------------------------|
| <code>interval_ms</code> | Specifies the desired LED blink interval time in milliseconds. |
|--------------------------|----------------------------------------------------------------|

**Returns**

none.

**7.656.2.3 LedBlinkTask()**

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

**Returns**

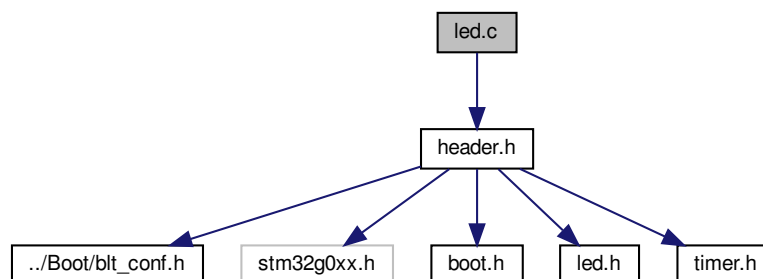
none.

**7.657 led.c File Reference**

LED driver source file.

```
#include "header.h"
```

Include dependency graph for ARMCM0\_STM32G0\_Nucleo\_G071RB\_IAR/Prog/led.c:

**Macros**

- `#define LED_TOGGLE_MS (500)`  
*Toggle interval time in milliseconds.*

**Functions**

- void `LedInit` (void)  
*Initializes the LED.*
- void `LedToggle` (void)  
*Toggles the LED at a fixed time interval.*

### 7.657.1 Detailed Description

LED driver source file.

### 7.657.2 Function Documentation

#### 7.657.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

##### Returns

none.

#### 7.657.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

##### Returns

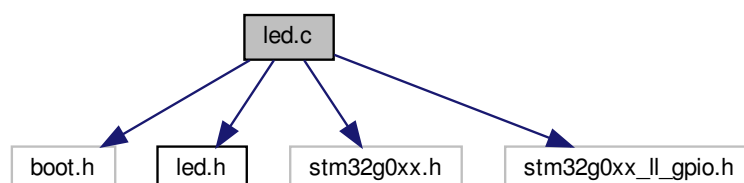
none.

## 7.658 led.c File Reference

LED driver source file.

```
#include "boot.h"
#include "led.h"
#include "stm32g0xx.h"
#include "stm32g0xx_ll_gpio.h"
```

Include dependency graph for ARMCM0\_STM32G0\_Nucleo\_G071RB\_Keil/Boot/led.c:





## Functions

- void [LedBlinkInit](#) ([blt\\_int16u](#) interval\_ms)  
*Initializes the LED blink driver.*
- void [LedBlinkTask](#) (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void [LedBlinkExit](#) (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

## Variables

- static [blt\\_int16u](#) [ledBlinkIntervalMs](#)  
*Holds the desired LED blink interval time.*

### 7.658.1 Detailed Description

LED driver source file.

### 7.658.2 Function Documentation

#### 7.658.2.1 LedBlinkExit()

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

#### Returns

none.

#### 7.658.2.2 LedBlinkInit()

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

#### Parameters

|                    |                                                                |
|--------------------|----------------------------------------------------------------|
| <i>interval_ms</i> | Specifies the desired LED blink interval time in milliseconds. |
|--------------------|----------------------------------------------------------------|

**Returns**

none.

**7.658.2.3 LedBlinkTask()**

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

**Returns**

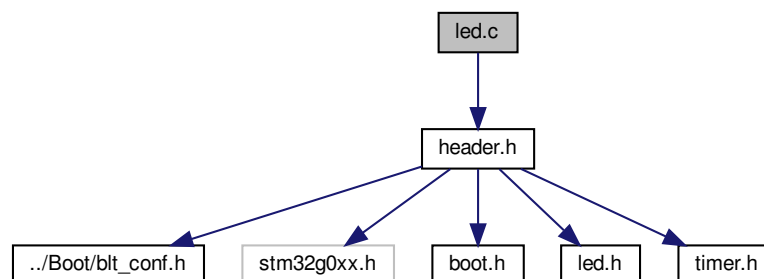
none.

**7.659 led.c File Reference**

LED driver source file.

```
#include "header.h"
```

Include dependency graph for ARMCM0\_STM32G0\_Nucleo\_G071RB\_Keil/Prog/led.c:

**Macros**

- `#define LED_TOGGLE_MS (500)`  
*Toggle interval time in milliseconds.*

**Functions**

- void `LedInit` (void)  
*Initializes the LED.*
- void `LedToggle` (void)  
*Toggles the LED at a fixed time interval.*

### 7.659.1 Detailed Description

LED driver source file.

### 7.659.2 Function Documentation

#### 7.659.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

##### Returns

none.

#### 7.659.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

##### Returns

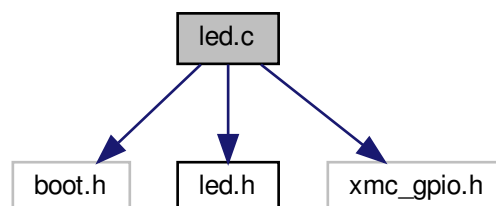
none.

## 7.660 led.c File Reference

LED driver source file.

```
#include "boot.h"
#include "led.h"
#include "xmc_gpio.h"
```

Include dependency graph for ARMCM0\_XMC1\_XMC1400\_Boot\_Kit\_GCC/Boot/led.c:



## Functions

- void `LedBlinkInit` (`blt_int16u` interval\_ms)  
*Initializes the LED blink driver.*
- void `LedBlinkTask` (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void `LedBlinkExit` (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

## Variables

- static `blt_int16u` `ledBlinkIntervalMs`  
*Holds the desired LED blink interval time.*

### 7.660.1 Detailed Description

LED driver source file.

### 7.660.2 Function Documentation

#### 7.660.2.1 LedBlinkExit()

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

#### Returns

none.

#### 7.660.2.2 LedBlinkInit()

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

#### Parameters

|                          |                                                                |
|--------------------------|----------------------------------------------------------------|
| <code>interval_ms</code> | Specifies the desired LED blink interval time in milliseconds. |
|--------------------------|----------------------------------------------------------------|

**Returns**

none.

**7.660.2.3 LedBlinkTask()**

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

**Returns**

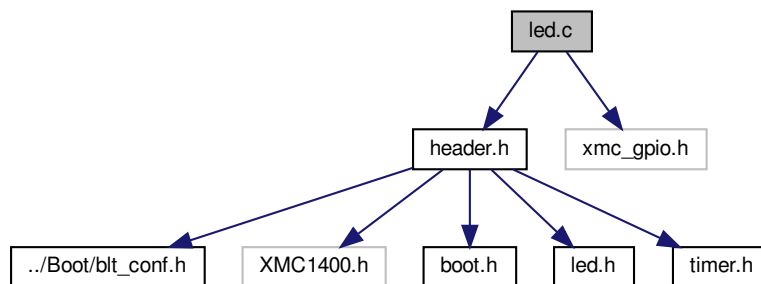
none.

**7.661 led.c File Reference**

LED driver source file.

```
#include "header.h"
#include "xmc_gpio.h"
```

Include dependency graph for ARMCM0\_XMC1\_XMC1400\_Boot\_Kit\_GCC/Prog/led.c:

**Macros**

- `#define LED_TOGGLE_MS (500)`  
*Toggle interval time in milliseconds.*

**Functions**

- `void LedInit (void)`  
*Initializes the LED.*
- `void LedToggle (void)`  
*Toggles the LED at a fixed time interval.*

### 7.661.1 Detailed Description

LED driver source file.

### 7.661.2 Function Documentation

#### 7.661.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

##### Returns

none.

#### 7.661.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

##### Returns

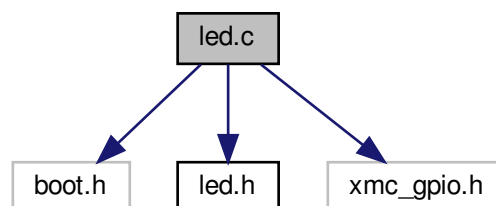
none.

## 7.662 led.c File Reference

LED driver source file.

```
#include "boot.h"
#include "led.h"
#include "xmc_gpio.h"
```

Include dependency graph for ARMCM0\_XMC1\_XMC1400\_Boot\_Kit\_IAR/Boot/led.c:



## Functions

- void [LedBlinkInit](#) ([blt\\_int16u](#) interval\_ms)  
*Initializes the LED blink driver.*
- void [LedBlinkTask](#) (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void [LedBlinkExit](#) (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

## Variables

- static [blt\\_int16u](#) [ledBlinkIntervalMs](#)  
*Holds the desired LED blink interval time.*

### 7.662.1 Detailed Description

LED driver source file.

### 7.662.2 Function Documentation

#### 7.662.2.1 LedBlinkExit()

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

#### Returns

none.

#### 7.662.2.2 LedBlinkInit()

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

#### Parameters

|                    |                                                                |
|--------------------|----------------------------------------------------------------|
| <i>interval_ms</i> | Specifies the desired LED blink interval time in milliseconds. |
|--------------------|----------------------------------------------------------------|

**Returns**

none.

**7.662.2.3 LedBlinkTask()**

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

**Returns**

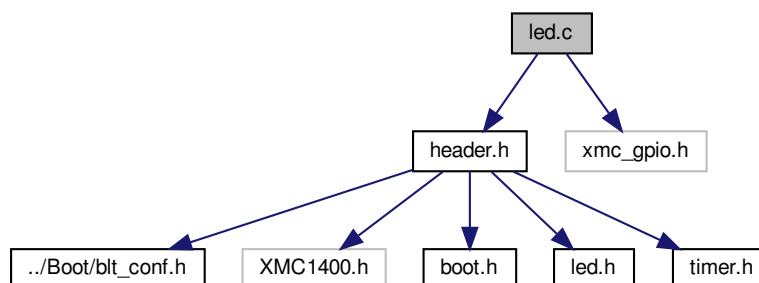
none.

**7.663 led.c File Reference**

LED driver source file.

```
#include "header.h"
#include "xmc_gpio.h"
```

Include dependency graph for ARMCM0\_XMC1\_XMC1400\_Boot\_Kit\_IAR/Prog/led.c:

**Macros**

- `#define LED_TOGGLE_MS (500)`  
*Toggle interval time in milliseconds.*

**Functions**

- void `LedInit` (void)  
*Initializes the LED.*
- void `LedToggle` (void)  
*Toggles the LED at a fixed time interval.*



### 7.663.1 Detailed Description

LED driver source file.

### 7.663.2 Function Documentation

#### 7.663.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

##### Returns

none.

#### 7.663.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

##### Returns

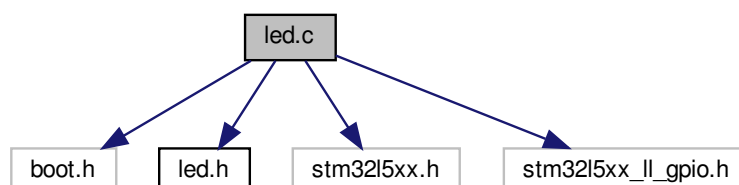
none.

## 7.664 led.c File Reference

LED driver source file.

```
#include "boot.h"
#include "led.h"
#include "stm32l5xx.h"
#include "stm32l5xx_ll_gpio.h"
```

Include dependency graph for ARMCM33\_STM32L5\_Nucleo\_L552ZE\_CubeIDE/Boot/App/led.c:



## Functions

- void `LedBlinkInit` (`blt_int16u` interval\_ms)  
*Initializes the LED blink driver.*
- void `LedBlinkTask` (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void `LedBlinkExit` (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

## Variables

- static `blt_int16u` `ledBlinkIntervalMs`  
*Holds the desired LED blink interval time.*

### 7.664.1 Detailed Description

LED driver source file.

### 7.664.2 Function Documentation

#### 7.664.2.1 `LedBlinkExit()`

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

#### Returns

none.

#### 7.664.2.2 `LedBlinkInit()`

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

#### Parameters

|                          |                                                                |
|--------------------------|----------------------------------------------------------------|
| <code>interval_ms</code> | Specifies the desired LED blink interval time in milliseconds. |
|--------------------------|----------------------------------------------------------------|

**Returns**

none.

**7.664.2.3 LedBlinkTask()**

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

**Returns**

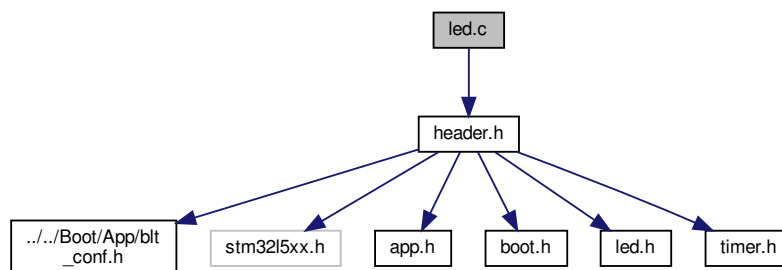
none.

**7.665 led.c File Reference**

LED driver source file.

```
#include "header.h"
```

Include dependency graph for ARMC33\_STM32L5\_Nucleo\_L552ZE\_CubeIDE/Prog/App/led.c:

**Macros**

- `#define LED_TOGGLE_MS (500)`  
*Toggle interval time in milliseconds.*

**Functions**

- `void LedInit (void)`  
*Initializes the LED.*
- `void LedToggle (void)`  
*Toggles the LED at a fixed time interval.*

### 7.665.1 Detailed Description

LED driver source file.

### 7.665.2 Function Documentation

#### 7.665.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

##### Returns

none.

#### 7.665.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

##### Returns

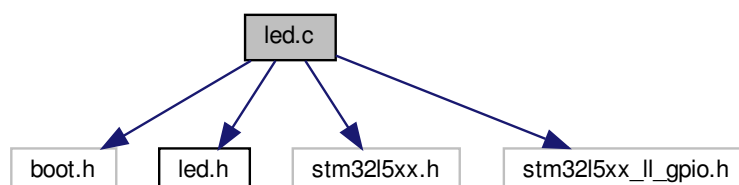
none.

## 7.666 led.c File Reference

LED driver source file.

```
#include "boot.h"
#include "led.h"
#include "stm32l5xx.h"
#include "stm32l5xx_ll_gpio.h"
```

Include dependency graph for ARMC33\_STM32L5\_Nucleo\_L552ZE\_GCC/Boot/led.c:



## Functions

- void [LedBlinkInit](#) ([blt\\_int16u](#) interval\_ms)  
*Initializes the LED blink driver.*
- void [LedBlinkTask](#) (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void [LedBlinkExit](#) (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

## Variables

- static [blt\\_int16u](#) [ledBlinkIntervalMs](#)  
*Holds the desired LED blink interval time.*

### 7.666.1 Detailed Description

LED driver source file.

### 7.666.2 Function Documentation

#### 7.666.2.1 LedBlinkExit()

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

#### Returns

none.

#### 7.666.2.2 LedBlinkInit()

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

#### Parameters

|                    |                                                                |
|--------------------|----------------------------------------------------------------|
| <i>interval_ms</i> | Specifies the desired LED blink interval time in milliseconds. |
|--------------------|----------------------------------------------------------------|

**Returns**

none.

**7.666.2.3 LedBlinkTask()**

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

**Returns**

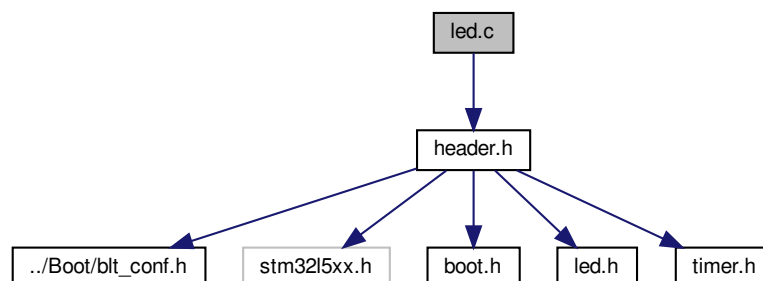
none.

**7.667 led.c File Reference**

LED driver source file.

```
#include "header.h"
```

Include dependency graph for ARMCM33\_STM32L5\_Nucleo\_L552ZE\_GCC/Prog/led.c:

**Macros**

- `#define LED_TOGGLE_MS (500)`  
*Toggle interval time in milliseconds.*

**Functions**

- void `LedInit` (void)  
*Initializes the LED.*
- void `LedToggle` (void)  
*Toggles the LED at a fixed time interval.*

### 7.667.1 Detailed Description

LED driver source file.

### 7.667.2 Function Documentation

#### 7.667.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

##### Returns

none.

#### 7.667.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

##### Returns

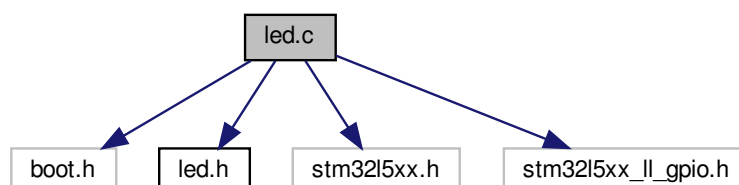
none.

## 7.668 led.c File Reference

LED driver source file.

```
#include "boot.h"
#include "led.h"
#include "stm32l5xx.h"
#include "stm32l5xx_ll_gpio.h"
```

Include dependency graph for ARMC33\_STM32L5\_Nucleo\_L552ZE\_IAR/Boot/led.c:



## Functions

- void [LedBlinkInit](#) ([blt\\_int16u](#) interval\_ms)  
*Initializes the LED blink driver.*
- void [LedBlinkTask](#) (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void [LedBlinkExit](#) (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

## Variables

- static [blt\\_int16u](#) [ledBlinkIntervalMs](#)  
*Holds the desired LED blink interval time.*

### 7.668.1 Detailed Description

LED driver source file.

### 7.668.2 Function Documentation

#### 7.668.2.1 [LedBlinkExit\(\)](#)

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

#### Returns

none.

#### 7.668.2.2 [LedBlinkInit\(\)](#)

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

#### Parameters

|                    |                                                                |
|--------------------|----------------------------------------------------------------|
| <i>interval_ms</i> | Specifies the desired LED blink interval time in milliseconds. |
|--------------------|----------------------------------------------------------------|



**Returns**

none.

**7.668.2.3 LedBlinkTask()**

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

**Returns**

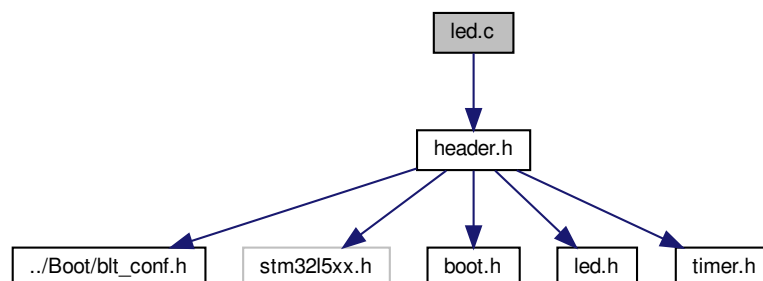
none.

**7.669 led.c File Reference**

LED driver source file.

```
#include "header.h"
```

Include dependency graph for ARMCM33\_STM32L5\_Nucleo\_L552ZE\_IAR/Prog/led.c:

**Macros**

- `#define LED_TOGGLE_MS (500)`  
*Toggle interval time in milliseconds.*

**Functions**

- void `LedInit` (void)  
*Initializes the LED.*
- void `LedToggle` (void)  
*Toggles the LED at a fixed time interval.*

### 7.669.1 Detailed Description

LED driver source file.

### 7.669.2 Function Documentation

#### 7.669.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

##### Returns

none.

#### 7.669.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

##### Returns

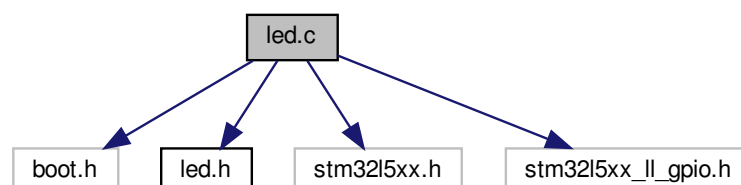
none.

## 7.670 led.c File Reference

LED driver source file.

```
#include "boot.h"
#include "led.h"
#include "stm32l5xx.h"
#include "stm32l5xx_ll_gpio.h"
```

Include dependency graph for ARMC33\_STM32L5\_Nucleo\_L552ZE\_Keil/Boot/led.c:



## Functions

- void [LedBlinkInit](#) ([blt\\_int16u](#) interval\_ms)  
*Initializes the LED blink driver.*
- void [LedBlinkTask](#) (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void [LedBlinkExit](#) (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

## Variables

- static [blt\\_int16u](#) [ledBlinkIntervalMs](#)  
*Holds the desired LED blink interval time.*

### 7.670.1 Detailed Description

LED driver source file.

### 7.670.2 Function Documentation

#### 7.670.2.1 LedBlinkExit()

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

#### Returns

none.

#### 7.670.2.2 LedBlinkInit()

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

#### Parameters

|                    |                                                                |
|--------------------|----------------------------------------------------------------|
| <i>interval_ms</i> | Specifies the desired LED blink interval time in milliseconds. |
|--------------------|----------------------------------------------------------------|

**Returns**

none.

**7.670.2.3 LedBlinkTask()**

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

**Returns**

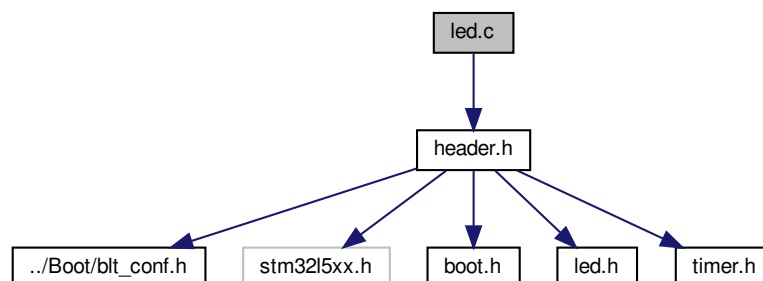
none.

**7.671 led.c File Reference**

LED driver source file.

```
#include "header.h"
```

Include dependency graph for ARMCM33\_STM32L5\_Nucleo\_L552ZE\_Keil/Prog/led.c:

**Macros**

- `#define LED_TOGGLE_MS (500)`  
*Toggle interval time in milliseconds.*

**Functions**

- `void LedInit (void)`  
*Initializes the LED.*
- `void LedToggle (void)`  
*Toggles the LED at a fixed time interval.*

### 7.671.1 Detailed Description

LED driver source file.

### 7.671.2 Function Documentation

#### 7.671.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

**Returns**

none.

#### 7.671.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

**Returns**

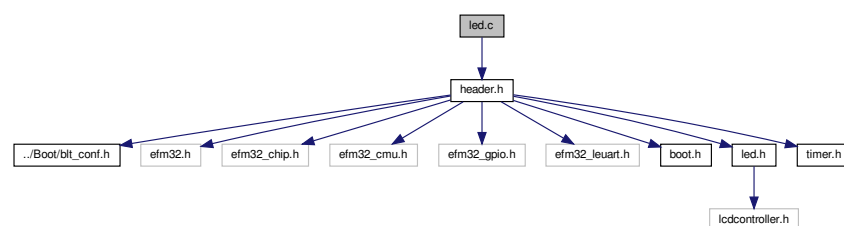
none.

## 7.672 led.c File Reference

LED driver source file.

```
#include "header.h"
```

Include dependency graph for ARMCM3\_EFM32\_Olimex\_EM32G880F128STK\_GCC/Prog/led.c:



## Macros

- `#define LED_TOGGLE_MS (500)`  
*Toggle interval time in milliseconds.*

## Functions

- `void LedInit (void)`  
*Initializes the LED. The board doesn't have a dedicated LED so an indicator on the LCD is used instead.*
- `void LedToggle (void)`  
*Toggles the LED at a fixed time interval.*

### 7.672.1 Detailed Description

LED driver source file.

### 7.672.2 Function Documentation

#### 7.672.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED. The board doesn't have a dedicated LED so an indicator on the LCD is used instead.

Initializes the LED.

#### Returns

none.

#### 7.672.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

#### Returns

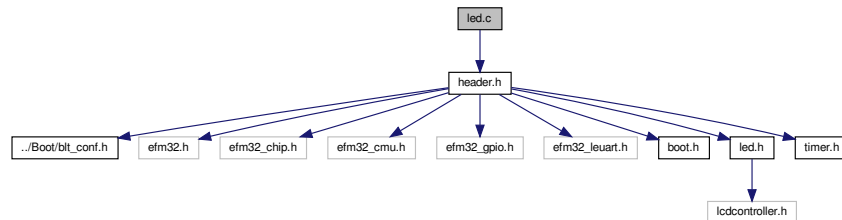
none.

## 7.673 led.c File Reference

LED driver source file.

```
#include "header.h"
```

Include dependency graph for ARMCM3\_EFM32\_Olimex\_EM32G880F128STK\_IAR/Prog/led.c:



### Macros

- `#define LED_TOGGLE_MS (500)`  
*Toggle interval time in milliseconds.*

### Functions

- `void LedInit (void)`  
*Initializes the LED. The board doesn't have a dedicted LED so an indicator on the LCD is used instead.*
- `void LedToggle (void)`  
*Toggles the LED at a fixed time interval.*

### 7.673.1 Detailed Description

LED driver source file.

### 7.673.2 Function Documentation

#### 7.673.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED. The board doesn't have a dedicted LED so an indicator on the LCD is used instead.

Initializes the LED.

#### Returns

none.

### 7.673.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

#### Returns

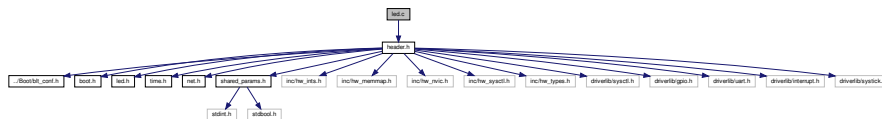
none.

## 7.674 led.c File Reference

LED driver source file.

```
#include "header.h"
```

Include dependency graph for ARMC3M3\_LM3S\_EK\_LM3S6965\_GCC/Prog/led.c:



## Macros

- #define [LED\\_TOGGLE\\_MS](#) (500)  
*Toggle interval time in milliseconds.*

## Functions

- void [LedInit](#) (void)  
*Initializes the LED.*
- void [LedToggle](#) (void)  
*Toggles the LED at a fixed time interval.*

### 7.674.1 Detailed Description

LED driver source file.

### 7.674.2 Function Documentation



## 7.674.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

## Returns

none.

## 7.674.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

## Returns

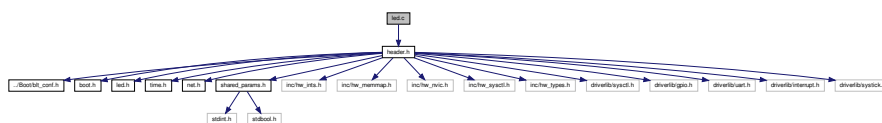
none.

## 7.675 led.c File Reference

LED driver source file.

```
#include "header.h"
```

Include dependency graph for ARMC3M3\_LM3S\_EK\_LM3S6965\_IAR/Prog/led.c:



## Macros

- `#define LED_TOGGLE_MS (500)`  
*Toggle interval time in milliseconds.*

## Functions

- `void LedInit (void)`  
*Initializes the LED.*
- `void LedToggle (void)`  
*Toggles the LED at a fixed time interval.*

### 7.675.1 Detailed Description

LED driver source file.

### 7.675.2 Function Documentation

#### 7.675.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

##### Returns

none.

#### 7.675.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

##### Returns

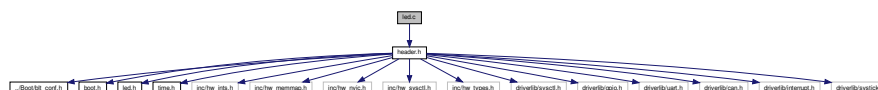
none.

## 7.676 led.c File Reference

LED driver source file.

```
#include "header.h"
```

Include dependency graph for ARMCM3\_LM3S\_EK\_LM3S8962\_GCC/Prog/led.c:



## Macros

- `#define LED_TOGGLE_MS (500)`  
*Toggle interval time in milliseconds.*

## Functions

- `void LedInit (void)`  
*Initializes the LED.*
- `void LedToggle (void)`  
*Toggles the LED at a fixed time interval.*

### 7.676.1 Detailed Description

LED driver source file.

### 7.676.2 Function Documentation

#### 7.676.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

##### Returns

none.

#### 7.676.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

##### Returns

none.

## 7.677 led.c File Reference

LED driver source file.

```
#include "header.h"
```

Include dependency graph for ARMCM3\_LM3S\_EK\_LM3S8962\_IAR/Prog/led.c:



## Macros

- #define [LED\\_TOGGLE\\_MS](#) (500)  
*Toggle interval time in milliseconds.*

## Functions

- void [LedInit](#) (void)  
*Initializes the LED.*
- void [LedToggle](#) (void)  
*Toggles the LED at a fixed time interval.*

### 7.677.1 Detailed Description

LED driver source file.

### 7.677.2 Function Documentation

#### 7.677.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

#### Returns

none.

### 7.677.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

#### Returns

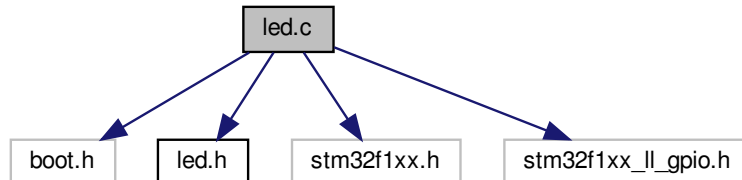
none.

## 7.678 led.c File Reference

LED driver source file.

```
#include "boot.h"
#include "led.h"
#include "stm32f1xx.h"
#include "stm32f1xx_ll_gpio.h"
```

Include dependency graph for ARMCM3\_STM32F1\_Nucleo\_F103RB\_CubeIDE/Boot/App/led.c:



## Functions

- void [LedBlinkInit](#) ([blt\\_int16u](#) interval\_ms)  
*Initializes the LED blink driver.*
- void [LedBlinkTask](#) (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void [LedBlinkExit](#) (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

## Variables

- static [blt\\_int16u](#) [ledBlinkIntervalMs](#)  
*Holds the desired LED blink interval time.*

### 7.678.1 Detailed Description

LED driver source file.

### 7.678.2 Function Documentation

#### 7.678.2.1 LedBlinkExit()

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

##### Returns

none.

#### 7.678.2.2 LedBlinkInit()

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

##### Parameters

|                    |                                                                |
|--------------------|----------------------------------------------------------------|
| <i>interval_ms</i> | Specifies the desired LED blink interval time in milliseconds. |
|--------------------|----------------------------------------------------------------|

##### Returns

none.

#### 7.678.2.3 LedBlinkTask()

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

##### Returns

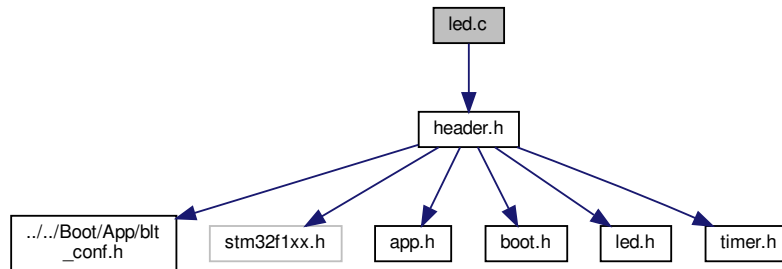
none.

## 7.679 led.c File Reference

LED driver source file.

```
#include "header.h"
```

Include dependency graph for ARMCM3\_STM32F1\_Nucleo\_F103RB\_CubeIDE/Prog/App/led.c:



### Macros

- `#define LED_TOGGLE_MS (500)`  
*Toggle interval time in milliseconds.*

### Functions

- `void LedInit (void)`  
*Initializes the LED.*
- `void LedToggle (void)`  
*Toggles the LED at a fixed time interval.*

#### 7.679.1 Detailed Description

LED driver source file.

#### 7.679.2 Function Documentation

##### 7.679.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

#### Returns

none.

### 7.679.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

#### Returns

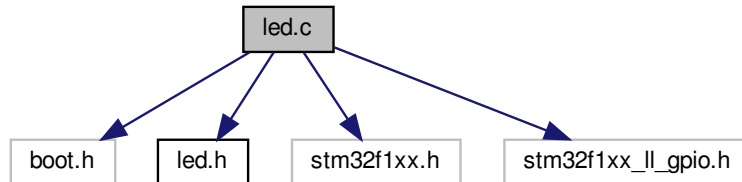
none.

## 7.680 led.c File Reference

LED driver source file.

```
#include "boot.h"
#include "led.h"
#include "stm32f1xx.h"
#include "stm32f1xx_ll_gpio.h"
```

Include dependency graph for ARMCM3\_STM32F1\_Nucleo\_F103RB\_GCC/Boot/led.c:



## Functions

- void [LedBlinkInit](#) ([blt\\_int16u](#) interval\_ms)  
*Initializes the LED blink driver.*
- void [LedBlinkTask](#) (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void [LedBlinkExit](#) (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

## Variables

- static [blt\\_int16u](#) [ledBlinkIntervalMs](#)  
*Holds the desired LED blink interval time.*



## 7.680.1 Detailed Description

LED driver source file.

## 7.680.2 Function Documentation

### 7.680.2.1 LedBlinkExit()

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

#### Returns

none.

### 7.680.2.2 LedBlinkInit()

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

#### Parameters

|                    |                                                                |
|--------------------|----------------------------------------------------------------|
| <i>interval_ms</i> | Specifies the desired LED blink interval time in milliseconds. |
|--------------------|----------------------------------------------------------------|

#### Returns

none.

### 7.680.2.3 LedBlinkTask()

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

#### Returns

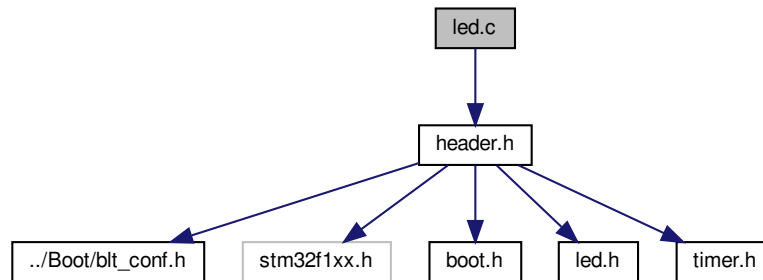
none.

## 7.681 led.c File Reference

LED driver source file.

```
#include "header.h"
```

Include dependency graph for ARMCM3\_STM32F1\_Nucleo\_F103RB\_GCC/Prog/led.c:



### Macros

- `#define LED_TOGGLE_MS (500)`  
*Toggle interval time in milliseconds.*

### Functions

- `void LedInit (void)`  
*Initializes the LED.*
- `void LedToggle (void)`  
*Toggles the LED at a fixed time interval.*

### 7.681.1 Detailed Description

LED driver source file.

### 7.681.2 Function Documentation

#### 7.681.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

#### Returns

none.

### 7.681.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

#### Returns

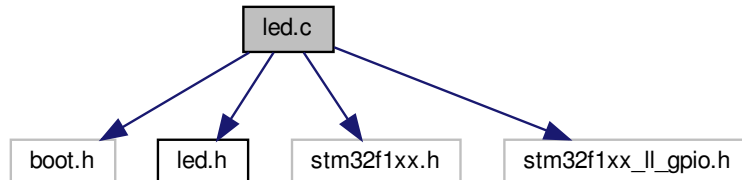
none.

## 7.682 led.c File Reference

LED driver source file.

```
#include "boot.h"
#include "led.h"
#include "stm32f1xx.h"
#include "stm32f1xx_ll_gpio.h"
```

Include dependency graph for ARMCM3\_STM32F1\_Nucleo\_F103RB\_IAR/Boot/led.c:



## Functions

- void [LedBlinkInit](#) ([blt\\_int16u](#) interval\_ms)  
*Initializes the LED blink driver.*
- void [LedBlinkTask](#) (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void [LedBlinkExit](#) (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

## Variables

- static [blt\\_int16u](#) [ledBlinkIntervalMs](#)  
*Holds the desired LED blink interval time.*

### 7.682.1 Detailed Description

LED driver source file.

### 7.682.2 Function Documentation

#### 7.682.2.1 LedBlinkExit()

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

##### Returns

none.

#### 7.682.2.2 LedBlinkInit()

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

##### Parameters

|                    |                                                                |
|--------------------|----------------------------------------------------------------|
| <i>interval_ms</i> | Specifies the desired LED blink interval time in milliseconds. |
|--------------------|----------------------------------------------------------------|

##### Returns

none.

#### 7.682.2.3 LedBlinkTask()

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

##### Returns

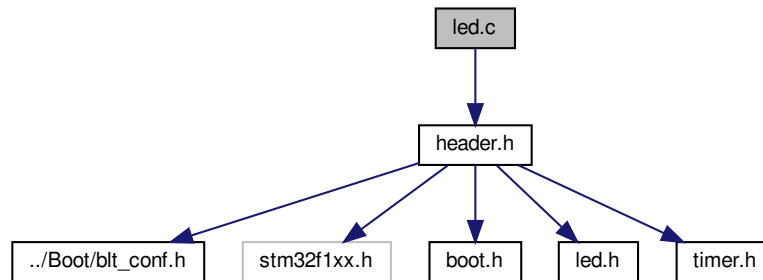
none.

## 7.683 led.c File Reference

LED driver source file.

```
#include "header.h"
```

Include dependency graph for ARMCM3\_STM32F1\_Nucleo\_F103RB\_IAR/Prog/led.c:



### Macros

- `#define LED_TOGGLE_MS (500)`  
*Toggle interval time in milliseconds.*

### Functions

- `void LedInit (void)`  
*Initializes the LED.*
- `void LedToggle (void)`  
*Toggles the LED at a fixed time interval.*

### 7.683.1 Detailed Description

LED driver source file.

### 7.683.2 Function Documentation

#### 7.683.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

Returns

none.

### 7.683.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

#### Returns

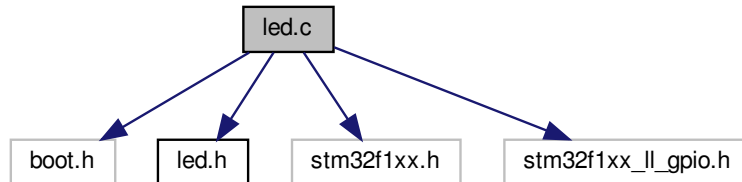
none.

## 7.684 led.c File Reference

LED driver source file.

```
#include "boot.h"
#include "led.h"
#include "stm32f1xx.h"
#include "stm32f1xx_ll_gpio.h"
```

Include dependency graph for ARMCM3\_STM32F1\_Nucleo\_F103RB\_Keil/Boot/led.c:



## Functions

- void [LedBlinkInit](#) ([blt\\_int16u](#) interval\_ms)  
*Initializes the LED blink driver.*
- void [LedBlinkTask](#) (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void [LedBlinkExit](#) (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

## Variables

- static [blt\\_int16u](#) [ledBlinkIntervalMs](#)  
*Holds the desired LED blink interval time.*

## 7.684.1 Detailed Description

LED driver source file.

## 7.684.2 Function Documentation

### 7.684.2.1 LedBlinkExit()

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

#### Returns

none.

### 7.684.2.2 LedBlinkInit()

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

#### Parameters

|                    |                                                                |
|--------------------|----------------------------------------------------------------|
| <i>interval_ms</i> | Specifies the desired LED blink interval time in milliseconds. |
|--------------------|----------------------------------------------------------------|

#### Returns

none.

### 7.684.2.3 LedBlinkTask()

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

#### Returns

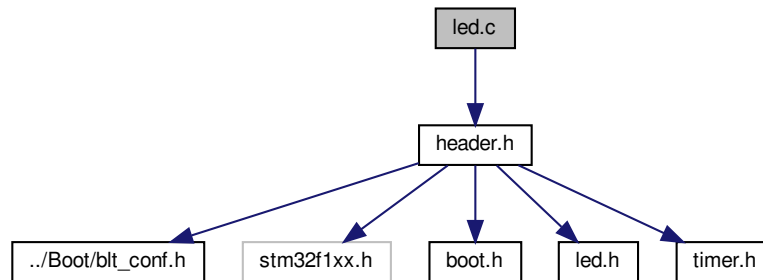
none.

## 7.685 led.c File Reference

LED driver source file.

```
#include "header.h"
```

Include dependency graph for ARMCM3\_STM32F1\_Nucleo\_F103RB\_Keil/Prog/led.c:



### Macros

- `#define LED_TOGGLE_MS (500)`  
*Toggle interval time in milliseconds.*

### Functions

- `void LedInit (void)`  
*Initializes the LED.*
- `void LedToggle (void)`  
*Toggles the LED at a fixed time interval.*

### 7.685.1 Detailed Description

LED driver source file.

### 7.685.2 Function Documentation

#### 7.685.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

#### Returns

none.



### 7.685.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

#### Returns

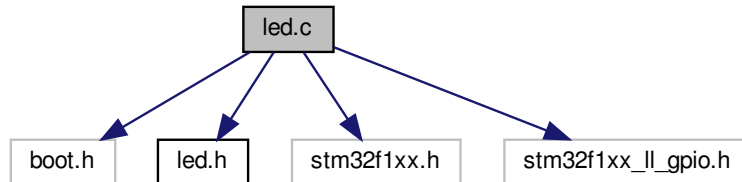
none.

## 7.686 led.c File Reference

LED driver source file.

```
#include "boot.h"
#include "led.h"
#include "stm32f1xx.h"
#include "stm32f1xx_ll_gpio.h"
```

Include dependency graph for ARMCM3\_STM32F1\_Olimex\_STM32H103\_CubeIDE/Boot/App/led.c:



## Functions

- void `LedBlinkInit` (`blt_int16u` interval\_ms)  
*Initializes the LED blink driver.*
- void `LedBlinkTask` (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void `LedBlinkExit` (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

## Variables

- static `blt_int16u` `ledBlinkIntervalMs`  
*Holds the desired LED blink interval time.*

### 7.686.1 Detailed Description

LED driver source file.

### 7.686.2 Function Documentation

#### 7.686.2.1 LedBlinkExit()

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

##### Returns

none.

#### 7.686.2.2 LedBlinkInit()

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

##### Parameters

|                    |                                                                |
|--------------------|----------------------------------------------------------------|
| <i>interval_ms</i> | Specifies the desired LED blink interval time in milliseconds. |
|--------------------|----------------------------------------------------------------|

##### Returns

none.

#### 7.686.2.3 LedBlinkTask()

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

##### Returns

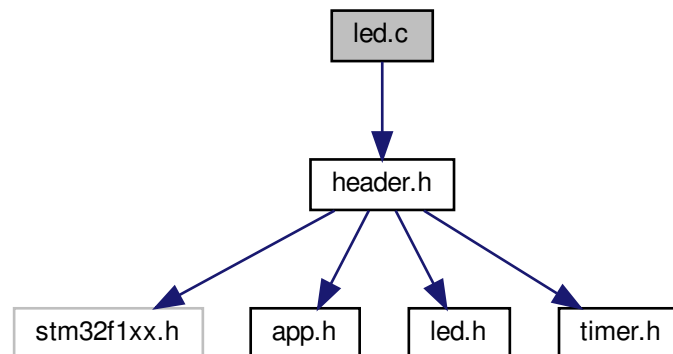
none.

## 7.687 led.c File Reference

LED driver source file.

```
#include "header.h"
```

Include dependency graph for ARMCM3\_STM32F1\_Olimex\_STM32H103\_CubeIDE/Prog/App/led.c:



### Macros

- `#define LED_TOGGLE_MS (500)`  
*Toggle interval time in milliseconds.*

### Functions

- `void LedInit (void)`  
*Initializes the LED.*
- `void LedToggle (void)`  
*Toggles the LED at a fixed time interval.*

### 7.687.1 Detailed Description

LED driver source file.

### 7.687.2 Function Documentation

### 7.687.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

#### Returns

none.

### 7.687.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

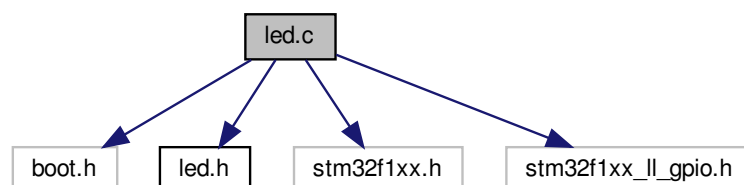
#### Returns

none.

## 7.688 led.c File Reference

LED driver source file.

```
#include "boot.h"
#include "led.h"
#include "stm32f1xx.h"
#include "stm32f1xx_ll_gpio.h"
Include dependency graph for ARMCM3_STM32F1_Olimex_STM32H103_GCC/Boot/led.c:
```



## Functions

- void [LedBlinkInit](#) ([blt\\_int16u](#) interval\_ms)  
*Initializes the LED blink driver.*
- void [LedBlinkTask](#) (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void [LedBlinkExit](#) (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

## Variables

- static [blt\\_int16u](#) [ledBlinkIntervalMs](#)  
*Holds the desired LED blink interval time.*

### 7.688.1 Detailed Description

LED driver source file.

### 7.688.2 Function Documentation

#### 7.688.2.1 LedBlinkExit()

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

#### Returns

none.

#### 7.688.2.2 LedBlinkInit()

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

#### Parameters

|                    |                                                                |
|--------------------|----------------------------------------------------------------|
| <i>interval_ms</i> | Specifies the desired LED blink interval time in milliseconds. |
|--------------------|----------------------------------------------------------------|

**Returns**

none.

**7.688.2.3 LedBlinkTask()**

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

**Returns**

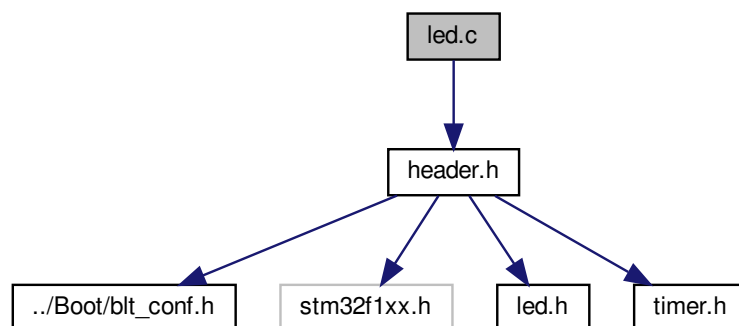
none.

**7.689 led.c File Reference**

LED driver source file.

```
#include "header.h"
```

Include dependency graph for ARMC32F1\_Olimex\_STM32H103\_GCC/Prog/led.c:

**Macros**

- `#define LED_TOGGLE_MS (500)`  
*Toggle interval time in milliseconds.*

**Functions**

- void `LedInit` (void)  
*Initializes the LED.*
- void `LedToggle` (void)  
*Toggles the LED at a fixed time interval.*

## 7.689.1 Detailed Description

LED driver source file.

## 7.689.2 Function Documentation

### 7.689.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

#### Returns

none.

### 7.689.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

#### Returns

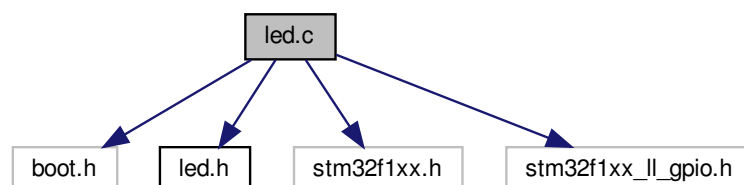
none.

## 7.690 led.c File Reference

LED driver source file.

```
#include "boot.h"
#include "led.h"
#include "stm32f1xx.h"
#include "stm32f1xx_ll_gpio.h"
```

Include dependency graph for ARMC3M3\_STM32F1\_Olimex\_STM32H103\_IAR/Boot/led.c:



## Functions

- void `LedBlinkInit` (`blt_int16u` interval\_ms)  
*Initializes the LED blink driver.*
- void `LedBlinkTask` (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void `LedBlinkExit` (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

## Variables

- static `blt_int16u` `ledBlinkIntervalMs`  
*Holds the desired LED blink interval time.*

### 7.690.1 Detailed Description

LED driver source file.

### 7.690.2 Function Documentation

#### 7.690.2.1 `LedBlinkExit()`

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

#### Returns

none.

#### 7.690.2.2 `LedBlinkInit()`

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

#### Parameters

|                          |                                                                |
|--------------------------|----------------------------------------------------------------|
| <code>interval_ms</code> | Specifies the desired LED blink interval time in milliseconds. |
|--------------------------|----------------------------------------------------------------|



**Returns**

none.

**7.690.2.3 LedBlinkTask()**

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

**Returns**

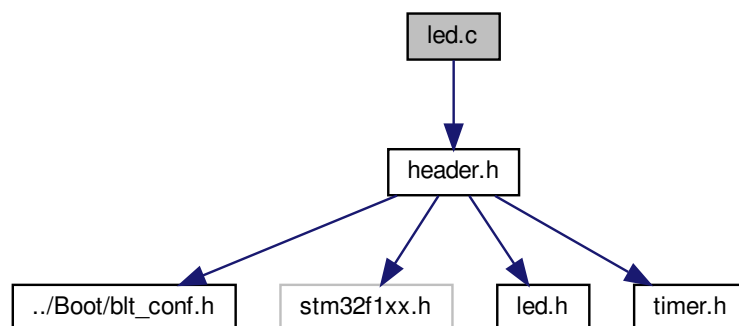
none.

**7.691 led.c File Reference**

LED driver source file.

```
#include "header.h"
```

Include dependency graph for ARMCM3\_STM32F1\_Olimex\_STM32H103\_IAR/Prog/led.c:

**Macros**

- `#define LED_TOGGLE_MS (500)`  
*Toggle interval time in milliseconds.*

**Functions**

- void `LedInit` (void)  
*Initializes the LED.*
- void `LedToggle` (void)  
*Toggles the LED at a fixed time interval.*

### 7.691.1 Detailed Description

LED driver source file.

### 7.691.2 Function Documentation

#### 7.691.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

##### Returns

none.

#### 7.691.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

##### Returns

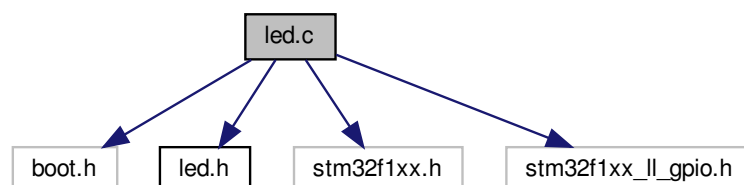
none.

## 7.692 led.c File Reference

LED driver source file.

```
#include "boot.h"
#include "led.h"
#include "stm32f1xx.h"
#include "stm32f1xx_ll_gpio.h"
```

Include dependency graph for ARMC3M3\_STM32F1\_Olimex\_STM32H103\_Keil/Boot/led.c:



## Functions

- void [LedBlinkInit](#) ([blt\\_int16u](#) interval\_ms)  
*Initializes the LED blink driver.*
- void [LedBlinkTask](#) (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void [LedBlinkExit](#) (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

## Variables

- static [blt\\_int16u](#) [ledBlinkIntervalMs](#)  
*Holds the desired LED blink interval time.*

### 7.692.1 Detailed Description

LED driver source file.

### 7.692.2 Function Documentation

#### 7.692.2.1 LedBlinkExit()

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

#### Returns

none.

#### 7.692.2.2 LedBlinkInit()

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

#### Parameters

|                    |                                                                |
|--------------------|----------------------------------------------------------------|
| <i>interval_ms</i> | Specifies the desired LED blink interval time in milliseconds. |
|--------------------|----------------------------------------------------------------|

**Returns**

none.

**7.692.2.3 LedBlinkTask()**

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

**Returns**

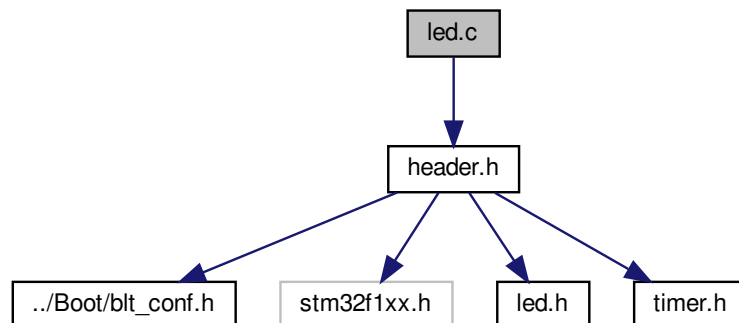
none.

**7.693 led.c File Reference**

LED driver source file.

```
#include "header.h"
```

Include dependency graph for ARMCM3\_STM32F1\_Olimex\_STM32H103\_Keil/Prog/led.c:

**Macros**

- `#define LED_TOGGLE_MS (500)`  
*Toggle interval time in milliseconds.*

**Functions**

- void `LedInit` (void)  
*Initializes the LED.*
- void `LedToggle` (void)  
*Toggles the LED at a fixed time interval.*

### 7.693.1 Detailed Description

LED driver source file.

### 7.693.2 Function Documentation

#### 7.693.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

##### Returns

none.

#### 7.693.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

##### Returns

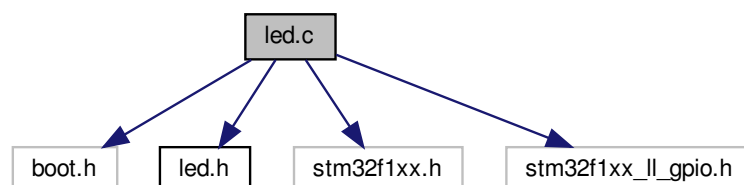
none.

## 7.694 led.c File Reference

LED driver source file.

```
#include "boot.h"
#include "led.h"
#include "stm32f1xx.h"
#include "stm32f1xx_ll_gpio.h"
```

Include dependency graph for ARMC3M3\_STM32F1\_Olimex\_STM32P103\_CubeIDE/Boot/App/led.c:



## Functions

- void `LedBlinkInit` (`blt_int16u` interval\_ms)  
*Initializes the LED blink driver.*
- void `LedBlinkTask` (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void `LedBlinkExit` (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

## Variables

- static `blt_int16u` `ledBlinkIntervalMs`  
*Holds the desired LED blink interval time.*

### 7.694.1 Detailed Description

LED driver source file.

### 7.694.2 Function Documentation

#### 7.694.2.1 `LedBlinkExit()`

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

#### Returns

none.

#### 7.694.2.2 `LedBlinkInit()`

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

#### Parameters

|                          |                                                                |
|--------------------------|----------------------------------------------------------------|
| <code>interval_ms</code> | Specifies the desired LED blink interval time in milliseconds. |
|--------------------------|----------------------------------------------------------------|

**Returns**

none.

**7.694.2.3 LedBlinkTask()**

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

**Returns**

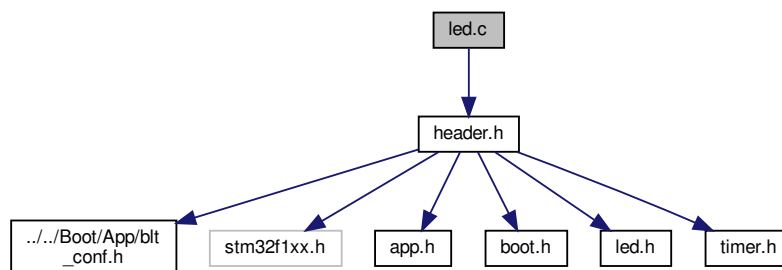
none.

**7.695 led.c File Reference**

LED driver source file.

```
#include "header.h"
```

Include dependency graph for ARMC32F1\_Olimex\_STM32P103\_CubeIDE/Prog/App/led.c:

**Macros**

- `#define LED_TOGGLE_MS (500)`  
*Toggle interval time in milliseconds.*

**Functions**

- `void LedInit (void)`  
*Initializes the LED.*
- `void LedToggle (void)`  
*Toggles the LED at a fixed time interval.*

### 7.695.1 Detailed Description

LED driver source file.

### 7.695.2 Function Documentation

#### 7.695.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

##### Returns

none.

#### 7.695.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

##### Returns

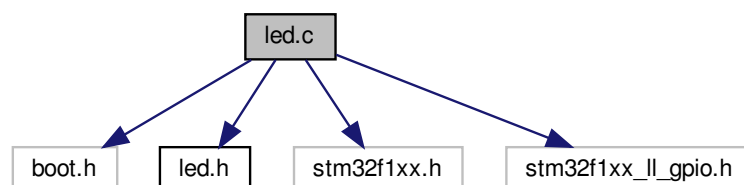
none.

## 7.696 led.c File Reference

LED driver source file.

```
#include "boot.h"
#include "led.h"
#include "stm32f1xx.h"
#include "stm32f1xx_ll_gpio.h"
```

Include dependency graph for ARMC3M3\_STM32F1\_Olimex\_STM32P103\_GCC/Boot/led.c:





## Functions

- void [LedBlinkInit](#) ([blt\\_int16u](#) interval\_ms)  
*Initializes the LED blink driver.*
- void [LedBlinkTask](#) (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void [LedBlinkExit](#) (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

## Variables

- static [blt\\_int16u](#) [ledBlinkIntervalMs](#)  
*Holds the desired LED blink interval time.*

### 7.696.1 Detailed Description

LED driver source file.

### 7.696.2 Function Documentation

#### 7.696.2.1 LedBlinkExit()

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

#### Returns

none.

#### 7.696.2.2 LedBlinkInit()

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

#### Parameters

|                    |                                                                |
|--------------------|----------------------------------------------------------------|
| <i>interval_ms</i> | Specifies the desired LED blink interval time in milliseconds. |
|--------------------|----------------------------------------------------------------|

**Returns**

none.

**7.696.2.3 LedBlinkTask()**

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

**Returns**

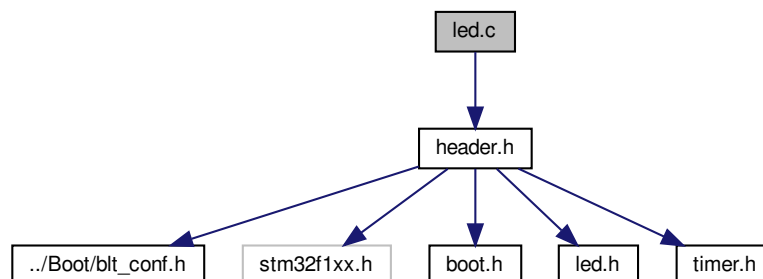
none.

**7.697 led.c File Reference**

LED driver source file.

```
#include "header.h"
```

Include dependency graph for ARMCM3\_STM32F1\_Olimex\_STM32P103\_GCC/Prog/led.c:

**Macros**

- `#define LED_TOGGLE_MS (500)`  
*Toggle interval time in milliseconds.*

**Functions**

- void `LedInit` (void)  
*Initializes the LED.*
- void `LedToggle` (void)  
*Toggles the LED at a fixed time interval.*

### 7.697.1 Detailed Description

LED driver source file.

### 7.697.2 Function Documentation

#### 7.697.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

##### Returns

none.

#### 7.697.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

##### Returns

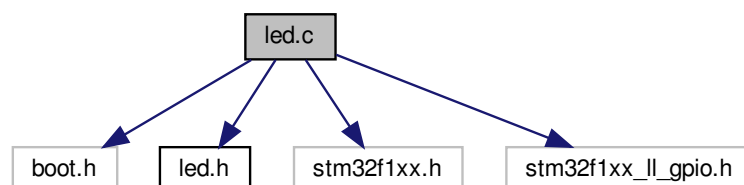
none.

## 7.698 led.c File Reference

LED driver source file.

```
#include "boot.h"
#include "led.h"
#include "stm32f1xx.h"
#include "stm32f1xx_ll_gpio.h"
```

Include dependency graph for ARMC3M3\_STM32F1\_Olimex\_STM32P103\_IAR/Boot/led.c:



## Functions

- void `LedBlinkInit` (`blt_int16u` interval\_ms)  
*Initializes the LED blink driver.*
- void `LedBlinkTask` (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void `LedBlinkExit` (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

## Variables

- static `blt_int16u` `ledBlinkIntervalMs`  
*Holds the desired LED blink interval time.*

### 7.698.1 Detailed Description

LED driver source file.

### 7.698.2 Function Documentation

#### 7.698.2.1 `LedBlinkExit()`

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

#### Returns

none.

#### 7.698.2.2 `LedBlinkInit()`

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

#### Parameters

|                          |                                                                |
|--------------------------|----------------------------------------------------------------|
| <code>interval_ms</code> | Specifies the desired LED blink interval time in milliseconds. |
|--------------------------|----------------------------------------------------------------|

**Returns**

none.

**7.698.2.3 LedBlinkTask()**

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

**Returns**

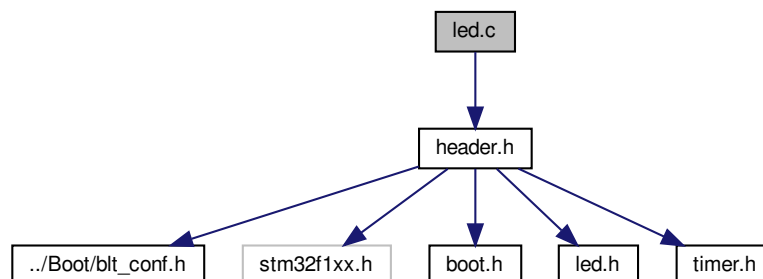
none.

**7.699 led.c File Reference**

LED driver source file.

```
#include "header.h"
```

Include dependency graph for ARMCM3\_STM32F1\_Olimex\_STM32P103\_IAR/Prog/led.c:

**Macros**

- `#define LED_TOGGLE_MS (500)`  
*Toggle interval time in milliseconds.*

**Functions**

- void `LedInit` (void)  
*Initializes the LED.*
- void `LedToggle` (void)  
*Toggles the LED at a fixed time interval.*

## 7.699.1 Detailed Description

LED driver source file.

## 7.699.2 Function Documentation

### 7.699.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

#### Returns

none.

### 7.699.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

#### Returns

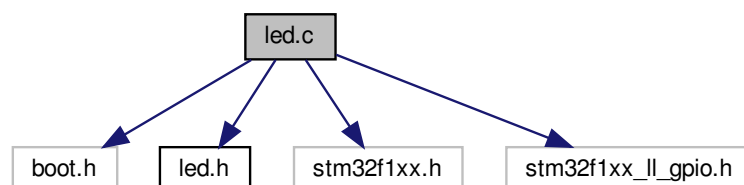
none.

## 7.700 led.c File Reference

LED driver source file.

```
#include "boot.h"
#include "led.h"
#include "stm32f1xx.h"
#include "stm32f1xx_ll_gpio.h"
```

Include dependency graph for ARMC3M3\_STM32F1\_Olimex\_STM32P103\_Keil/Boot/led.c:



## Functions

- void [LedBlinkInit](#) ([blt\\_int16u](#) interval\_ms)  
*Initializes the LED blink driver.*
- void [LedBlinkTask](#) (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void [LedBlinkExit](#) (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

## Variables

- static [blt\\_int16u](#) [ledBlinkIntervalMs](#)  
*Holds the desired LED blink interval time.*

### 7.700.1 Detailed Description

LED driver source file.

### 7.700.2 Function Documentation

#### 7.700.2.1 LedBlinkExit()

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

#### Returns

none.

#### 7.700.2.2 LedBlinkInit()

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

#### Parameters

|                    |                                                                |
|--------------------|----------------------------------------------------------------|
| <i>interval_ms</i> | Specifies the desired LED blink interval time in milliseconds. |
|--------------------|----------------------------------------------------------------|

**Returns**

none.

**7.700.2.3 LedBlinkTask()**

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

**Returns**

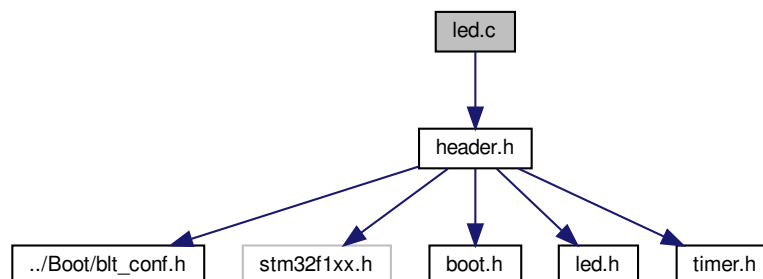
none.

**7.701 led.c File Reference**

LED driver source file.

```
#include "header.h"
```

Include dependency graph for ARMCM3\_STM32F1\_Olimex\_STM32P103\_Keil/Prog/led.c:

**Macros**

- `#define LED_TOGGLE_MS (500)`  
*Toggle interval time in milliseconds.*

**Functions**

- void `LedInit` (void)  
*Initializes the LED.*
- void `LedToggle` (void)  
*Toggles the LED at a fixed time interval.*



### 7.701.1 Detailed Description

LED driver source file.

### 7.701.2 Function Documentation

#### 7.701.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

##### Returns

none.

#### 7.701.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

##### Returns

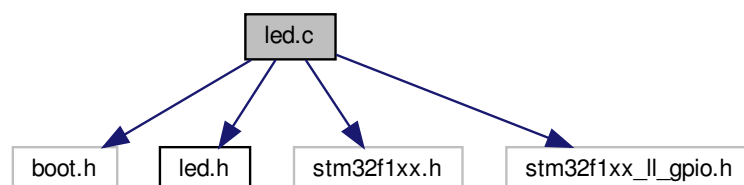
none.

## 7.702 led.c File Reference

LED driver source file.

```
#include "boot.h"
#include "led.h"
#include "stm32f1xx.h"
#include "stm32f1xx_ll_gpio.h"
```

Include dependency graph for ARMC3M3\_STM32F1\_Olimexino\_STM32\_CubelIDE/Boot/App/led.c:



## Functions

- void `LedBlinkInit` (`blt_int16u` interval\_ms)  
*Initializes the LED blink driver.*
- void `LedBlinkTask` (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void `LedBlinkExit` (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

## Variables

- static `blt_int16u` `ledBlinkIntervalMs`  
*Holds the desired LED blink interval time.*

### 7.702.1 Detailed Description

LED driver source file.

### 7.702.2 Function Documentation

#### 7.702.2.1 LedBlinkExit()

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

#### Returns

none.

#### 7.702.2.2 LedBlinkInit()

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

#### Parameters

|                          |                                                                |
|--------------------------|----------------------------------------------------------------|
| <code>interval_ms</code> | Specifies the desired LED blink interval time in milliseconds. |
|--------------------------|----------------------------------------------------------------|

**Returns**

none.

**7.702.2.3 LedBlinkTask()**

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

**Returns**

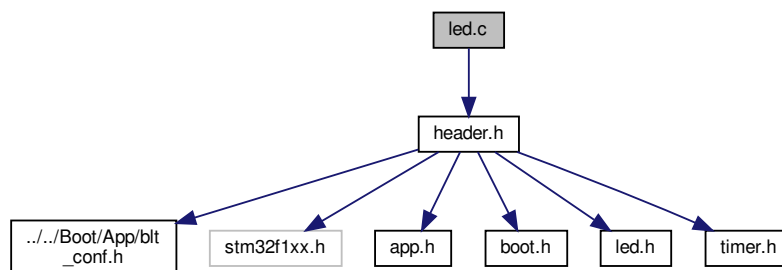
none.

**7.703 led.c File Reference**

LED driver source file.

```
#include "header.h"
```

Include dependency graph for ARMC32F1\_Olimexino\_STM32\_CubeIDE/Prog/App/led.c:

**Macros**

- `#define LED_TOGGLE_MS (500)`  
*Toggle interval time in milliseconds.*

**Functions**

- void `LedInit` (void)  
*Initializes the LED.*
- void `LedToggle` (void)  
*Toggles the LED at a fixed time interval.*

### 7.703.1 Detailed Description

LED driver source file.

### 7.703.2 Function Documentation

#### 7.703.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

##### Returns

none.

#### 7.703.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

##### Returns

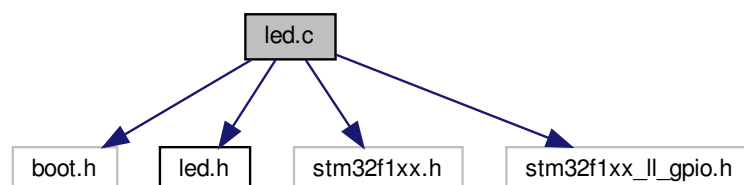
none.

## 7.704 led.c File Reference

LED driver source file.

```
#include "boot.h"
#include "led.h"
#include "stm32f1xx.h"
#include "stm32f1xx_ll_gpio.h"
```

Include dependency graph for ARMC32F1\_Olimexino\_STM32\_GCC/Boot/led.c:



## Functions

- void [LedBlinkInit](#) ([blt\\_int16u](#) interval\_ms)  
*Initializes the LED blink driver.*
- void [LedBlinkTask](#) (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void [LedBlinkExit](#) (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

## Variables

- static [blt\\_int16u](#) [ledBlinkIntervalMs](#)  
*Holds the desired LED blink interval time.*

### 7.704.1 Detailed Description

LED driver source file.

### 7.704.2 Function Documentation

#### 7.704.2.1 LedBlinkExit()

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

#### Returns

none.

#### 7.704.2.2 LedBlinkInit()

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

#### Parameters

|                    |                                                                |
|--------------------|----------------------------------------------------------------|
| <i>interval_ms</i> | Specifies the desired LED blink interval time in milliseconds. |
|--------------------|----------------------------------------------------------------|

**Returns**

none.

**7.704.2.3 LedBlinkTask()**

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

**Returns**

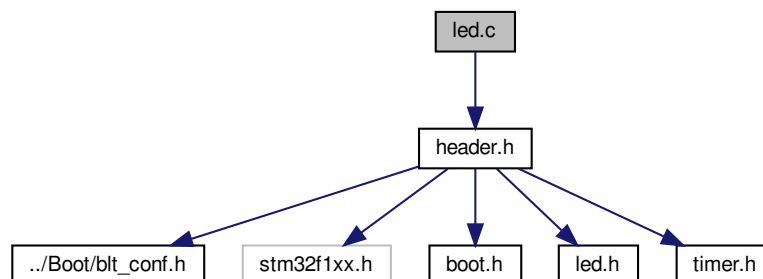
none.

**7.705 led.c File Reference**

LED driver source file.

```
#include "header.h"
```

Include dependency graph for ARMCM3\_STM32F1\_Olimexino\_STM32\_GCC/Prog/led.c:

**Macros**

- `#define LED_TOGGLE_MS (500)`  
*Toggle interval time in milliseconds.*

**Functions**

- void `LedInit` (void)  
*Initializes the LED.*
- void `LedToggle` (void)  
*Toggles the LED at a fixed time interval.*

### 7.705.1 Detailed Description

LED driver source file.

### 7.705.2 Function Documentation

#### 7.705.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

##### Returns

none.

#### 7.705.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

##### Returns

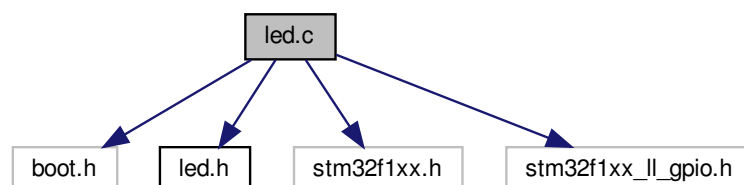
none.

## 7.706 led.c File Reference

LED driver source file.

```
#include "boot.h"
#include "led.h"
#include "stm32f1xx.h"
#include "stm32f1xx_ll_gpio.h"
```

Include dependency graph for ARMC3M3\_STM32F1\_Olimexino\_STM32\_IAR/Boot/led.c:



## Functions

- void `LedBlinkInit` (`blt_int16u` interval\_ms)  
*Initializes the LED blink driver.*
- void `LedBlinkTask` (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void `LedBlinkExit` (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

## Variables

- static `blt_int16u` `ledBlinkIntervalMs`  
*Holds the desired LED blink interval time.*

### 7.706.1 Detailed Description

LED driver source file.

### 7.706.2 Function Documentation

#### 7.706.2.1 `LedBlinkExit()`

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

#### Returns

none.

#### 7.706.2.2 `LedBlinkInit()`

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

#### Parameters

|                          |                                                                |
|--------------------------|----------------------------------------------------------------|
| <code>interval_ms</code> | Specifies the desired LED blink interval time in milliseconds. |
|--------------------------|----------------------------------------------------------------|



**Returns**

none.

**7.706.2.3 LedBlinkTask()**

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

**Returns**

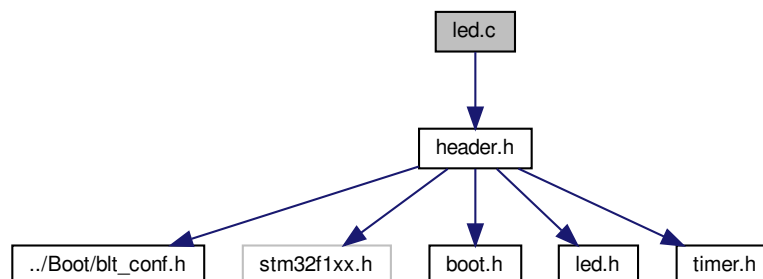
none.

**7.707 led.c File Reference**

LED driver source file.

```
#include "header.h"
```

Include dependency graph for ARMCM3\_STM32F1\_Olimexino\_STM32\_IAR/Prog/led.c:

**Macros**

- `#define LED_TOGGLE_MS (500)`  
*Toggle interval time in milliseconds.*

**Functions**

- void `LedInit` (void)  
*Initializes the LED.*
- void `LedToggle` (void)  
*Toggles the LED at a fixed time interval.*

### 7.707.1 Detailed Description

LED driver source file.

### 7.707.2 Function Documentation

#### 7.707.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

##### Returns

none.

#### 7.707.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

##### Returns

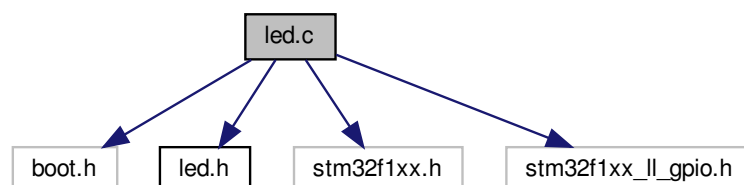
none.

## 7.708 led.c File Reference

LED driver source file.

```
#include "boot.h"
#include "led.h"
#include "stm32f1xx.h"
#include "stm32f1xx_ll_gpio.h"
```

Include dependency graph for ARMC3M3\_STM32F1\_Olimexino\_STM32\_Keil/Boot/led.c:



## Functions

- void [LedBlinkInit](#) ([blt\\_int16u](#) interval\_ms)  
*Initializes the LED blink driver.*
- void [LedBlinkTask](#) (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void [LedBlinkExit](#) (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

## Variables

- static [blt\\_int16u](#) [ledBlinkIntervalMs](#)  
*Holds the desired LED blink interval time.*

### 7.708.1 Detailed Description

LED driver source file.

### 7.708.2 Function Documentation

#### 7.708.2.1 LedBlinkExit()

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

#### Returns

none.

#### 7.708.2.2 LedBlinkInit()

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

#### Parameters

|                    |                                                                |
|--------------------|----------------------------------------------------------------|
| <i>interval_ms</i> | Specifies the desired LED blink interval time in milliseconds. |
|--------------------|----------------------------------------------------------------|

**Returns**

none.

**7.708.2.3 LedBlinkTask()**

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

**Returns**

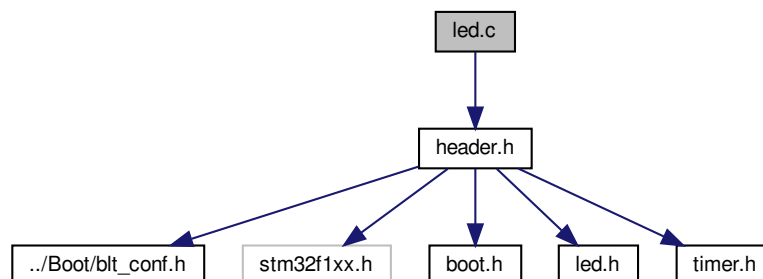
none.

**7.709 led.c File Reference**

LED driver source file.

```
#include "header.h"
```

Include dependency graph for ARMCM3\_STM32F1\_Olimexino\_STM32\_Keil/Prog/led.c:

**Macros**

- `#define LED_TOGGLE_MS (500)`  
*Toggle interval time in milliseconds.*

**Functions**

- `void LedInit (void)`  
*Initializes the LED.*
- `void LedToggle (void)`  
*Toggles the LED at a fixed time interval.*

### 7.709.1 Detailed Description

LED driver source file.

### 7.709.2 Function Documentation

#### 7.709.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

##### Returns

none.

#### 7.709.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

##### Returns

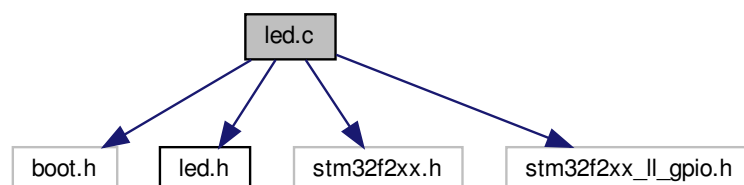
none.

## 7.710 led.c File Reference

LED driver source file.

```
#include "boot.h"
#include "led.h"
#include "stm32f2xx.h"
#include "stm32f2xx_ll_gpio.h"
```

Include dependency graph for ARMC3M3\_STM32F2\_Olimex\_STM32P207\_CubeIDE/Boot/App/led.c:



## Functions

- void `LedBlinkInit` (`blt_int16u` interval\_ms)  
*Initializes the LED blink driver.*
- void `LedBlinkTask` (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void `LedBlinkExit` (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

## Variables

- static `blt_int16u` `ledBlinkIntervalMs`  
*Holds the desired LED blink interval time.*

### 7.710.1 Detailed Description

LED driver source file.

### 7.710.2 Function Documentation

#### 7.710.2.1 `LedBlinkExit()`

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

#### Returns

none.

#### 7.710.2.2 `LedBlinkInit()`

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

#### Parameters

|                          |                                                                |
|--------------------------|----------------------------------------------------------------|
| <code>interval_ms</code> | Specifies the desired LED blink interval time in milliseconds. |
|--------------------------|----------------------------------------------------------------|

**Returns**

none.

**7.710.2.3 LedBlinkTask()**

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

**Returns**

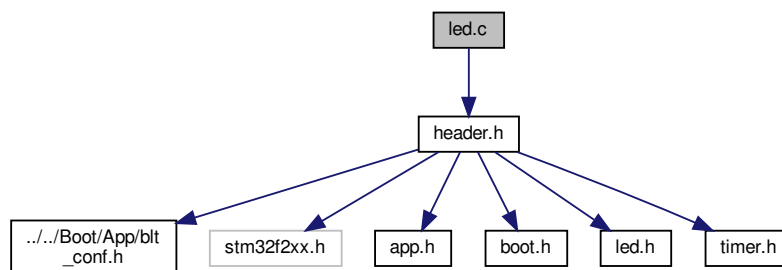
none.

**7.711 led.c File Reference**

LED driver source file.

```
#include "header.h"
```

Include dependency graph for ARMC32F2\_Olimex\_STM32P207\_CubeIDE/Prog/App/led.c:

**Macros**

- `#define LED_TOGGLE_MS (500)`  
*Toggle interval time in milliseconds.*

**Functions**

- `void LedInit (void)`  
*Initializes the LED.*
- `void LedToggle (void)`  
*Toggles the LED at a fixed time interval.*

### 7.711.1 Detailed Description

LED driver source file.

### 7.711.2 Function Documentation

#### 7.711.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

**Returns**

none.

#### 7.711.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

**Returns**

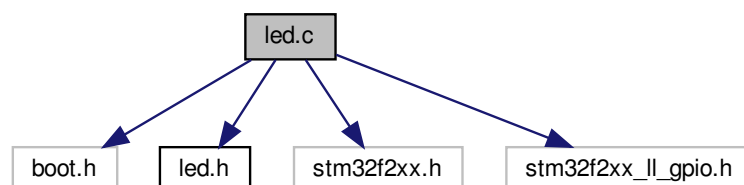
none.

## 7.712 led.c File Reference

LED driver source file.

```
#include "boot.h"
#include "led.h"
#include "stm32f2xx.h"
#include "stm32f2xx_ll_gpio.h"
```

Include dependency graph for ARMC3M3\_STM32F2\_Olimex\_STM32P207\_GCC/Boot/led.c:





## Functions

- void [LedBlinkInit](#) ([blt\\_int16u](#) interval\_ms)  
*Initializes the LED blink driver.*
- void [LedBlinkTask](#) (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void [LedBlinkExit](#) (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

## Variables

- static [blt\\_int16u](#) [ledBlinkIntervalMs](#)  
*Holds the desired LED blink interval time.*

### 7.712.1 Detailed Description

LED driver source file.

### 7.712.2 Function Documentation

#### 7.712.2.1 LedBlinkExit()

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

#### Returns

none.

#### 7.712.2.2 LedBlinkInit()

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

#### Parameters

|                    |                                                                |
|--------------------|----------------------------------------------------------------|
| <i>interval_ms</i> | Specifies the desired LED blink interval time in milliseconds. |
|--------------------|----------------------------------------------------------------|

**Returns**

none.

**7.712.2.3 LedBlinkTask()**

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

**Returns**

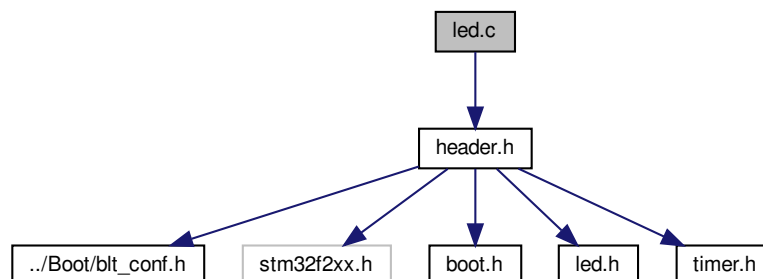
none.

**7.713 led.c File Reference**

LED driver source file.

```
#include "header.h"
```

Include dependency graph for ARMCM3\_STM32F2\_Olimex\_STM32P207\_GCC/Prog/led.c:

**Macros**

- `#define LED_TOGGLE_MS (500)`  
*Toggle interval time in milliseconds.*

**Functions**

- void `LedInit` (void)  
*Initializes the LED.*
- void `LedToggle` (void)  
*Toggles the LED at a fixed time interval.*

### 7.713.1 Detailed Description

LED driver source file.

### 7.713.2 Function Documentation

#### 7.713.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

##### Returns

none.

#### 7.713.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

##### Returns

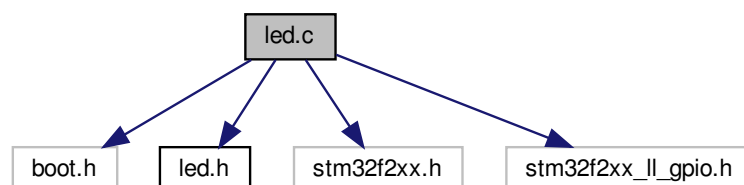
none.

## 7.714 led.c File Reference

LED driver source file.

```
#include "boot.h"
#include "led.h"
#include "stm32f2xx.h"
#include "stm32f2xx_ll_gpio.h"
```

Include dependency graph for ARMC3M3\_STM32F2\_Olimex\_STM32P207\_IAR/Boot/led.c:



## Functions

- void `LedBlinkInit` (`blt_int16u` interval\_ms)  
*Initializes the LED blink driver.*
- void `LedBlinkTask` (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void `LedBlinkExit` (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

## Variables

- static `blt_int16u` `ledBlinkIntervalMs`  
*Holds the desired LED blink interval time.*

### 7.714.1 Detailed Description

LED driver source file.

### 7.714.2 Function Documentation

#### 7.714.2.1 `LedBlinkExit()`

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

#### Returns

none.

#### 7.714.2.2 `LedBlinkInit()`

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

#### Parameters

|                          |                                                                |
|--------------------------|----------------------------------------------------------------|
| <code>interval_ms</code> | Specifies the desired LED blink interval time in milliseconds. |
|--------------------------|----------------------------------------------------------------|

**Returns**

none.

**7.714.2.3 LedBlinkTask()**

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

**Returns**

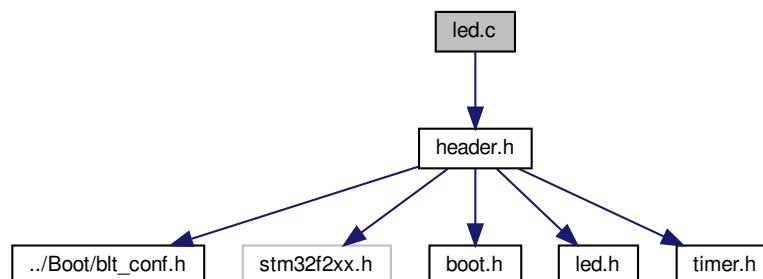
none.

**7.715 led.c File Reference**

LED driver source file.

```
#include "header.h"
```

Include dependency graph for ARMCM3\_STM32F2\_Olimex\_STM32P207\_IAR/Prog/led.c:

**Macros**

- `#define LED_TOGGLE_MS (500)`  
*Toggle interval time in milliseconds.*

**Functions**

- void `LedInit` (void)  
*Initializes the LED.*
- void `LedToggle` (void)  
*Toggles the LED at a fixed time interval.*

### 7.715.1 Detailed Description

LED driver source file.

### 7.715.2 Function Documentation

#### 7.715.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

##### Returns

none.

#### 7.715.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

##### Returns

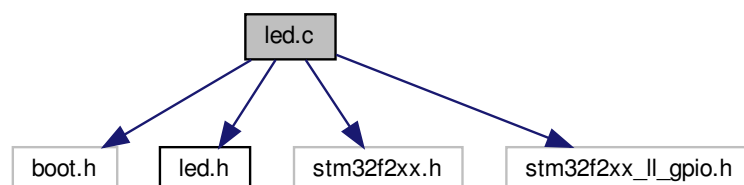
none.

## 7.716 led.c File Reference

LED driver source file.

```
#include "boot.h"
#include "led.h"
#include "stm32f2xx.h"
#include "stm32f2xx_ll_gpio.h"
```

Include dependency graph for ARMC3M3\_STM32F2\_Olimex\_STM32P207\_Keil/Boot/led.c:



## Functions

- void [LedBlinkInit](#) ([blt\\_int16u](#) interval\_ms)  
*Initializes the LED blink driver.*
- void [LedBlinkTask](#) (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void [LedBlinkExit](#) (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

## Variables

- static [blt\\_int16u](#) [ledBlinkIntervalMs](#)  
*Holds the desired LED blink interval time.*

### 7.716.1 Detailed Description

LED driver source file.

### 7.716.2 Function Documentation

#### 7.716.2.1 LedBlinkExit()

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

#### Returns

none.

#### 7.716.2.2 LedBlinkInit()

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

#### Parameters

|                    |                                                                |
|--------------------|----------------------------------------------------------------|
| <i>interval_ms</i> | Specifies the desired LED blink interval time in milliseconds. |
|--------------------|----------------------------------------------------------------|

**Returns**

none.

**7.716.2.3 LedBlinkTask()**

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

**Returns**

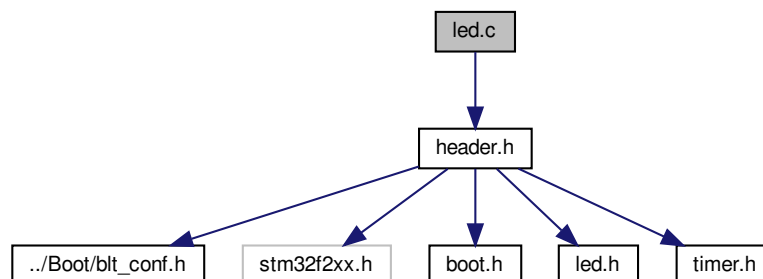
none.

**7.717 led.c File Reference**

LED driver source file.

```
#include "header.h"
```

Include dependency graph for ARMCM3\_STM32F2\_Olimex\_STM32P207\_Keil/Prog/led.c:

**Macros**

- `#define LED_TOGGLE_MS (500)`  
*Toggle interval time in milliseconds.*

**Functions**

- void `LedInit` (void)  
*Initializes the LED.*
- void `LedToggle` (void)  
*Toggles the LED at a fixed time interval.*



### 7.717.1 Detailed Description

LED driver source file.

### 7.717.2 Function Documentation

#### 7.717.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

##### Returns

none.

#### 7.717.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

##### Returns

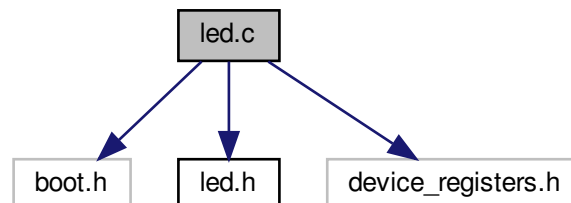
none.

## 7.718 led.c File Reference

LED driver source file.

```
#include "boot.h"
#include "led.h"
#include "device_registers.h"
```

Include dependency graph for ARMCM4\_S32K14\_S32K144EVB\_GCC/Boot/led.c:



## Functions

- void `LedBlinkInit` (`blt_int16u` interval\_ms)  
*Initializes the LED blink driver.*
- void `LedBlinkTask` (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void `LedBlinkExit` (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

## Variables

- static `blt_int16u` `ledBlinkIntervalMs`  
*Holds the desired LED blink interval time.*

### 7.718.1 Detailed Description

LED driver source file.

### 7.718.2 Function Documentation

#### 7.718.2.1 LedBlinkExit()

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

#### Returns

none.

#### 7.718.2.2 LedBlinkInit()

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

#### Parameters

|                          |                                                                |
|--------------------------|----------------------------------------------------------------|
| <code>interval_ms</code> | Specifies the desired LED blink interval time in milliseconds. |
|--------------------------|----------------------------------------------------------------|

**Returns**

none.

**7.718.2.3 LedBlinkTask()**

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

**Returns**

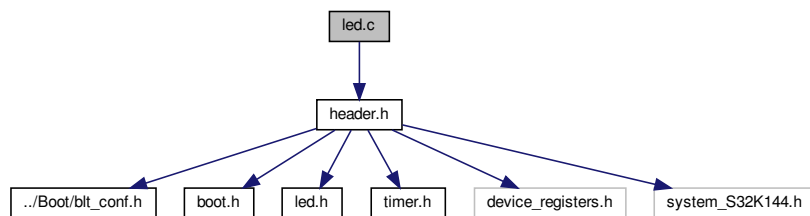
none.

**7.719 led.c File Reference**

LED driver source file.

```
#include "header.h"
```

Include dependency graph for ARMCM4\_S32K14\_S32K144EVB\_GCC/Prog/led.c:

**Macros**

- `#define LED_TOGGLE_MS (500U)`  
*Toggle interval time in milliseconds.*

**Functions**

- `void LedInit (void)`  
*Initializes the LED.*
- `void LedToggle (void)`  
*Toggles the LED at a fixed time interval.*

### 7.719.1 Detailed Description

LED driver source file.

### 7.719.2 Function Documentation

#### 7.719.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

##### Returns

none.

#### 7.719.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

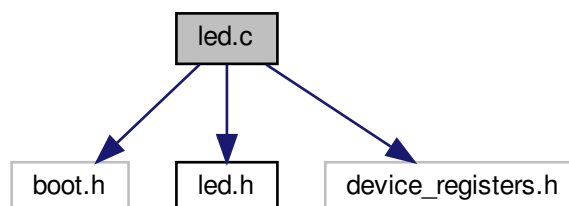
##### Returns

none.

## 7.720 led.c File Reference

LED driver source file.

```
#include "boot.h"
#include "led.h"
#include "device_registers.h"
Include dependency graph for ARMCM4_S32K14_S32K144EVB_IAR/Boot/led.c:
```



## Functions

- void [LedBlinkInit](#) ([blt\\_int16u](#) interval\_ms)  
*Initializes the LED blink driver.*
- void [LedBlinkTask](#) (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void [LedBlinkExit](#) (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

## Variables

- static [blt\\_int16u](#) [ledBlinkIntervalMs](#)  
*Holds the desired LED blink interval time.*

### 7.720.1 Detailed Description

LED driver source file.

### 7.720.2 Function Documentation

#### 7.720.2.1 LedBlinkExit()

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

#### Returns

none.

#### 7.720.2.2 LedBlinkInit()

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

#### Parameters

|                    |                                                                |
|--------------------|----------------------------------------------------------------|
| <i>interval_ms</i> | Specifies the desired LED blink interval time in milliseconds. |
|--------------------|----------------------------------------------------------------|

**Returns**

none.

**7.720.2.3 LedBlinkTask()**

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

**Returns**

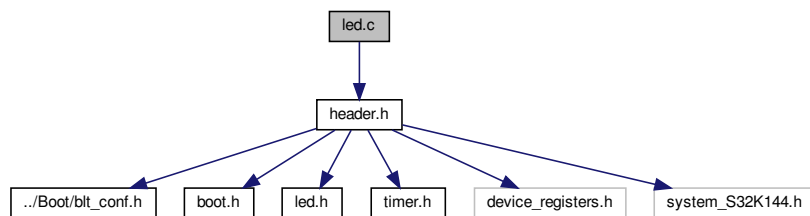
none.

**7.721 led.c File Reference**

LED driver source file.

```
#include "header.h"
```

Include dependency graph for ARMCM4\_S32K14\_S32K144EVB\_IAR/Prog/led.c:

**Macros**

- `#define LED_TOGGLE_MS (500U)`  
*Toggle interval time in milliseconds.*

**Functions**

- `void LedInit (void)`  
*Initializes the LED.*
- `void LedToggle (void)`  
*Toggles the LED at a fixed time interval.*

### 7.721.1 Detailed Description

LED driver source file.

### 7.721.2 Function Documentation

#### 7.721.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

**Returns**

none.

#### 7.721.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

**Returns**

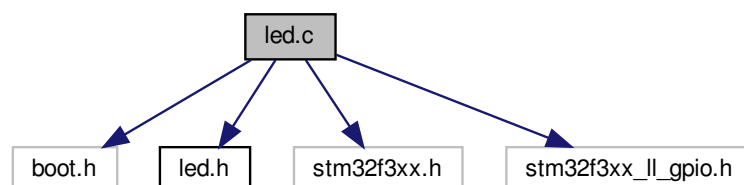
none.

## 7.722 led.c File Reference

LED driver source file.

```
#include "boot.h"
#include "led.h"
#include "stm32f3xx.h"
#include "stm32f3xx_ll_gpio.h"
```

Include dependency graph for ARMC4\_STM32F3\_Discovery\_F303VC\_CubeIDE/Boot/App/led.c:



## Functions

- void `LedBlinkInit` (`blt_int16u` interval\_ms)  
*Initializes the LED blink driver.*
- void `LedBlinkTask` (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void `LedBlinkExit` (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

## Variables

- static `blt_int16u` `ledBlinkIntervalMs`  
*Holds the desired LED blink interval time.*

### 7.722.1 Detailed Description

LED driver source file.

### 7.722.2 Function Documentation

#### 7.722.2.1 `LedBlinkExit()`

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

#### Returns

none.

#### 7.722.2.2 `LedBlinkInit()`

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

#### Parameters

|                          |                                                                |
|--------------------------|----------------------------------------------------------------|
| <code>interval_ms</code> | Specifies the desired LED blink interval time in milliseconds. |
|--------------------------|----------------------------------------------------------------|



**Returns**

none.

**7.722.2.3 LedBlinkTask()**

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

**Returns**

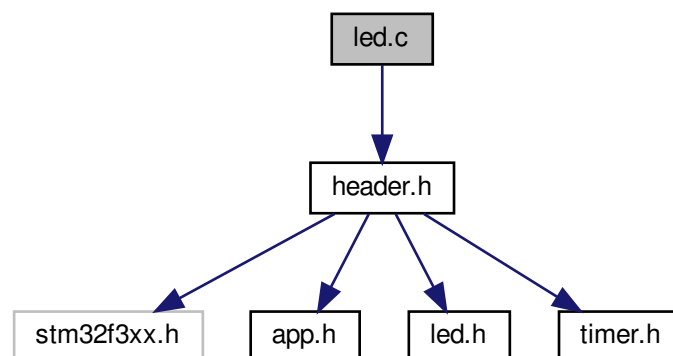
none.

**7.723 led.c File Reference**

LED driver source file.

```
#include "header.h"
```

Include dependency graph for ARMCM4\_STM32F3\_Discovery\_F303VC\_CubeIDE/Prog/App/led.c:

**Macros**

- `#define LED_TOGGLE_MS (500)`  
*Toggle interval time in milliseconds.*

## Functions

- void [LedInit](#) (void)  
*Initializes the LED.*
- void [LedToggle](#) (void)  
*Toggles the LED at a fixed time interval.*

### 7.723.1 Detailed Description

LED driver source file.

### 7.723.2 Function Documentation

#### 7.723.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

#### Returns

none.

#### 7.723.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

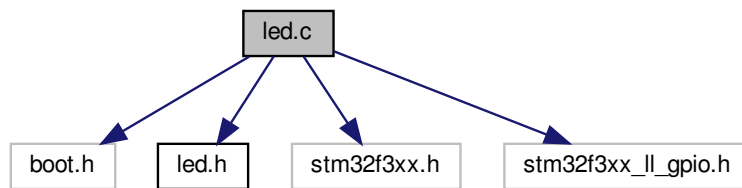
#### Returns

none.

## 7.724 led.c File Reference

LED driver source file.

```
#include "boot.h"
#include "led.h"
#include "stm32f3xx.h"
#include "stm32f3xx_ll_gpio.h"
Include dependency graph for ARMCM4_STM32F3_Discovery_F303VC_GCC/Boot/led.c:
```



### Functions

- void [LedBlinkInit](#) ([blt\\_int16u](#) interval\_ms)  
*Initializes the LED blink driver.*
- void [LedBlinkTask](#) (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void [LedBlinkExit](#) (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

### Variables

- static [blt\\_int16u](#) [ledBlinkIntervalMs](#)  
*Holds the desired LED blink interval time.*

#### 7.724.1 Detailed Description

LED driver source file.

#### 7.724.2 Function Documentation

### 7.724.2.1 LedBlinkExit()

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

#### Returns

none.

### 7.724.2.2 LedBlinkInit()

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

#### Parameters

|                    |                                                                |
|--------------------|----------------------------------------------------------------|
| <i>interval_ms</i> | Specifies the desired LED blink interval time in milliseconds. |
|--------------------|----------------------------------------------------------------|

#### Returns

none.

### 7.724.2.3 LedBlinkTask()

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

#### Returns

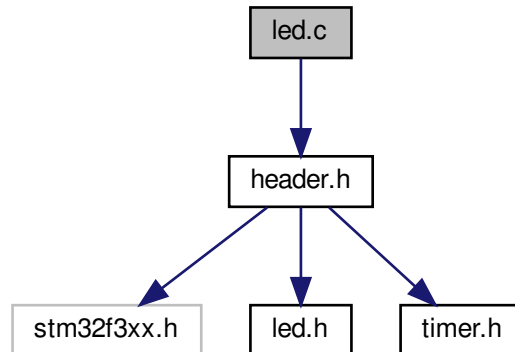
none.

## 7.725 led.c File Reference

LED driver source file.

```
#include "header.h"
```

Include dependency graph for ARMCM4\_STM32F3\_Discovery\_F303VC\_GCC/Prog/led.c:



## Macros

- `#define LED_TOGGLE_MS (500)`  
*Toggle interval time in milliseconds.*

## Functions

- `void LedInit (void)`  
*Initializes the LED.*
- `void LedToggle (void)`  
*Toggles the LED at a fixed time interval.*

### 7.725.1 Detailed Description

LED driver source file.

### 7.725.2 Function Documentation

#### 7.725.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

#### Returns

none.

### 7.725.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

#### Returns

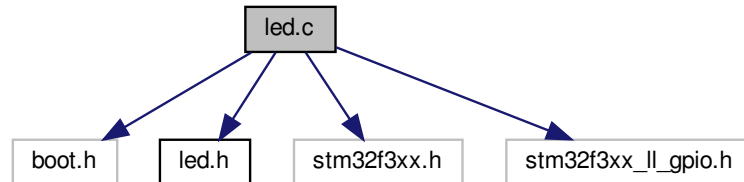
none.

## 7.726 led.c File Reference

LED driver source file.

```
#include "boot.h"
#include "led.h"
#include "stm32f3xx.h"
#include "stm32f3xx_ll_gpio.h"
```

Include dependency graph for ARMCM4\_STM32F3\_Discovery\_F303VC\_IAR/Boot/led.c:



## Functions

- void [LedBlinkInit](#) ([blt\\_int16u](#) interval\_ms)  
*Initializes the LED blink driver.*
- void [LedBlinkTask](#) (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void [LedBlinkExit](#) (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

## Variables

- static [blt\\_int16u](#) [ledBlinkIntervalMs](#)  
*Holds the desired LED blink interval time.*

## 7.726.1 Detailed Description

LED driver source file.

## 7.726.2 Function Documentation

### 7.726.2.1 LedBlinkExit()

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

#### Returns

none.

### 7.726.2.2 LedBlinkInit()

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

#### Parameters

|                    |                                                                |
|--------------------|----------------------------------------------------------------|
| <i>interval_ms</i> | Specifies the desired LED blink interval time in milliseconds. |
|--------------------|----------------------------------------------------------------|

#### Returns

none.

### 7.726.2.3 LedBlinkTask()

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

#### Returns

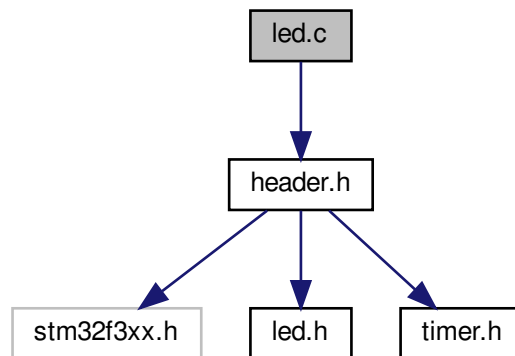
none.

## 7.727 led.c File Reference

LED driver source file.

```
#include "header.h"
```

Include dependency graph for ARMCM4\_STM32F3\_Discovery\_F303VC\_IAR/Prog/led.c:



### Macros

- `#define LED_TOGGLE_MS (500)`  
*Toggle interval time in milliseconds.*

### Functions

- `void LedInit (void)`  
*Initializes the LED.*
- `void LedToggle (void)`  
*Toggles the LED at a fixed time interval.*

### 7.727.1 Detailed Description

LED driver source file.

### 7.727.2 Function Documentation



### 7.727.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

#### Returns

none.

### 7.727.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

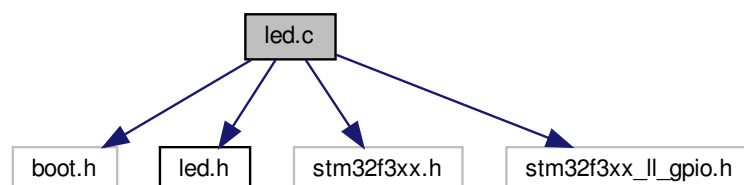
#### Returns

none.

## 7.728 led.c File Reference

LED driver source file.

```
#include "boot.h"
#include "led.h"
#include "stm32f3xx.h"
#include "stm32f3xx_ll_gpio.h"
Include dependency graph for ARMCM4_STM32F3_Discovery_F303VC_Keil/Boot/led.c:
```



## Functions

- void `LedBlinkInit` (`blt_int16u` interval\_ms)  
*Initializes the LED blink driver.*
- void `LedBlinkTask` (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void `LedBlinkExit` (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

## Variables

- static `blt_int16u` `ledBlinkIntervalMs`  
*Holds the desired LED blink interval time.*

### 7.728.1 Detailed Description

LED driver source file.

### 7.728.2 Function Documentation

#### 7.728.2.1 `LedBlinkExit()`

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

#### Returns

none.

#### 7.728.2.2 `LedBlinkInit()`

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

#### Parameters

|                          |                                                                |
|--------------------------|----------------------------------------------------------------|
| <code>interval_ms</code> | Specifies the desired LED blink interval time in milliseconds. |
|--------------------------|----------------------------------------------------------------|

**Returns**

none.

**7.728.2.3 LedBlinkTask()**

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

**Returns**

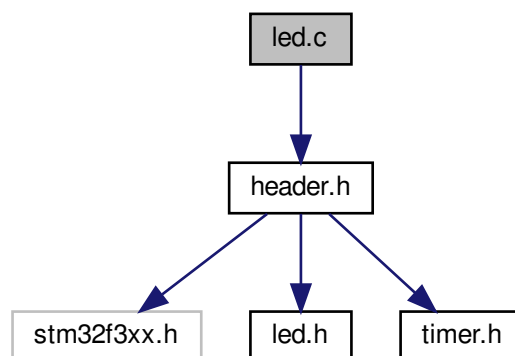
none.

**7.729 led.c File Reference**

LED driver source file.

```
#include "header.h"
```

Include dependency graph for ARMCM4\_STM32F3\_Discovery\_F303VC\_Keil/Prog/led.c:

**Macros**

- #define **LED\_TOGGLE\_MS** (500)  
*Toggle interval time in milliseconds.*

## Functions

- void [LedInit](#) (void)  
*Initializes the LED.*
- void [LedToggle](#) (void)  
*Toggles the LED at a fixed time interval.*

### 7.729.1 Detailed Description

LED driver source file.

### 7.729.2 Function Documentation

#### 7.729.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

#### Returns

none.

#### 7.729.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

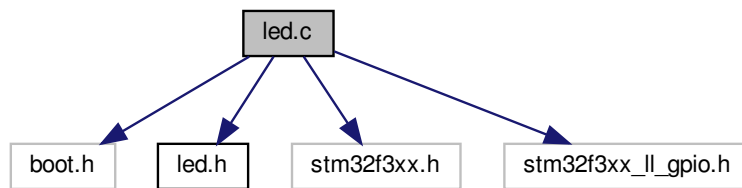
#### Returns

none.

## 7.730 led.c File Reference

LED driver source file.

```
#include "boot.h"
#include "led.h"
#include "stm32f3xx.h"
#include "stm32f3xx_ll_gpio.h"
Include dependency graph for ARMCM4_STM32F3_Nucleo_F303K8_CubeIDE/Boot/App/led.c:
```



### Functions

- void `LedBlinkInit` (`blt_int16u` interval\_ms)  
*Initializes the LED blink driver.*
- void `LedBlinkTask` (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void `LedBlinkExit` (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

### Variables

- static `blt_int16u` `ledBlinkIntervalMs`  
*Holds the desired LED blink interval time.*

#### 7.730.1 Detailed Description

LED driver source file.

#### 7.730.2 Function Documentation

### 7.730.2.1 LedBlinkExit()

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

#### Returns

none.

### 7.730.2.2 LedBlinkInit()

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

#### Parameters

|                    |                                                                |
|--------------------|----------------------------------------------------------------|
| <i>interval_ms</i> | Specifies the desired LED blink interval time in milliseconds. |
|--------------------|----------------------------------------------------------------|

#### Returns

none.

### 7.730.2.3 LedBlinkTask()

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

#### Returns

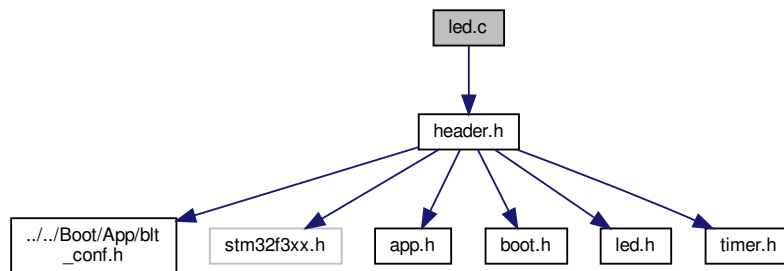
none.

## 7.731 led.c File Reference

LED driver source file.

```
#include "header.h"
```

Include dependency graph for ARMCM4\_STM32F3\_Nucleo\_F303K8\_CubeIDE/Prog/App/led.c:



## Macros

- `#define LED_TOGGLE_MS (500)`  
*Toggle interval time in milliseconds.*

## Functions

- `void LedInit (void)`  
*Initializes the LED.*
- `void LedToggle (void)`  
*Toggles the LED at a fixed time interval.*

### 7.731.1 Detailed Description

LED driver source file.

### 7.731.2 Function Documentation

#### 7.731.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

#### Returns

none.

### 7.731.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

#### Returns

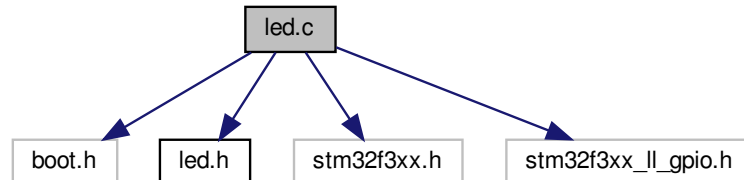
none.

## 7.732 led.c File Reference

LED driver source file.

```
#include "boot.h"
#include "led.h"
#include "stm32f3xx.h"
#include "stm32f3xx_ll_gpio.h"
```

Include dependency graph for ARMCM4\_STM32F3\_Nucleo\_F303K8\_GCC/Boot/led.c:



## Functions

- void [LedBlinkInit](#) ([blt\\_int16u](#) interval\_ms)  
*Initializes the LED blink driver.*
- void [LedBlinkTask](#) (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void [LedBlinkExit](#) (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

## Variables

- static [blt\\_int16u](#) [ledBlinkIntervalMs](#)  
*Holds the desired LED blink interval time.*



## 7.732.1 Detailed Description

LED driver source file.

## 7.732.2 Function Documentation

### 7.732.2.1 LedBlinkExit()

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

#### Returns

none.

### 7.732.2.2 LedBlinkInit()

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

#### Parameters

|                    |                                                                |
|--------------------|----------------------------------------------------------------|
| <i>interval_ms</i> | Specifies the desired LED blink interval time in milliseconds. |
|--------------------|----------------------------------------------------------------|

#### Returns

none.

### 7.732.2.3 LedBlinkTask()

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

#### Returns

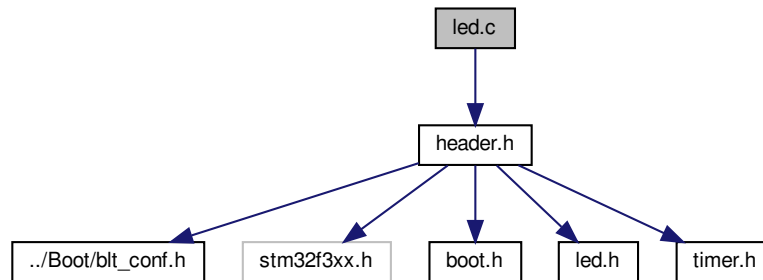
none.

## 7.733 led.c File Reference

LED driver source file.

```
#include "header.h"
```

Include dependency graph for ARMCM4\_STM32F3\_Nucleo\_F303K8\_GCC/Prog/led.c:



### Macros

- `#define LED_TOGGLE_MS (500)`  
*Toggle interval time in milliseconds.*

### Functions

- `void LedInit (void)`  
*Initializes the LED.*
- `void LedToggle (void)`  
*Toggles the LED at a fixed time interval.*

#### 7.733.1 Detailed Description

LED driver source file.

#### 7.733.2 Function Documentation

##### 7.733.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

#### Returns

none.

### 7.733.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

#### Returns

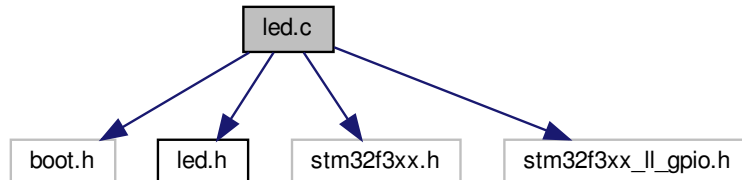
none.

## 7.734 led.c File Reference

LED driver source file.

```
#include "boot.h"
#include "led.h"
#include "stm32f3xx.h"
#include "stm32f3xx_ll_gpio.h"
```

Include dependency graph for ARMCM4\_STM32F3\_Nucleo\_F303K8\_IAR/Boot/led.c:



## Functions

- void [LedBlinkInit](#) ([blt\\_int16u](#) interval\_ms)  
*Initializes the LED blink driver.*
- void [LedBlinkTask](#) (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void [LedBlinkExit](#) (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

## Variables

- static [blt\\_int16u](#) [ledBlinkIntervalMs](#)  
*Holds the desired LED blink interval time.*

### 7.734.1 Detailed Description

LED driver source file.

### 7.734.2 Function Documentation

#### 7.734.2.1 LedBlinkExit()

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

##### Returns

none.

#### 7.734.2.2 LedBlinkInit()

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

##### Parameters

|                    |                                                                |
|--------------------|----------------------------------------------------------------|
| <i>interval_ms</i> | Specifies the desired LED blink interval time in milliseconds. |
|--------------------|----------------------------------------------------------------|

##### Returns

none.

#### 7.734.2.3 LedBlinkTask()

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

##### Returns

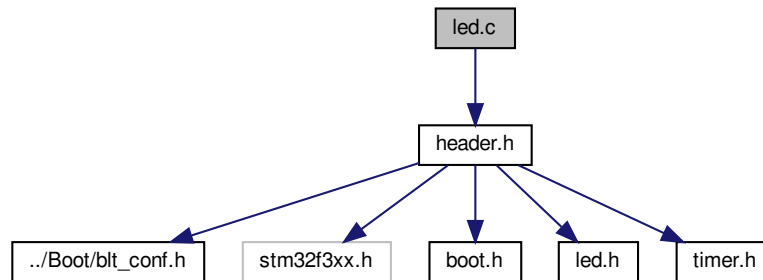
none.

## 7.735 led.c File Reference

LED driver source file.

```
#include "header.h"
```

Include dependency graph for ARMCM4\_STM32F3\_Nucleo\_F303K8\_IAR/Prog/led.c:



### Macros

- `#define LED_TOGGLE_MS (500)`  
*Toggle interval time in milliseconds.*

### Functions

- `void LedInit (void)`  
*Initializes the LED.*
- `void LedToggle (void)`  
*Toggles the LED at a fixed time interval.*

#### 7.735.1 Detailed Description

LED driver source file.

#### 7.735.2 Function Documentation

##### 7.735.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

Returns

none.

### 7.735.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

#### Returns

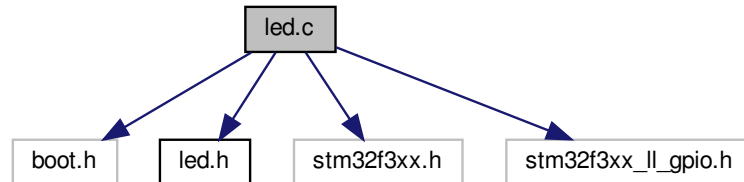
none.

## 7.736 led.c File Reference

LED driver source file.

```
#include "boot.h"
#include "led.h"
#include "stm32f3xx.h"
#include "stm32f3xx_ll_gpio.h"
```

Include dependency graph for ARMCM4\_STM32F3\_Nucleo\_F303K8\_Keil/Boot/led.c:



## Functions

- void [LedBlinkInit](#) ([blt\\_int16u](#) interval\_ms)  
*Initializes the LED blink driver.*
- void [LedBlinkTask](#) (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void [LedBlinkExit](#) (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

## Variables

- static [blt\\_int16u](#) [ledBlinkIntervalMs](#)  
*Holds the desired LED blink interval time.*

## 7.736.1 Detailed Description

LED driver source file.

## 7.736.2 Function Documentation

### 7.736.2.1 LedBlinkExit()

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

#### Returns

none.

### 7.736.2.2 LedBlinkInit()

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

#### Parameters

|                    |                                                                |
|--------------------|----------------------------------------------------------------|
| <i>interval_ms</i> | Specifies the desired LED blink interval time in milliseconds. |
|--------------------|----------------------------------------------------------------|

#### Returns

none.

### 7.736.2.3 LedBlinkTask()

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

#### Returns

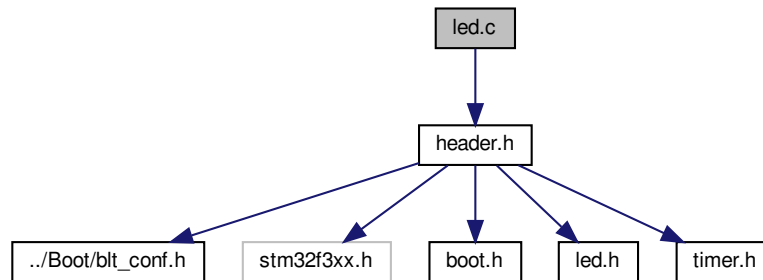
none.

## 7.737 led.c File Reference

LED driver source file.

```
#include "header.h"
```

Include dependency graph for ARMCM4\_STM32F3\_Nucleo\_F303K8\_Keil/Prog/led.c:



### Macros

- `#define LED_TOGGLE_MS (500)`  
*Toggle interval time in milliseconds.*

### Functions

- `void LedInit (void)`  
*Initializes the LED.*
- `void LedToggle (void)`  
*Toggles the LED at a fixed time interval.*

### 7.737.1 Detailed Description

LED driver source file.

### 7.737.2 Function Documentation

#### 7.737.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

#### Returns

none.



### 7.737.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

#### Returns

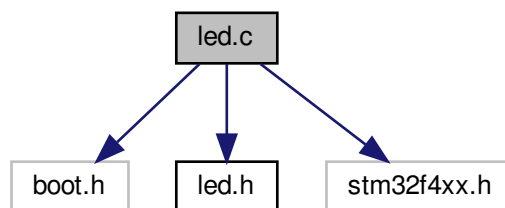
none.

## 7.738 led.c File Reference

LED driver source file.

```
#include "boot.h"
#include "led.h"
#include "stm32f4xx.h"
```

Include dependency graph for ARMCM4\_STM32F4\_Nucleo\_F429ZI\_CubeIDE/Boot/App/led.c:



## Functions

- void `LedBlinkInit` (`blt_int16u` interval\_ms)  
*Initializes the LED blink driver.*
- void `LedBlinkTask` (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void `LedBlinkExit` (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

## Variables

- static `blt_int16u` `ledBlinkIntervalMs`  
*Holds the desired LED blink interval time.*

### 7.738.1 Detailed Description

LED driver source file.

### 7.738.2 Function Documentation

#### 7.738.2.1 LedBlinkExit()

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

##### Returns

none.

#### 7.738.2.2 LedBlinkInit()

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

##### Parameters

|                    |                                                                |
|--------------------|----------------------------------------------------------------|
| <i>interval_ms</i> | Specifies the desired LED blink interval time in milliseconds. |
|--------------------|----------------------------------------------------------------|

##### Returns

none.

#### 7.738.2.3 LedBlinkTask()

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

##### Returns

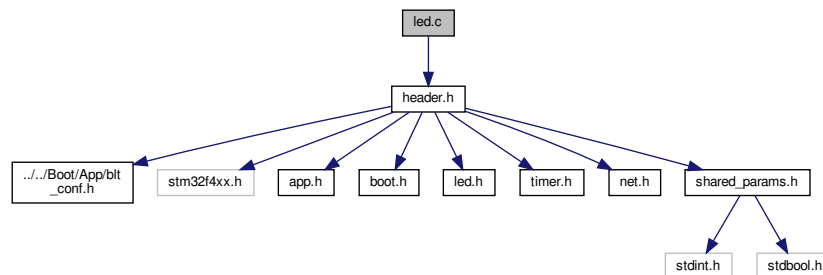
none.

## 7.739 led.c File Reference

LED driver source file.

```
#include "header.h"
```

Include dependency graph for ARMCM4\_STM32F4\_Nucleo\_F429ZI\_CubeIDE/Prog/App/led.c:



### Macros

- `#define LED_TOGGLE_MS (500)`  
*Toggle interval time in milliseconds.*

### Functions

- `void LedInit (void)`  
*Initializes the LED.*
- `void LedToggle (void)`  
*Toggles the LED at a fixed time interval.*

### 7.739.1 Detailed Description

LED driver source file.

### 7.739.2 Function Documentation

#### 7.739.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

Returns

none.

### 7.739.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

#### Returns

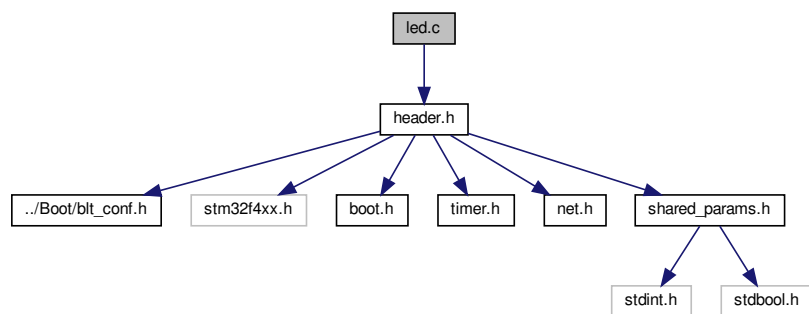
none.

## 7.740 led.c File Reference

LED driver source file.

```
#include "header.h"
```

Include dependency graph for ARMC4\_STM32F4\_Nucleo\_F429ZI\_GCC/Prog/led.c:



## Macros

- `#define LED_TOGGLE_MS (500)`  
*Toggle interval time in milliseconds.*

## Functions

- `void LedInit (void)`  
*Initializes the LED.*
- `void LedToggle (void)`  
*Toggles the LED at a fixed time interval.*

### 7.740.1 Detailed Description

LED driver source file.

## 7.740.2 Function Documentation

### 7.740.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

#### Returns

none.

### 7.740.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

#### Returns

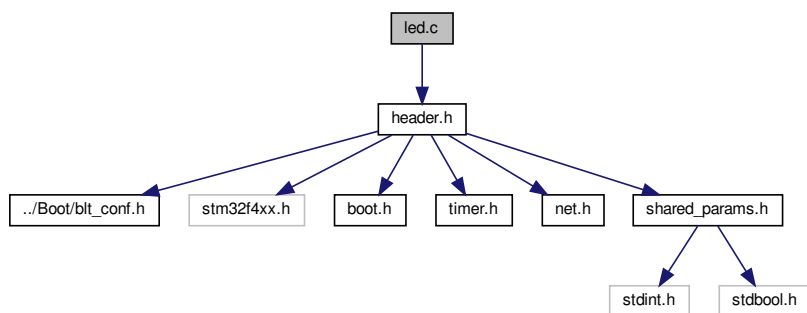
none.

## 7.741 led.c File Reference

LED driver source file.

```
#include "header.h"
```

Include dependency graph for ARMCM4\_STM32F4\_Nucleo\_F429ZI\_IAR/Prog/led.c:



## Macros

- `#define LED_TOGGLE_MS (500)`  
*Toggle interval time in milliseconds.*

## Functions

- `void LedInit (void)`  
*Initializes the LED.*
- `void LedToggle (void)`  
*Toggles the LED at a fixed time interval.*

### 7.741.1 Detailed Description

LED driver source file.

### 7.741.2 Function Documentation

#### 7.741.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

#### Returns

none.

#### 7.741.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

#### Returns

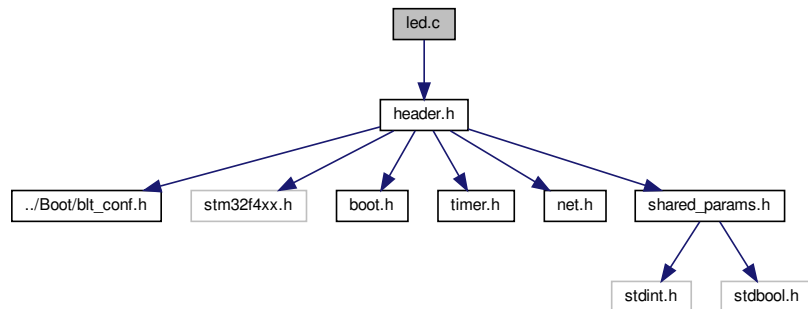
none.

## 7.742 led.c File Reference

LED driver source file.

```
#include "header.h"
```

Include dependency graph for ARMCM4\_STM32F4\_Nucleo\_F429ZI\_Keil/Prog/led.c:



### Macros

- `#define LED_TOGGLE_MS (500)`  
*Toggle interval time in milliseconds.*

### Functions

- void `LedInit` (void)  
*Initializes the LED.*
- void `LedToggle` (void)  
*Toggles the LED at a fixed time interval.*

### 7.742.1 Detailed Description

LED driver source file.

### 7.742.2 Function Documentation

#### 7.742.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

Returns

none.

### 7.742.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

#### Returns

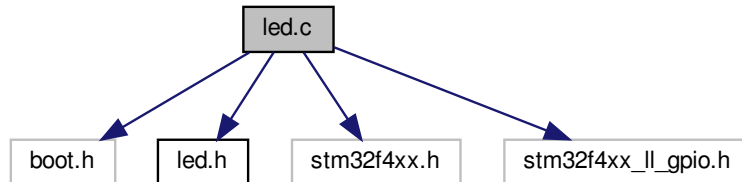
none.

## 7.743 led.c File Reference

LED driver source file.

```
#include "boot.h"
#include "led.h"
#include "stm32f4xx.h"
#include "stm32f4xx_ll_gpio.h"
```

Include dependency graph for ARMCM4\_STM32F4\_Olimex\_STM32P405\_CubeIDE/Boot/App/led.c:



## Functions

- void [LedBlinkInit](#) ([blt\\_int16u](#) interval\_ms)  
*Initializes the LED blink driver.*
- void [LedBlinkTask](#) (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void [LedBlinkExit](#) (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

## Variables

- static [blt\\_int16u](#) [ledBlinkIntervalMs](#)  
*Holds the desired LED blink interval time.*



## 7.743.1 Detailed Description

LED driver source file.

## 7.743.2 Function Documentation

### 7.743.2.1 LedBlinkExit()

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

#### Returns

none.

### 7.743.2.2 LedBlinkInit()

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

#### Parameters

|                    |                                                                |
|--------------------|----------------------------------------------------------------|
| <i>interval_ms</i> | Specifies the desired LED blink interval time in milliseconds. |
|--------------------|----------------------------------------------------------------|

#### Returns

none.

### 7.743.2.3 LedBlinkTask()

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

#### Returns

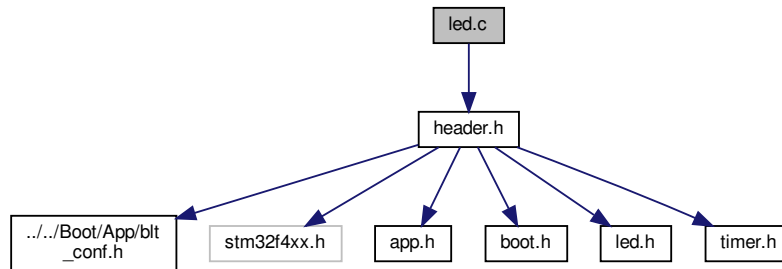
none.

## 7.744 led.c File Reference

LED driver source file.

```
#include "header.h"
```

Include dependency graph for ARMCM4\_STM32F4\_Olimex\_STM32P405\_CubeIDE/Prog/App/led.c:



### Macros

- `#define LED_TOGGLE_MS (500)`  
*Toggle interval time in milliseconds.*

### Functions

- `void LedInit (void)`  
*Initializes the LED.*
- `void LedToggle (void)`  
*Toggles the LED at a fixed time interval.*

### 7.744.1 Detailed Description

LED driver source file.

### 7.744.2 Function Documentation

#### 7.744.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

#### Returns

none.

### 7.744.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

#### Returns

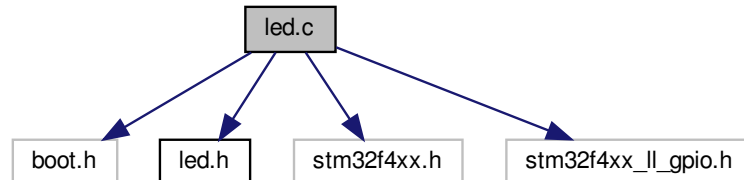
none.

## 7.745 led.c File Reference

LED driver source file.

```
#include "boot.h"
#include "led.h"
#include "stm32f4xx.h"
#include "stm32f4xx_ll_gpio.h"
```

Include dependency graph for ARMCM4\_STM32F4\_Olimex\_STM32P405\_GCC/Boot/led.c:



## Functions

- void [LedBlinkInit](#) ([blt\\_int16u](#) interval\_ms)  
*Initializes the LED blink driver.*
- void [LedBlinkTask](#) (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void [LedBlinkExit](#) (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

## Variables

- static [blt\\_int16u](#) [ledBlinkIntervalMs](#)  
*Holds the desired LED blink interval time.*

### 7.745.1 Detailed Description

LED driver source file.

### 7.745.2 Function Documentation

#### 7.745.2.1 LedBlinkExit()

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

##### Returns

none.

#### 7.745.2.2 LedBlinkInit()

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

##### Parameters

|                    |                                                                |
|--------------------|----------------------------------------------------------------|
| <i>interval_ms</i> | Specifies the desired LED blink interval time in milliseconds. |
|--------------------|----------------------------------------------------------------|

##### Returns

none.

#### 7.745.2.3 LedBlinkTask()

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

##### Returns

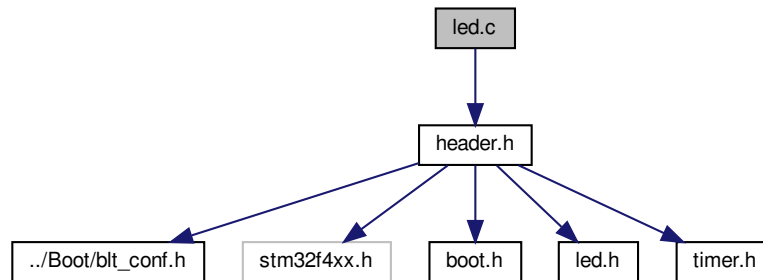
none.

## 7.746 led.c File Reference

LED driver source file.

```
#include "header.h"
```

Include dependency graph for ARMCM4\_STM32F4\_Olimex\_STM32P405\_GCC/Prog/led.c:



### Macros

- `#define LED_TOGGLE_MS (500)`  
*Toggle interval time in milliseconds.*

### Functions

- `void LedInit (void)`  
*Initializes the LED.*
- `void LedToggle (void)`  
*Toggles the LED at a fixed time interval.*

#### 7.746.1 Detailed Description

LED driver source file.

#### 7.746.2 Function Documentation

##### 7.746.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

#### Returns

none.

### 7.746.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

#### Returns

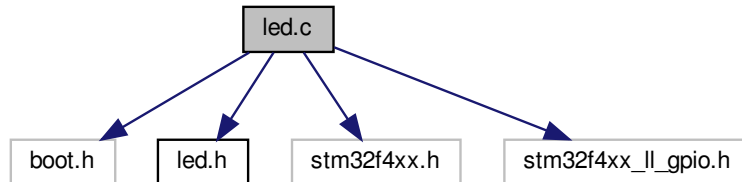
none.

## 7.747 led.c File Reference

LED driver source file.

```
#include "boot.h"
#include "led.h"
#include "stm32f4xx.h"
#include "stm32f4xx_ll_gpio.h"
```

Include dependency graph for ARMCM4\_STM32F4\_Olimex\_STM32P405\_IAR/Boot/led.c:



## Functions

- void [LedBlinkInit](#) ([blt\\_int16u](#) interval\_ms)  
*Initializes the LED blink driver.*
- void [LedBlinkTask](#) (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void [LedBlinkExit](#) (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

## Variables

- static [blt\\_int16u](#) [ledBlinkIntervalMs](#)  
*Holds the desired LED blink interval time.*

## 7.747.1 Detailed Description

LED driver source file.

## 7.747.2 Function Documentation

### 7.747.2.1 LedBlinkExit()

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

#### Returns

none.

### 7.747.2.2 LedBlinkInit()

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

#### Parameters

|                    |                                                                |
|--------------------|----------------------------------------------------------------|
| <i>interval_ms</i> | Specifies the desired LED blink interval time in milliseconds. |
|--------------------|----------------------------------------------------------------|

#### Returns

none.

### 7.747.2.3 LedBlinkTask()

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

#### Returns

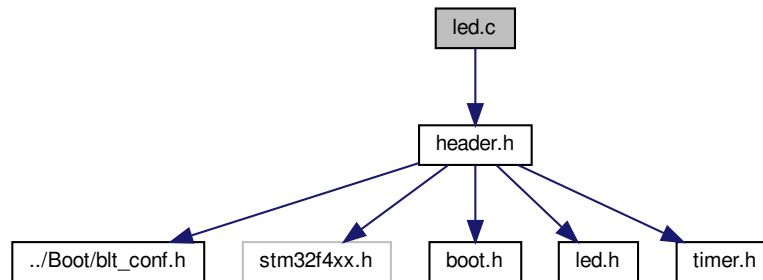
none.

## 7.748 led.c File Reference

LED driver source file.

```
#include "header.h"
```

Include dependency graph for ARMCM4\_STM32F4\_Olimex\_STM32P405\_IAR/Prog/led.c:



### Macros

- `#define LED_TOGGLE_MS (500)`  
*Toggle interval time in milliseconds.*

### Functions

- `void LedInit (void)`  
*Initializes the LED.*
- `void LedToggle (void)`  
*Toggles the LED at a fixed time interval.*

### 7.748.1 Detailed Description

LED driver source file.

### 7.748.2 Function Documentation

#### 7.748.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

#### Returns

none.



### 7.748.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

#### Returns

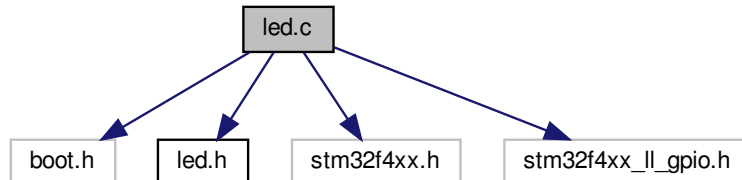
none.

## 7.749 led.c File Reference

LED driver source file.

```
#include "boot.h"
#include "led.h"
#include "stm32f4xx.h"
#include "stm32f4xx_ll_gpio.h"
```

Include dependency graph for ARMCM4\_STM32F4\_Olimex\_STM32P405\_Keil/Boot/led.c:



## Functions

- void `LedBlinkInit` (`blt_int16u` interval\_ms)  
*Initializes the LED blink driver.*
- void `LedBlinkTask` (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void `LedBlinkExit` (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

## Variables

- static `blt_int16u` `ledBlinkIntervalMs`  
*Holds the desired LED blink interval time.*

### 7.749.1 Detailed Description

LED driver source file.

### 7.749.2 Function Documentation

#### 7.749.2.1 LedBlinkExit()

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

##### Returns

none.

#### 7.749.2.2 LedBlinkInit()

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

##### Parameters

|                    |                                                                |
|--------------------|----------------------------------------------------------------|
| <i>interval_ms</i> | Specifies the desired LED blink interval time in milliseconds. |
|--------------------|----------------------------------------------------------------|

##### Returns

none.

#### 7.749.2.3 LedBlinkTask()

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

##### Returns

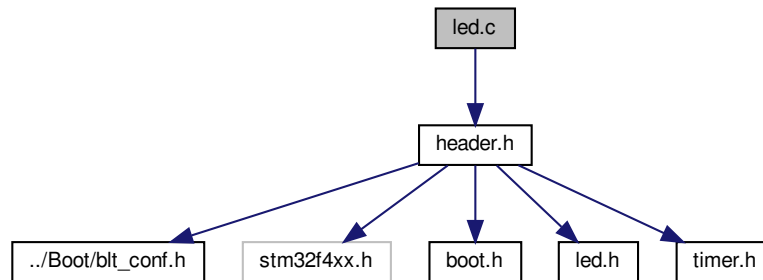
none.

## 7.750 led.c File Reference

LED driver source file.

```
#include "header.h"
```

Include dependency graph for ARMCM4\_STM32F4\_Olimex\_STM32P405\_Keil/Prog/led.c:



### Macros

- `#define LED_TOGGLE_MS (500)`  
*Toggle interval time in milliseconds.*

### Functions

- `void LedInit (void)`  
*Initializes the LED.*
- `void LedToggle (void)`  
*Toggles the LED at a fixed time interval.*

### 7.750.1 Detailed Description

LED driver source file.

### 7.750.2 Function Documentation

#### 7.750.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

Returns

none.

### 7.750.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

#### Returns

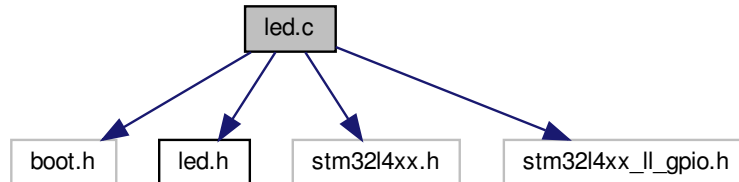
none.

## 7.751 led.c File Reference

LED driver source file.

```
#include "boot.h"
#include "led.h"
#include "stm32l4xx.h"
#include "stm32l4xx_ll_gpio.h"
```

Include dependency graph for ARMCM4\_STM32L4\_Nucleo\_L476RG\_CubeIDE/Boot/App/led.c:



## Functions

- void [LedBlinkInit](#) ([blt\\_int16u](#) interval\_ms)  
*Initializes the LED blink driver.*
- void [LedBlinkTask](#) (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void [LedBlinkExit](#) (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

## Variables

- static [blt\\_int16u](#) [ledBlinkIntervalMs](#)  
*Holds the desired LED blink interval time.*

## 7.751.1 Detailed Description

LED driver source file.

## 7.751.2 Function Documentation

### 7.751.2.1 LedBlinkExit()

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

#### Returns

none.

### 7.751.2.2 LedBlinkInit()

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

#### Parameters

|                    |                                                                |
|--------------------|----------------------------------------------------------------|
| <i>interval_ms</i> | Specifies the desired LED blink interval time in milliseconds. |
|--------------------|----------------------------------------------------------------|

#### Returns

none.

### 7.751.2.3 LedBlinkTask()

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

#### Returns

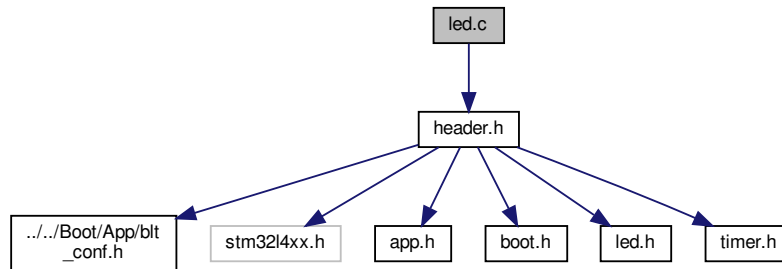
none.

## 7.752 led.c File Reference

LED driver source file.

```
#include "header.h"
```

Include dependency graph for ARMCM4\_STM32L4\_Nucleo\_L476RG\_CubeIDE/Prog/App/led.c:



### Macros

- `#define LED_TOGGLE_MS (500)`  
*Toggle interval time in milliseconds.*

### Functions

- `void LedInit (void)`  
*Initializes the LED.*
- `void LedToggle (void)`  
*Toggles the LED at a fixed time interval.*

### 7.752.1 Detailed Description

LED driver source file.

### 7.752.2 Function Documentation

#### 7.752.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

#### Returns

none.

### 7.752.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

#### Returns

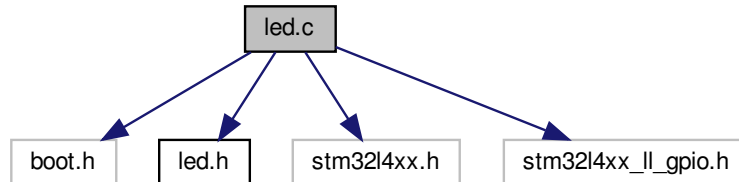
none.

## 7.753 led.c File Reference

LED driver source file.

```
#include "boot.h"
#include "led.h"
#include "stm32l4xx.h"
#include "stm32l4xx_ll_gpio.h"
```

Include dependency graph for ARMCM4\_STM32L4\_Nucleo\_L476RG\_GCC/Boot/led.c:



## Functions

- void [LedBlinkInit](#) ([blt\\_int16u](#) interval\_ms)  
*Initializes the LED blink driver.*
- void [LedBlinkTask](#) (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void [LedBlinkExit](#) (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

## Variables

- static [blt\\_int16u](#) [ledBlinkIntervalMs](#)  
*Holds the desired LED blink interval time.*

### 7.753.1 Detailed Description

LED driver source file.

### 7.753.2 Function Documentation

#### 7.753.2.1 LedBlinkExit()

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

##### Returns

none.

#### 7.753.2.2 LedBlinkInit()

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

##### Parameters

|                    |                                                                |
|--------------------|----------------------------------------------------------------|
| <i>interval_ms</i> | Specifies the desired LED blink interval time in milliseconds. |
|--------------------|----------------------------------------------------------------|

##### Returns

none.

#### 7.753.2.3 LedBlinkTask()

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

##### Returns

none.

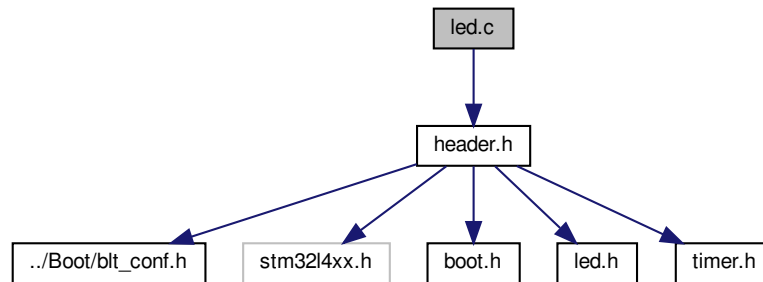


## 7.754 led.c File Reference

LED driver source file.

```
#include "header.h"
```

Include dependency graph for ARMCM4\_STM32L4\_Nucleo\_L476RG\_GCC/Prog/led.c:



### Macros

- `#define LED_TOGGLE_MS (500)`  
*Toggle interval time in milliseconds.*

### Functions

- `void LedInit (void)`  
*Initializes the LED.*
- `void LedToggle (void)`  
*Toggles the LED at a fixed time interval.*

### 7.754.1 Detailed Description

LED driver source file.

### 7.754.2 Function Documentation

#### 7.754.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

#### Returns

none.

### 7.754.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

#### Returns

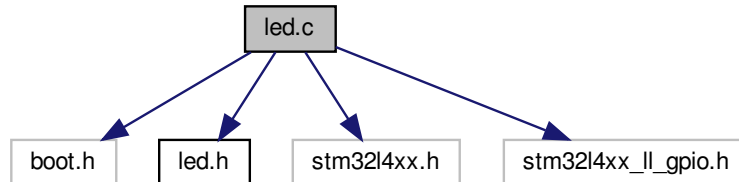
none.

## 7.755 led.c File Reference

LED driver source file.

```
#include "boot.h"
#include "led.h"
#include "stm32l4xx.h"
#include "stm32l4xx_ll_gpio.h"
```

Include dependency graph for ARMCM4\_STM32L4\_Nucleo\_L476RG\_IAR/Boot/led.c:



## Functions

- void [LedBlinkInit](#) ([blt\\_int16u](#) interval\_ms)  
*Initializes the LED blink driver.*
- void [LedBlinkTask](#) (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void [LedBlinkExit](#) (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

## Variables

- static [blt\\_int16u](#) [ledBlinkIntervalMs](#)  
*Holds the desired LED blink interval time.*

## 7.755.1 Detailed Description

LED driver source file.

## 7.755.2 Function Documentation

### 7.755.2.1 LedBlinkExit()

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

#### Returns

none.

### 7.755.2.2 LedBlinkInit()

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

#### Parameters

|                    |                                                                |
|--------------------|----------------------------------------------------------------|
| <i>interval_ms</i> | Specifies the desired LED blink interval time in milliseconds. |
|--------------------|----------------------------------------------------------------|

#### Returns

none.

### 7.755.2.3 LedBlinkTask()

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

#### Returns

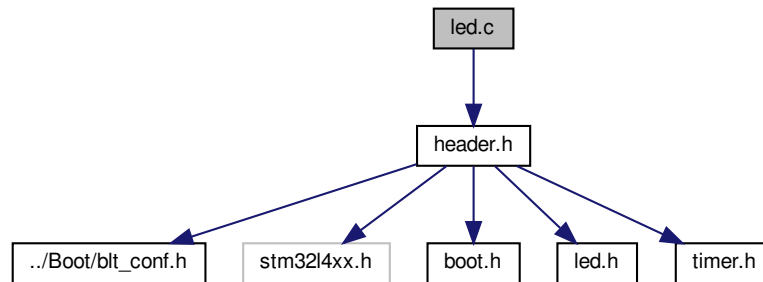
none.

## 7.756 led.c File Reference

LED driver source file.

```
#include "header.h"
```

Include dependency graph for ARMCM4\_STM32L4\_Nucleo\_L476RG\_IAR/Prog/led.c:



### Macros

- `#define LED_TOGGLE_MS (500)`  
*Toggle interval time in milliseconds.*

### Functions

- `void LedInit (void)`  
*Initializes the LED.*
- `void LedToggle (void)`  
*Toggles the LED at a fixed time interval.*

### 7.756.1 Detailed Description

LED driver source file.

### 7.756.2 Function Documentation

#### 7.756.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

#### Returns

none.

### 7.756.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

#### Returns

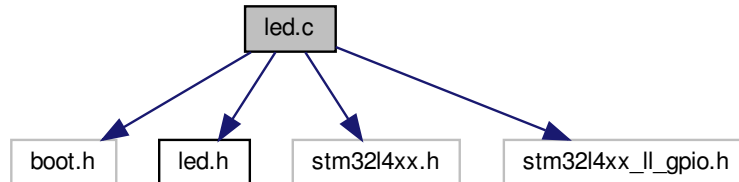
none.

## 7.757 led.c File Reference

LED driver source file.

```
#include "boot.h"
#include "led.h"
#include "stm32l4xx.h"
#include "stm32l4xx_ll_gpio.h"
```

Include dependency graph for ARMCM4\_STM32L4\_Nucleo\_L476RG\_Keil/Boot/led.c:



## Functions

- void [LedBlinkInit](#) ([blt\\_int16u](#) interval\_ms)  
*Initializes the LED blink driver.*
- void [LedBlinkTask](#) (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void [LedBlinkExit](#) (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

## Variables

- static [blt\\_int16u](#) [ledBlinkIntervalMs](#)  
*Holds the desired LED blink interval time.*

### 7.757.1 Detailed Description

LED driver source file.

### 7.757.2 Function Documentation

#### 7.757.2.1 LedBlinkExit()

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

##### Returns

none.

#### 7.757.2.2 LedBlinkInit()

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

##### Parameters

|                    |                                                                |
|--------------------|----------------------------------------------------------------|
| <i>interval_ms</i> | Specifies the desired LED blink interval time in milliseconds. |
|--------------------|----------------------------------------------------------------|

##### Returns

none.

#### 7.757.2.3 LedBlinkTask()

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

##### Returns

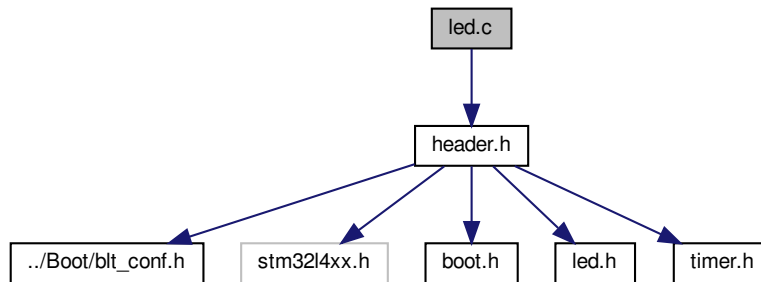
none.

## 7.758 led.c File Reference

LED driver source file.

```
#include "header.h"
```

Include dependency graph for ARMCM4\_STM32L4\_Nucleo\_L476RG\_Keil/Prog/led.c:



### Macros

- `#define LED_TOGGLE_MS (500)`  
*Toggle interval time in milliseconds.*

### Functions

- `void LedInit (void)`  
*Initializes the LED.*
- `void LedToggle (void)`  
*Toggles the LED at a fixed time interval.*

## 7.758.1 Detailed Description

LED driver source file.

## 7.758.2 Function Documentation

### 7.758.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

#### Returns

none.

### 7.758.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

#### Returns

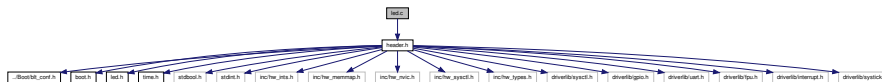
none.

## 7.759 led.c File Reference

LED driver source file.

```
#include "header.h"
```

Include dependency graph for ARMCM4\_TM4C\_DK\_TM4C123G\_IAR/Prog/led.c:



## Macros

- `#define LED_TOGGLE_MS (500)`  
*Toggle interval time in milliseconds.*

## Functions

- void `LedInit` (void)  
*Initializes the LED.*
- void `LedToggle` (void)  
*Toggles the LED at a fixed time interval.*

### 7.759.1 Detailed Description

LED driver source file.

### 7.759.2 Function Documentation



### 7.759.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

#### Returns

none.

### 7.759.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

#### Returns

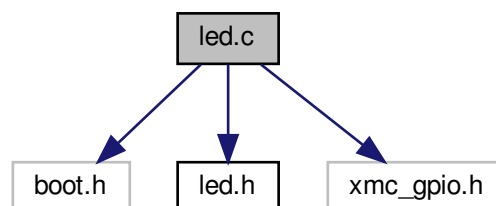
none.

## 7.760 led.c File Reference

LED driver source file.

```
#include "boot.h"
#include "led.h"
#include "xmc_gpio.h"
```

Include dependency graph for ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_GCC/Boot/led.c:



## Functions

- void [LedBlinkInit](#) ([blt\\_int16u](#) interval\_ms)  
*Initializes the LED blink driver.*
- void [LedBlinkTask](#) (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void [LedBlinkExit](#) (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

## Variables

- static [blt\\_int16u](#) [ledBlinkIntervalMs](#)  
*Holds the desired LED blink interval time.*

### 7.760.1 Detailed Description

LED driver source file.

### 7.760.2 Function Documentation

#### 7.760.2.1 LedBlinkExit()

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

#### Returns

none.

#### 7.760.2.2 LedBlinkInit()

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

#### Parameters

|                    |                                                                |
|--------------------|----------------------------------------------------------------|
| <i>interval_ms</i> | Specifies the desired LED blink interval time in milliseconds. |
|--------------------|----------------------------------------------------------------|

**Returns**

none.

**7.760.2.3 LedBlinkTask()**

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

**Returns**

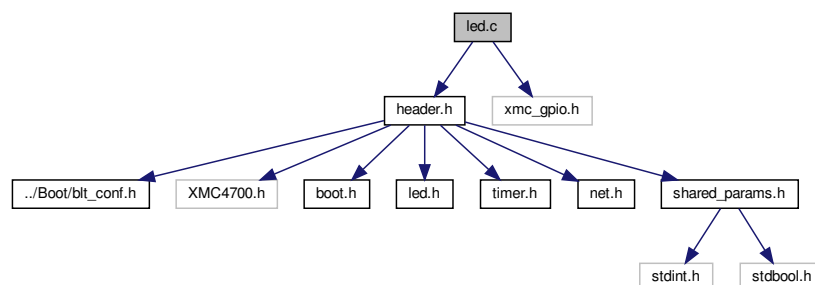
none.

**7.761 led.c File Reference**

LED driver source file.

```
#include "header.h"
#include "xmc_gpio.h"
```

Include dependency graph for ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_GCC/Prog/led.c:

**Macros**

- `#define LED_TOGGLE_MS (500)`  
*Toggle interval time in milliseconds.*

**Functions**

- void `LedInit` (void)  
*Initializes the LED.*
- void `LedToggle` (void)  
*Toggles the LED at a fixed time interval.*

### 7.761.1 Detailed Description

LED driver source file.

### 7.761.2 Function Documentation

#### 7.761.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

##### Returns

none.

#### 7.761.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

##### Returns

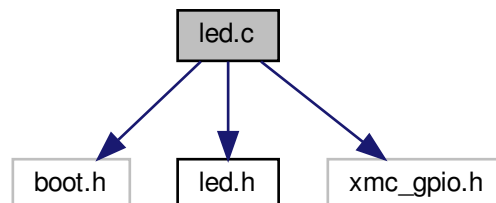
none.

## 7.762 led.c File Reference

LED driver source file.

```
#include "boot.h"
#include "led.h"
#include "xmc_gpio.h"
```

Include dependency graph for ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_IAR/Boot/led.c:



## Functions

- void [LedBlinkInit](#) ([blt\\_int16u](#) interval\_ms)  
*Initializes the LED blink driver.*
- void [LedBlinkTask](#) (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void [LedBlinkExit](#) (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

## Variables

- static [blt\\_int16u](#) [ledBlinkIntervalMs](#)  
*Holds the desired LED blink interval time.*

### 7.762.1 Detailed Description

LED driver source file.

### 7.762.2 Function Documentation

#### 7.762.2.1 LedBlinkExit()

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

#### Returns

none.

#### 7.762.2.2 LedBlinkInit()

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

#### Parameters

|                    |                                                                |
|--------------------|----------------------------------------------------------------|
| <i>interval_ms</i> | Specifies the desired LED blink interval time in milliseconds. |
|--------------------|----------------------------------------------------------------|

**Returns**

none.

**7.762.2.3 LedBlinkTask()**

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

**Returns**

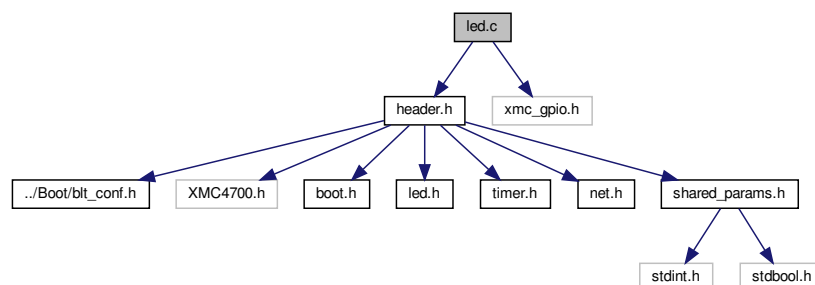
none.

**7.763 led.c File Reference**

LED driver source file.

```
#include "header.h"
#include "xmc_gpio.h"
```

Include dependency graph for ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_IAR/Prog/led.c:

**Macros**

- `#define LED_TOGGLE_MS (500)`  
*Toggle interval time in milliseconds.*

**Functions**

- void `LedInit` (void)  
*Initializes the LED.*
- void `LedToggle` (void)  
*Toggles the LED at a fixed time interval.*

### 7.763.1 Detailed Description

LED driver source file.

### 7.763.2 Function Documentation

#### 7.763.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

##### Returns

none.

#### 7.763.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

##### Returns

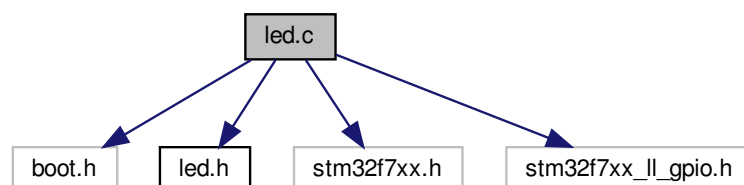
none.

## 7.764 led.c File Reference

LED driver source file.

```
#include "boot.h"
#include "led.h"
#include "stm32f7xx.h"
#include "stm32f7xx_ll_gpio.h"
```

Include dependency graph for ARMC7\_STM32F7\_Nucleo\_F746ZG\_CubeIDE/Boot/App/led.c:



## Functions

- void `LedBlinkInit` (`blt_int16u` interval\_ms)  
*Initializes the LED blink driver.*
- void `LedBlinkTask` (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void `LedBlinkExit` (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

## Variables

- static `blt_int16u` `ledBlinkIntervalMs`  
*Holds the desired LED blink interval time.*

### 7.764.1 Detailed Description

LED driver source file.

### 7.764.2 Function Documentation

#### 7.764.2.1 `LedBlinkExit()`

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

#### Returns

none.

#### 7.764.2.2 `LedBlinkInit()`

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

#### Parameters

|                          |                                                                |
|--------------------------|----------------------------------------------------------------|
| <code>interval_ms</code> | Specifies the desired LED blink interval time in milliseconds. |
|--------------------------|----------------------------------------------------------------|



**Returns**

none.

**7.764.2.3 LedBlinkTask()**

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

**Returns**

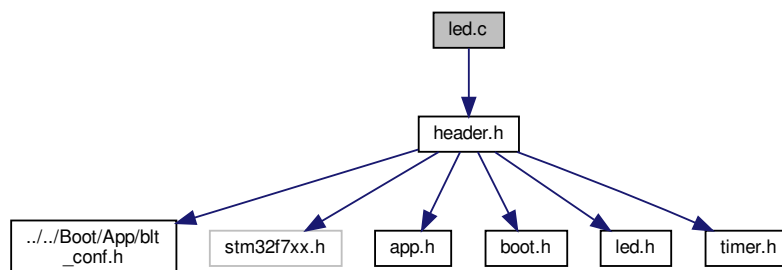
none.

**7.765 led.c File Reference**

LED driver source file.

```
#include "header.h"
```

Include dependency graph for ARMC7\_STM32F7\_Nucleo\_F746ZG\_CubeIDE/Prog/App/led.c:

**Macros**

- `#define LED_TOGGLE_MS (500)`  
*Toggle interval time in milliseconds.*

**Functions**

- `void LedInit (void)`  
*Initializes the LED.*
- `void LedToggle (void)`  
*Toggles the LED at a fixed time interval.*

### 7.765.1 Detailed Description

LED driver source file.

### 7.765.2 Function Documentation

#### 7.765.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

##### Returns

none.

#### 7.765.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

##### Returns

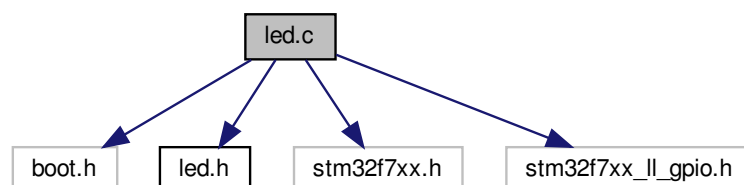
none.

## 7.766 led.c File Reference

LED driver source file.

```
#include "boot.h"
#include "led.h"
#include "stm32f7xx.h"
#include "stm32f7xx_ll_gpio.h"
```

Include dependency graph for ARMC7\_STM32F7\_Nucleo\_F746ZG\_GCC/Boot/led.c:



## Functions

- void [LedBlinkInit](#) ([blt\\_int16u](#) interval\_ms)  
*Initializes the LED blink driver.*
- void [LedBlinkTask](#) (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void [LedBlinkExit](#) (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

## Variables

- static [blt\\_int16u](#) [ledBlinkIntervalMs](#)  
*Holds the desired LED blink interval time.*

### 7.766.1 Detailed Description

LED driver source file.

### 7.766.2 Function Documentation

#### 7.766.2.1 LedBlinkExit()

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

#### Returns

none.

#### 7.766.2.2 LedBlinkInit()

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

#### Parameters

|                    |                                                                |
|--------------------|----------------------------------------------------------------|
| <i>interval_ms</i> | Specifies the desired LED blink interval time in milliseconds. |
|--------------------|----------------------------------------------------------------|

**Returns**

none.

**7.766.2.3 LedBlinkTask()**

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

**Returns**

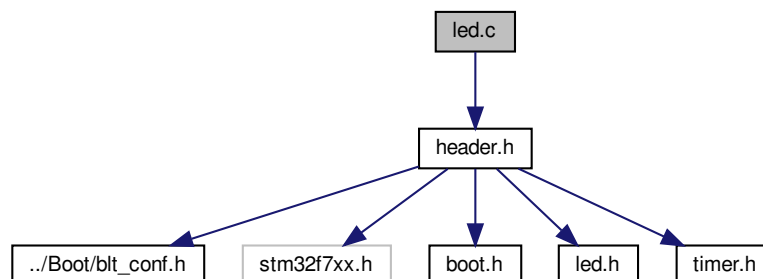
none.

**7.767 led.c File Reference**

LED driver source file.

```
#include "header.h"
```

Include dependency graph for ARMCM7\_STM32F7\_Nucleo\_F746ZG\_GCC/Prog/led.c:

**Macros**

- `#define LED_TOGGLE_MS (500)`  
*Toggle interval time in milliseconds.*

**Functions**

- void `LedInit` (void)  
*Initializes the LED.*
- void `LedToggle` (void)  
*Toggles the LED at a fixed time interval.*

### 7.767.1 Detailed Description

LED driver source file.

### 7.767.2 Function Documentation

#### 7.767.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

##### Returns

none.

#### 7.767.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

##### Returns

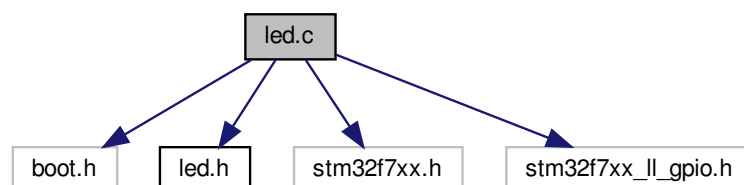
none.

## 7.768 led.c File Reference

LED driver source file.

```
#include "boot.h"
#include "led.h"
#include "stm32f7xx.h"
#include "stm32f7xx_ll_gpio.h"
```

Include dependency graph for ARMC7\_STM32F7\_Nucleo\_F746ZG\_IAR/Boot/led.c:



## Functions

- void `LedBlinkInit` (`blt_int16u` interval\_ms)  
*Initializes the LED blink driver.*
- void `LedBlinkTask` (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void `LedBlinkExit` (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

## Variables

- static `blt_int16u` `ledBlinkIntervalMs`  
*Holds the desired LED blink interval time.*

### 7.768.1 Detailed Description

LED driver source file.

### 7.768.2 Function Documentation

#### 7.768.2.1 `LedBlinkExit()`

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

#### Returns

none.

#### 7.768.2.2 `LedBlinkInit()`

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

#### Parameters

|                          |                                                                |
|--------------------------|----------------------------------------------------------------|
| <code>interval_ms</code> | Specifies the desired LED blink interval time in milliseconds. |
|--------------------------|----------------------------------------------------------------|

**Returns**

none.

**7.768.2.3 LedBlinkTask()**

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

**Returns**

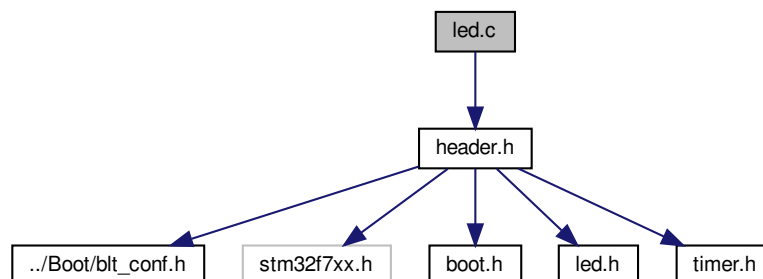
none.

**7.769 led.c File Reference**

LED driver source file.

```
#include "header.h"
```

Include dependency graph for ARMCM7\_STM32F7\_Nucleo\_F746ZG\_IAR/Prog/led.c:

**Macros**

- `#define LED_TOGGLE_MS (500)`  
*Toggle interval time in milliseconds.*

**Functions**

- void `LedInit` (void)  
*Initializes the LED.*
- void `LedToggle` (void)  
*Toggles the LED at a fixed time interval.*

### 7.769.1 Detailed Description

LED driver source file.

### 7.769.2 Function Documentation

#### 7.769.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

##### Returns

none.

#### 7.769.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

##### Returns

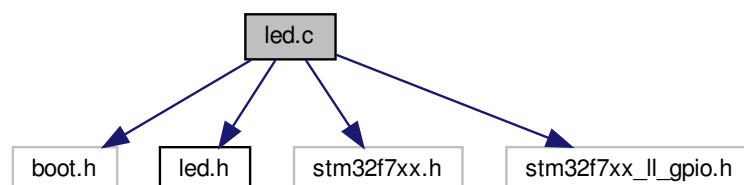
none.

## 7.770 led.c File Reference

LED driver source file.

```
#include "boot.h"
#include "led.h"
#include "stm32f7xx.h"
#include "stm32f7xx_ll_gpio.h"
```

Include dependency graph for ARMC77\_STM32F7\_Nucleo\_F746ZG\_Keil/Boot/led.c:





## Functions

- void [LedBlinkInit](#) ([blt\\_int16u](#) interval\_ms)  
*Initializes the LED blink driver.*
- void [LedBlinkTask](#) (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void [LedBlinkExit](#) (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

## Variables

- static [blt\\_int16u](#) [ledBlinkIntervalMs](#)  
*Holds the desired LED blink interval time.*

### 7.770.1 Detailed Description

LED driver source file.

### 7.770.2 Function Documentation

#### 7.770.2.1 LedBlinkExit()

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

#### Returns

none.

#### 7.770.2.2 LedBlinkInit()

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

#### Parameters

|                    |                                                                |
|--------------------|----------------------------------------------------------------|
| <i>interval_ms</i> | Specifies the desired LED blink interval time in milliseconds. |
|--------------------|----------------------------------------------------------------|

**Returns**

none.

**7.770.2.3 LedBlinkTask()**

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

**Returns**

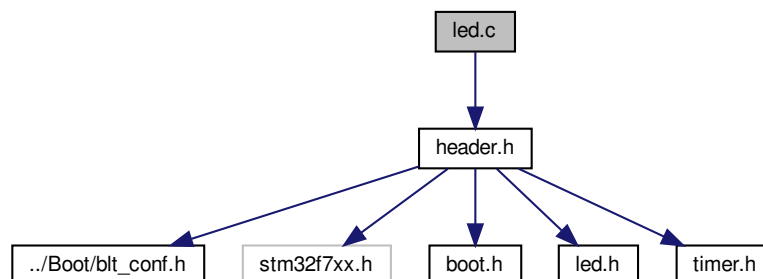
none.

**7.771 led.c File Reference**

LED driver source file.

```
#include "header.h"
```

Include dependency graph for ARMCM7\_STM32F7\_Nucleo\_F746ZG\_Keil/Prog/led.c:

**Macros**

- `#define LED_TOGGLE_MS (500)`  
*Toggle interval time in milliseconds.*

**Functions**

- `void LedInit (void)`  
*Initializes the LED.*
- `void LedToggle (void)`  
*Toggles the LED at a fixed time interval.*

### 7.771.1 Detailed Description

LED driver source file.

### 7.771.2 Function Documentation

#### 7.771.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

##### Returns

none.

#### 7.771.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

##### Returns

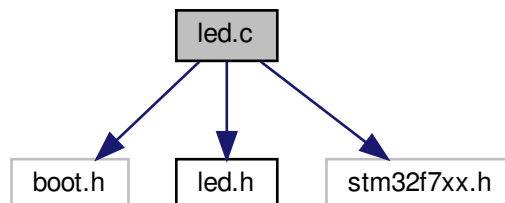
none.

## 7.772 led.c File Reference

LED driver source file.

```
#include "boot.h"
#include "led.h"
#include "stm32f7xx.h"
```

Include dependency graph for ARMCM7\_STM32F7\_Nucleo\_F767ZI\_CubeIDE/Boot/App/led.c:



## Functions

- void `LedBlinkInit` (`blt_int16u` interval\_ms)  
*Initializes the LED blink driver.*
- void `LedBlinkTask` (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void `LedBlinkExit` (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

## Variables

- static `blt_int16u` `ledBlinkIntervalMs`  
*Holds the desired LED blink interval time.*

### 7.772.1 Detailed Description

LED driver source file.

### 7.772.2 Function Documentation

#### 7.772.2.1 `LedBlinkExit()`

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

#### Returns

none.

#### 7.772.2.2 `LedBlinkInit()`

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

#### Parameters

|                          |                                                                |
|--------------------------|----------------------------------------------------------------|
| <code>interval_ms</code> | Specifies the desired LED blink interval time in milliseconds. |
|--------------------------|----------------------------------------------------------------|

**Returns**

none.

**7.772.2.3 LedBlinkTask()**

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

**Returns**

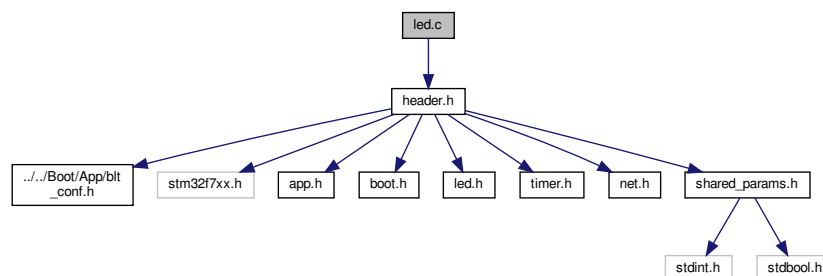
none.

**7.773 led.c File Reference**

LED driver source file.

```
#include "header.h"
```

Include dependency graph for ARMCM7\_STM32F7\_Nucleo\_F767ZI\_CubeIDE/Prog/App/led.c:

**Macros**

- `#define LED_TOGGLE_MS (500)`  
*Toggle interval time in milliseconds.*

**Functions**

- void `LedInit` (void)  
*Initializes the LED.*
- void `LedToggle` (void)  
*Toggles the LED at a fixed time interval.*

### 7.773.1 Detailed Description

LED driver source file.

### 7.773.2 Function Documentation

#### 7.773.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

##### Returns

none.

#### 7.773.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

##### Returns

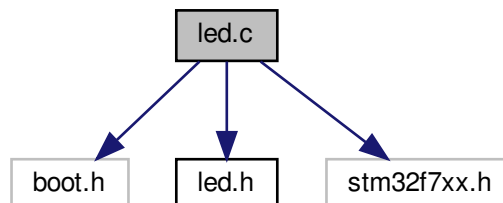
none.

## 7.774 led.c File Reference

LED driver source file.

```
#include "boot.h"
#include "led.h"
#include "stm32f7xx.h"
```

Include dependency graph for ARMCM7\_STM32F7\_Nucleo\_F767ZI\_GCC/Boot/led.c:



## Functions

- void [LedBlinkInit](#) ([blt\\_int16u](#) interval\_ms)  
*Initializes the LED blink driver.*
- void [LedBlinkTask](#) (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void [LedBlinkExit](#) (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

## Variables

- static [blt\\_int16u](#) [ledBlinkIntervalMs](#)  
*Holds the desired LED blink interval time.*

### 7.774.1 Detailed Description

LED driver source file.

### 7.774.2 Function Documentation

#### 7.774.2.1 LedBlinkExit()

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

#### Returns

none.

#### 7.774.2.2 LedBlinkInit()

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

#### Parameters

|                    |                                                                |
|--------------------|----------------------------------------------------------------|
| <i>interval_ms</i> | Specifies the desired LED blink interval time in milliseconds. |
|--------------------|----------------------------------------------------------------|

**Returns**

none.

**7.774.2.3 LedBlinkTask()**

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

**Returns**

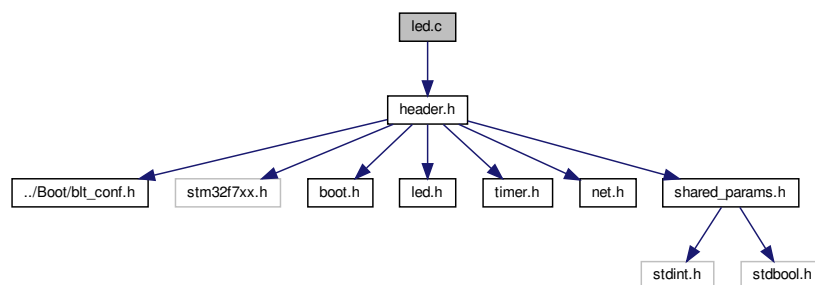
none.

**7.775 led.c File Reference**

LED driver source file.

```
#include "header.h"
```

Include dependency graph for ARMCM7\_STM32F7\_Nucleo\_F767ZI\_GCC/Prog/led.c:

**Macros**

- `#define LED_TOGGLE_MS (500)`  
*Toggle interval time in milliseconds.*

**Functions**

- void `LedInit` (void)  
*Initializes the LED.*
- void `LedToggle` (void)  
*Toggles the LED at a fixed time interval.*



### 7.775.1 Detailed Description

LED driver source file.

### 7.775.2 Function Documentation

#### 7.775.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

##### Returns

none.

#### 7.775.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

##### Returns

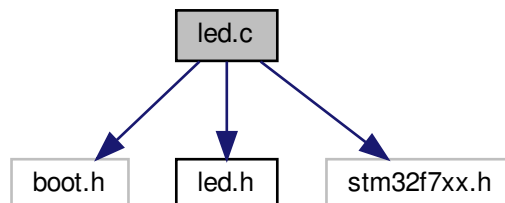
none.

## 7.776 led.c File Reference

LED driver source file.

```
#include "boot.h"
#include "led.h"
#include "stm32f7xx.h"
```

Include dependency graph for ARMCM7\_STM32F7\_Nucleo\_F767ZI\_IAR/Boot/led.c:



## Functions

- void `LedBlinkInit` (`blt_int16u` interval\_ms)  
*Initializes the LED blink driver.*
- void `LedBlinkTask` (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void `LedBlinkExit` (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

## Variables

- static `blt_int16u` `ledBlinkIntervalMs`  
*Holds the desired LED blink interval time.*

### 7.776.1 Detailed Description

LED driver source file.

### 7.776.2 Function Documentation

#### 7.776.2.1 `LedBlinkExit()`

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

#### Returns

none.

#### 7.776.2.2 `LedBlinkInit()`

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

#### Parameters

|                          |                                                                |
|--------------------------|----------------------------------------------------------------|
| <code>interval_ms</code> | Specifies the desired LED blink interval time in milliseconds. |
|--------------------------|----------------------------------------------------------------|

**Returns**

none.

**7.776.2.3 LedBlinkTask()**

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

**Returns**

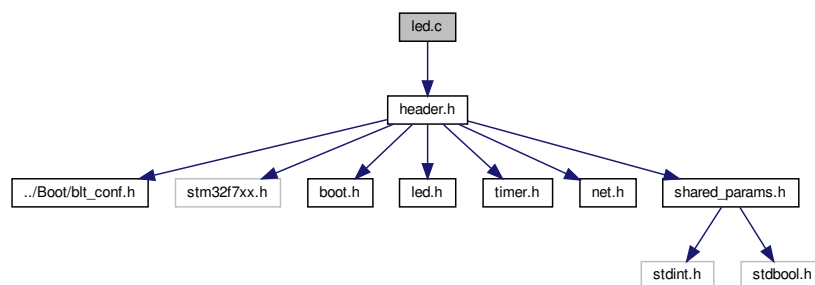
none.

**7.777 led.c File Reference**

LED driver source file.

```
#include "header.h"
```

Include dependency graph for ARMC7\_STM32F7\_Nucleo\_F767ZI\_IAR/Prog/led.c:

**Macros**

- `#define LED_TOGGLE_MS (500)`  
*Toggle interval time in milliseconds.*

**Functions**

- void `LedInit` (void)  
*Initializes the LED.*
- void `LedToggle` (void)  
*Toggles the LED at a fixed time interval.*

### 7.777.1 Detailed Description

LED driver source file.

### 7.777.2 Function Documentation

#### 7.777.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

##### Returns

none.

#### 7.777.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

##### Returns

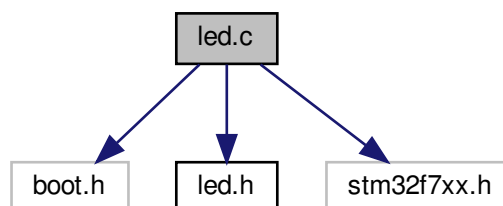
none.

## 7.778 led.c File Reference

LED driver source file.

```
#include "boot.h"
#include "led.h"
#include "stm32f7xx.h"
```

Include dependency graph for ARMCM7\_STM32F7\_Nucleo\_F767ZI\_Keil/Boot/led.c:



## Functions

- void [LedBlinkInit](#) ([blt\\_int16u](#) interval\_ms)  
*Initializes the LED blink driver.*
- void [LedBlinkTask](#) (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void [LedBlinkExit](#) (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

## Variables

- static [blt\\_int16u](#) [ledBlinkIntervalMs](#)  
*Holds the desired LED blink interval time.*

### 7.778.1 Detailed Description

LED driver source file.

### 7.778.2 Function Documentation

#### 7.778.2.1 LedBlinkExit()

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

#### Returns

none.

#### 7.778.2.2 LedBlinkInit()

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

#### Parameters

|                    |                                                                |
|--------------------|----------------------------------------------------------------|
| <i>interval_ms</i> | Specifies the desired LED blink interval time in milliseconds. |
|--------------------|----------------------------------------------------------------|

**Returns**

none.

**7.778.2.3 LedBlinkTask()**

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

**Returns**

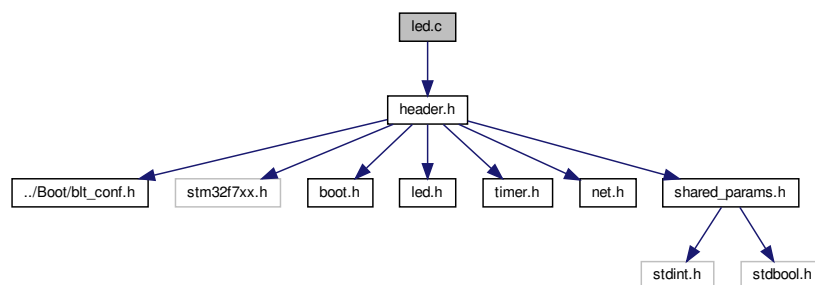
none.

**7.779 led.c File Reference**

LED driver source file.

```
#include "header.h"
```

Include dependency graph for ARMC7\_STM32F7\_Nucleo\_F767ZI\_Keil/Prog/led.c:

**Macros**

- `#define LED_TOGGLE_MS (500)`  
*Toggle interval time in milliseconds.*

**Functions**

- void `LedInit` (void)  
*Initializes the LED.*
- void `LedToggle` (void)  
*Toggles the LED at a fixed time interval.*

### 7.779.1 Detailed Description

LED driver source file.

### 7.779.2 Function Documentation

#### 7.779.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

##### Returns

none.

#### 7.779.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

##### Returns

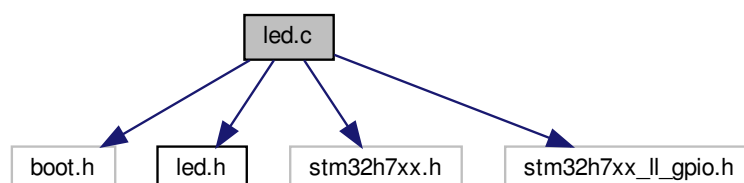
none.

## 7.780 led.c File Reference

LED driver source file.

```
#include "boot.h"
#include "led.h"
#include "stm32h7xx.h"
#include "stm32h7xx_ll_gpio.h"
```

Include dependency graph for ARMCM7\_STM32H7\_Nucleo\_H743ZI\_CubeIDE/Boot/App/led.c:



## Functions

- void `LedBlinkInit` (`blt_int16u` interval\_ms)  
*Initializes the LED blink driver.*
- void `LedBlinkTask` (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void `LedBlinkExit` (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

## Variables

- static `blt_int16u` `ledBlinkIntervalMs`  
*Holds the desired LED blink interval time.*

### 7.780.1 Detailed Description

LED driver source file.

### 7.780.2 Function Documentation

#### 7.780.2.1 `LedBlinkExit()`

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

#### Returns

none.

#### 7.780.2.2 `LedBlinkInit()`

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

#### Parameters

|                          |                                                                |
|--------------------------|----------------------------------------------------------------|
| <code>interval_ms</code> | Specifies the desired LED blink interval time in milliseconds. |
|--------------------------|----------------------------------------------------------------|



**Returns**

none.

**7.780.2.3 LedBlinkTask()**

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

**Returns**

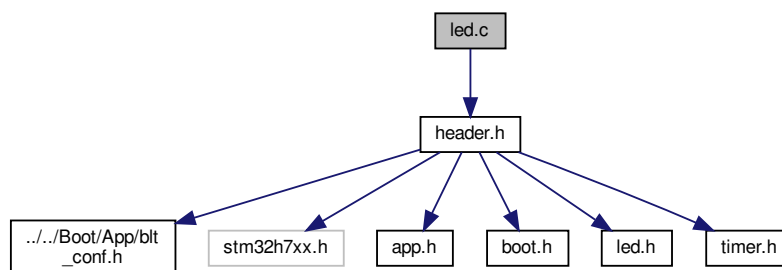
none.

**7.781 led.c File Reference**

LED driver source file.

```
#include "header.h"
```

Include dependency graph for ARMC7\_STM32H7\_Nucleo\_H743ZI\_CubeIDE/Prog/App/led.c:

**Macros**

- `#define LED_TOGGLE_MS (500)`  
*Toggle interval time in milliseconds.*

**Functions**

- `void LedInit (void)`  
*Initializes the LED.*
- `void LedToggle (void)`  
*Toggles the LED at a fixed time interval.*

### 7.781.1 Detailed Description

LED driver source file.

### 7.781.2 Function Documentation

#### 7.781.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

##### Returns

none.

#### 7.781.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

##### Returns

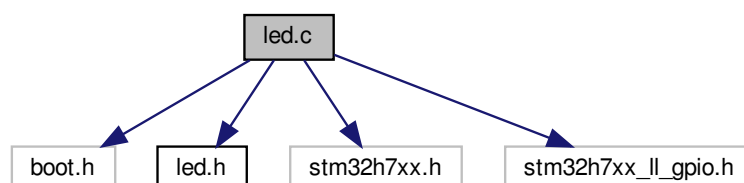
none.

## 7.782 led.c File Reference

LED driver source file.

```
#include "boot.h"
#include "led.h"
#include "stm32h7xx.h"
#include "stm32h7xx_ll_gpio.h"
```

Include dependency graph for ARMCM7\_STM32H7\_Nucleo\_H743ZI\_GCC/Boot/led.c:



## Functions

- void [LedBlinkInit](#) ([blt\\_int16u](#) interval\_ms)  
*Initializes the LED blink driver.*
- void [LedBlinkTask](#) (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void [LedBlinkExit](#) (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

## Variables

- static [blt\\_int16u](#) [ledBlinkIntervalMs](#)  
*Holds the desired LED blink interval time.*

### 7.782.1 Detailed Description

LED driver source file.

### 7.782.2 Function Documentation

#### 7.782.2.1 LedBlinkExit()

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

#### Returns

none.

#### 7.782.2.2 LedBlinkInit()

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

#### Parameters

|                    |                                                                |
|--------------------|----------------------------------------------------------------|
| <i>interval_ms</i> | Specifies the desired LED blink interval time in milliseconds. |
|--------------------|----------------------------------------------------------------|

**Returns**

none.

**7.782.2.3 LedBlinkTask()**

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

**Returns**

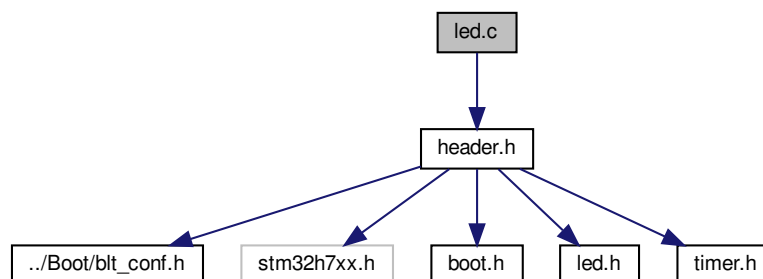
none.

**7.783 led.c File Reference**

LED driver source file.

```
#include "header.h"
```

Include dependency graph for ARMCM7\_STM32H7\_Nucleo\_H743ZI\_GCC/Prog/led.c:

**Macros**

- `#define LED_TOGGLE_MS (500)`  
*Toggle interval time in milliseconds.*

**Functions**

- void `LedInit` (void)  
*Initializes the LED.*
- void `LedToggle` (void)  
*Toggles the LED at a fixed time interval.*

### 7.783.1 Detailed Description

LED driver source file.

### 7.783.2 Function Documentation

#### 7.783.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

##### Returns

none.

#### 7.783.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

##### Returns

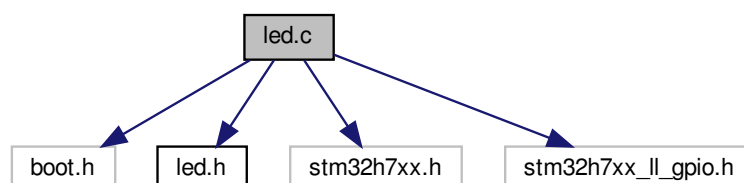
none.

## 7.784 led.c File Reference

LED driver source file.

```
#include "boot.h"
#include "led.h"
#include "stm32h7xx.h"
#include "stm32h7xx_ll_gpio.h"
```

Include dependency graph for ARMCM7\_STM32H7\_Nucleo\_H743ZI\_IAR/Boot/led.c:



## Functions

- void `LedBlinkInit` (`blt_int16u` interval\_ms)  
*Initializes the LED blink driver.*
- void `LedBlinkTask` (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void `LedBlinkExit` (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

## Variables

- static `blt_int16u` `ledBlinkIntervalMs`  
*Holds the desired LED blink interval time.*

### 7.784.1 Detailed Description

LED driver source file.

### 7.784.2 Function Documentation

#### 7.784.2.1 `LedBlinkExit()`

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

#### Returns

none.

#### 7.784.2.2 `LedBlinkInit()`

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

#### Parameters

|                          |                                                                |
|--------------------------|----------------------------------------------------------------|
| <code>interval_ms</code> | Specifies the desired LED blink interval time in milliseconds. |
|--------------------------|----------------------------------------------------------------|

**Returns**

none.

**7.784.2.3 LedBlinkTask()**

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

**Returns**

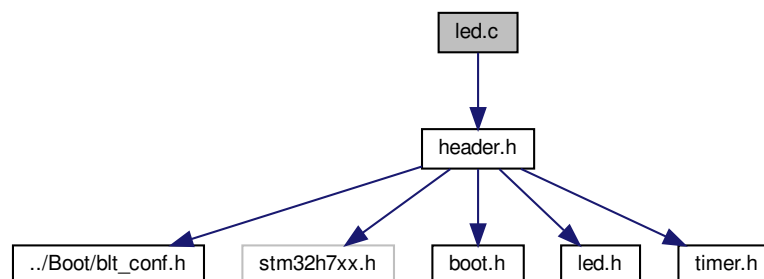
none.

**7.785 led.c File Reference**

LED driver source file.

```
#include "header.h"
```

Include dependency graph for ARMCM7\_STM32H7\_Nucleo\_H743ZI\_IAR/Prog/led.c:

**Macros**

- `#define LED_TOGGLE_MS (500)`  
*Toggle interval time in milliseconds.*

**Functions**

- void `LedInit` (void)  
*Initializes the LED.*
- void `LedToggle` (void)  
*Toggles the LED at a fixed time interval.*

## 7.785.1 Detailed Description

LED driver source file.

## 7.785.2 Function Documentation

### 7.785.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

#### Returns

none.

### 7.785.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

#### Returns

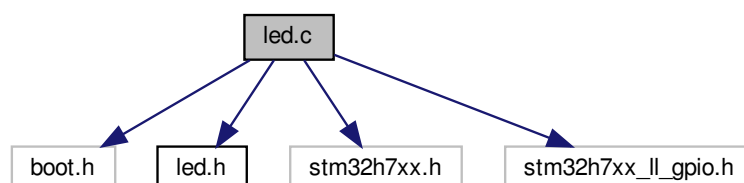
none.

## 7.786 led.c File Reference

LED driver source file.

```
#include "boot.h"
#include "led.h"
#include "stm32h7xx.h"
#include "stm32h7xx_ll_gpio.h"
```

Include dependency graph for ARMCM7\_STM32H7\_Nucleo\_H743ZI\_Keil/Boot/led.c:





## Functions

- void [LedBlinkInit](#) ([blt\\_int16u](#) interval\_ms)  
*Initializes the LED blink driver.*
- void [LedBlinkTask](#) (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void [LedBlinkExit](#) (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

## Variables

- static [blt\\_int16u](#) [ledBlinkIntervalMs](#)  
*Holds the desired LED blink interval time.*

### 7.786.1 Detailed Description

LED driver source file.

### 7.786.2 Function Documentation

#### 7.786.2.1 LedBlinkExit()

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

#### Returns

none.

#### 7.786.2.2 LedBlinkInit()

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

#### Parameters

|                    |                                                                |
|--------------------|----------------------------------------------------------------|
| <i>interval_ms</i> | Specifies the desired LED blink interval time in milliseconds. |
|--------------------|----------------------------------------------------------------|

**Returns**

none.

**7.786.2.3 LedBlinkTask()**

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

**Returns**

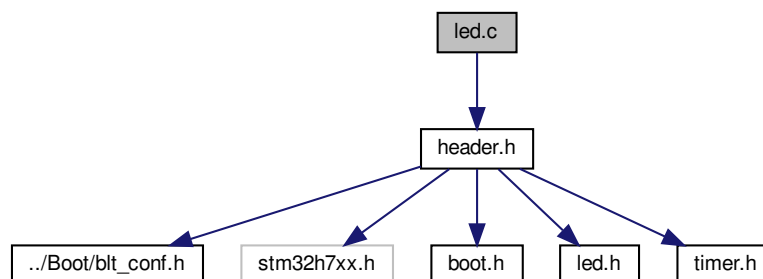
none.

**7.787 led.c File Reference**

LED driver source file.

```
#include "header.h"
```

Include dependency graph for ARMCM7\_STM32H7\_Nucleo\_H743ZI\_Keil/Prog/led.c:

**Macros**

- `#define LED_TOGGLE_MS (500)`  
*Toggle interval time in milliseconds.*

**Functions**

- void `LedInit` (void)  
*Initializes the LED.*
- void `LedToggle` (void)  
*Toggles the LED at a fixed time interval.*

### 7.787.1 Detailed Description

LED driver source file.

### 7.787.2 Function Documentation

#### 7.787.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

##### Returns

none.

#### 7.787.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

##### Returns

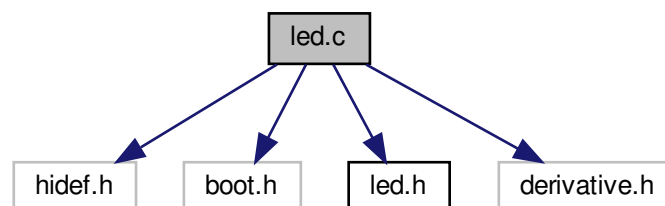
none.

## 7.788 led.c File Reference

LED driver source file.

```
#include <hidef.h>
#include "boot.h"
#include "led.h"
#include "derivative.h"
```

Include dependency graph for HCS12\_DevKit\_S12G128\_CodeWarrior/Boot/led.c:



## Functions

- void `LedBlinkInit` (`blt_int16u` interval\_ms)  
*Initializes the LED blink driver.*
- void `LedBlinkTask` (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void `LedBlinkExit` (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

## Variables

- static `blt_int16u` `ledBlinkIntervalMs`  
*Holds the desired LED blink interval time.*

### 7.788.1 Detailed Description

LED driver source file.

### 7.788.2 Function Documentation

#### 7.788.2.1 `LedBlinkExit()`

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

#### Returns

none.

#### 7.788.2.2 `LedBlinkInit()`

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

#### Parameters

|                          |                                                                |
|--------------------------|----------------------------------------------------------------|
| <code>interval_ms</code> | Specifies the desired LED blink interval time in milliseconds. |
|--------------------------|----------------------------------------------------------------|

**Returns**

none.

**7.788.2.3 LedBlinkTask()**

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

**Returns**

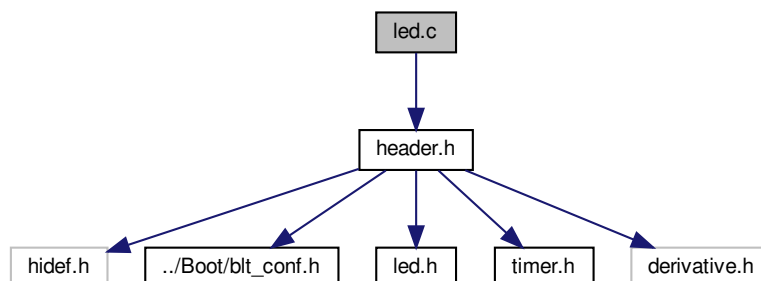
none.

**7.789 led.c File Reference**

LED driver source file.

```
#include "header.h"
```

Include dependency graph for HCS12\_DevKit\_S12G128\_CodeWarrior/Prog/led.c:

**Macros**

- `#define LED_TOGGLE_MS (500)`  
*Toggle interval time in milliseconds.*

**Functions**

- `void LedInit (void)`  
*Initializes the LED.*
- `void LedToggle (void)`  
*Toggles the LED at a fixed time interval.*

### 7.789.1 Detailed Description

LED driver source file.

### 7.789.2 Function Documentation

#### 7.789.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

##### Returns

none.

#### 7.789.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

##### Returns

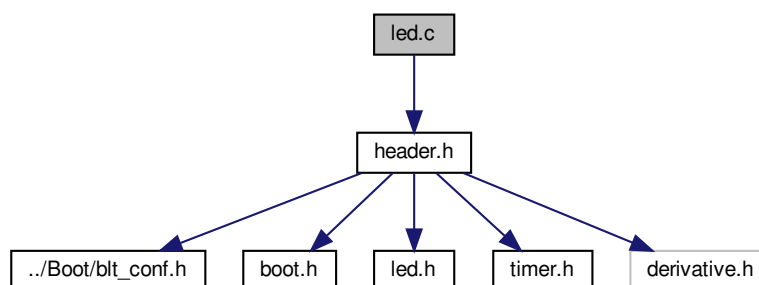
none.

## 7.790 led.c File Reference

LED driver source file.

```
#include "header.h"
```

Include dependency graph for HCS12\_Evbplus\_Dragon12p\_CodeWarrior/Prog/led.c:



## Macros

- `#define LED_TOGGLE_MS (500)`  
*Toggle interval time in milliseconds.*

## Functions

- `void LedInit (void)`  
*Initializes the LED.*
- `void LedToggle (void)`  
*Toggles the LED at a fixed time interval.*

### 7.790.1 Detailed Description

LED driver source file.

### 7.790.2 Function Documentation

#### 7.790.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

##### Returns

none.

#### 7.790.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

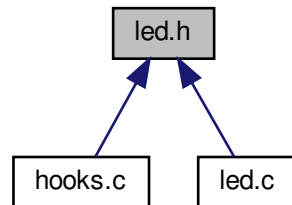
##### Returns

none.

## 7.791 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:



### Functions

- void [LedBlinkInit](#) ([blt\\_int16u](#) interval\_ms)  
*Initializes the LED blink driver.*
- void [LedBlinkTask](#) (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void [LedBlinkExit](#) (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

### 7.791.1 Detailed Description

LED driver header file.

### 7.791.2 Function Documentation

#### 7.791.2.1 LedBlinkExit()

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

#### Returns

none.

#### 7.791.2.2 LedBlinkInit()

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.



## Parameters

|                          |                                                                |
|--------------------------|----------------------------------------------------------------|
| <code>interval_ms</code> | Specifies the desired LED blink interval time in milliseconds. |
|--------------------------|----------------------------------------------------------------|

## Returns

none.

## 7.791.2.3 LedBlinkTask()

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

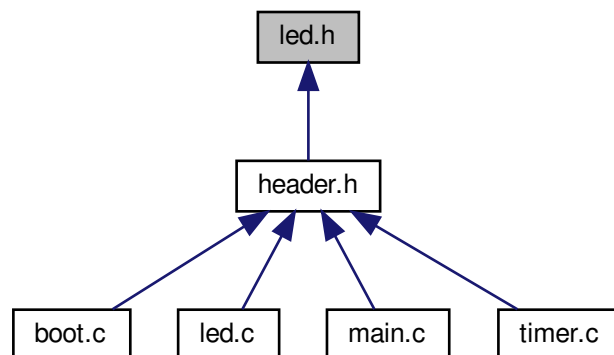
## Returns

none.

## 7.792 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:



## Functions

- void [LedInit](#) (void)  
*Initializes the LED.*
- void [LedToggle](#) (void)  
*Toggles the LED at a fixed time interval.*

### 7.792.1 Detailed Description

LED driver header file.

### 7.792.2 Function Documentation

#### 7.792.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

##### Returns

none.

Initializes the LED.

##### Returns

none.

#### 7.792.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

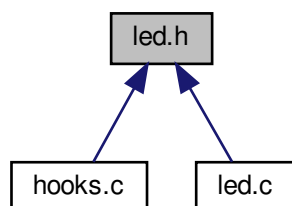
##### Returns

none.

## 7.793 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:



## Functions

- void [LedBlinkInit](#) ([blt\\_int16u](#) interval\_ms)  
*Initializes the LED blink driver.*
- void [LedBlinkTask](#) (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void [LedBlinkExit](#) (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

### 7.793.1 Detailed Description

LED driver header file.

### 7.793.2 Function Documentation

#### 7.793.2.1 LedBlinkExit()

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

#### Returns

none.

#### 7.793.2.2 LedBlinkInit()

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

#### Parameters

|                    |                                                                |
|--------------------|----------------------------------------------------------------|
| <i>interval_ms</i> | Specifies the desired LED blink interval time in milliseconds. |
|--------------------|----------------------------------------------------------------|

#### Returns

none.

### 7.793.2.3 LedBlinkTask()

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

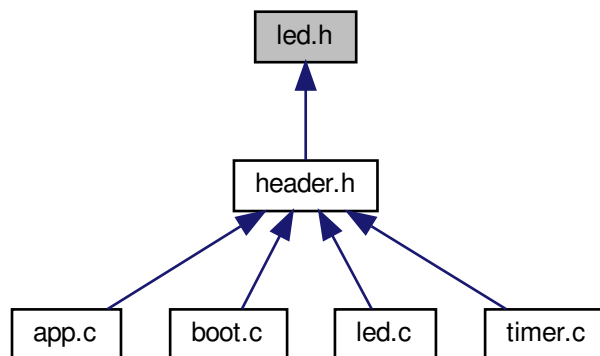
#### Returns

none.

## 7.794 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:



### Functions

- void [LedInit](#) (void)  
*Initializes the LED.*
- void [LedToggle](#) (void)  
*Toggles the LED at a fixed time interval.*

### 7.794.1 Detailed Description

LED driver header file.

### 7.794.2 Function Documentation

### 7.794.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

#### Returns

none.

Initializes the LED.

#### Returns

none.

### 7.794.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

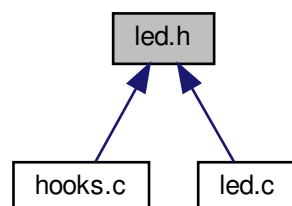
#### Returns

none.

## 7.795 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:



## Functions

- void [LedBlinkInit](#) ([blt\\_int16u](#) interval\_ms)  
*Initializes the LED blink driver.*
- void [LedBlinkTask](#) (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void [LedBlinkExit](#) (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

### 7.795.1 Detailed Description

LED driver header file.

### 7.795.2 Function Documentation

#### 7.795.2.1 LedBlinkExit()

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

#### Returns

none.

#### 7.795.2.2 LedBlinkInit()

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

#### Parameters

|                    |                                                                |
|--------------------|----------------------------------------------------------------|
| <i>interval_ms</i> | Specifies the desired LED blink interval time in milliseconds. |
|--------------------|----------------------------------------------------------------|

#### Returns

none.

### 7.795.2.3 LedBlinkTask()

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

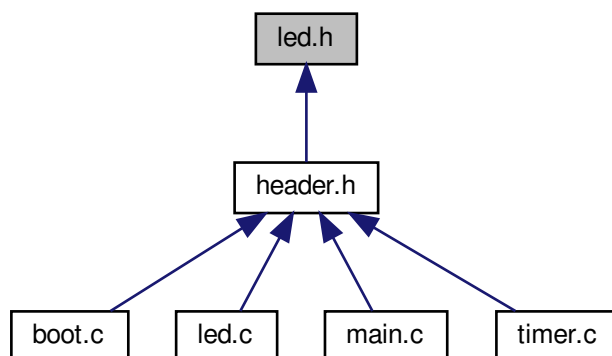
#### Returns

none.

## 7.796 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:



### Functions

- void [LedInit](#) (void)  
*Initializes the LED.*
- void [LedToggle](#) (void)  
*Toggles the LED at a fixed time interval.*

### 7.796.1 Detailed Description

LED driver header file.

### 7.796.2 Function Documentation

### 7.796.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

#### Returns

none.

Initializes the LED.

#### Returns

none.

### 7.796.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

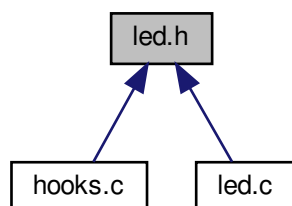
#### Returns

none.

## 7.797 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:





## Functions

- void [LedBlinkInit](#) ([blt\\_int16u](#) interval\_ms)  
*Initializes the LED blink driver.*
- void [LedBlinkTask](#) (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void [LedBlinkExit](#) (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

### 7.797.1 Detailed Description

LED driver header file.

### 7.797.2 Function Documentation

#### 7.797.2.1 LedBlinkExit()

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

#### Returns

none.

#### 7.797.2.2 LedBlinkInit()

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

#### Parameters

|                    |                                                                |
|--------------------|----------------------------------------------------------------|
| <i>interval_ms</i> | Specifies the desired LED blink interval time in milliseconds. |
|--------------------|----------------------------------------------------------------|

#### Returns

none.

### 7.797.2.3 LedBlinkTask()

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

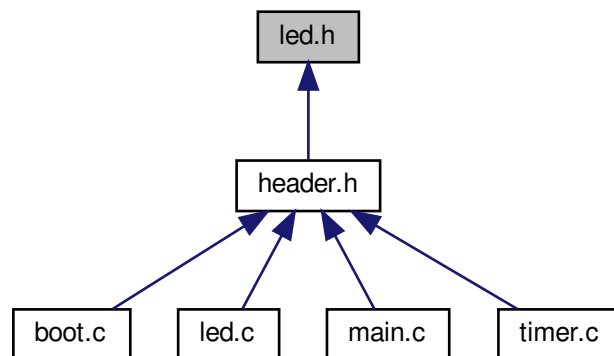
#### Returns

none.

## 7.798 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:



### Functions

- void [LedInit](#) (void)  
*Initializes the LED.*
- void [LedToggle](#) (void)  
*Toggles the LED at a fixed time interval.*

### 7.798.1 Detailed Description

LED driver header file.

### 7.798.2 Function Documentation

### 7.798.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

#### Returns

none.

Initializes the LED.

#### Returns

none.

### 7.798.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

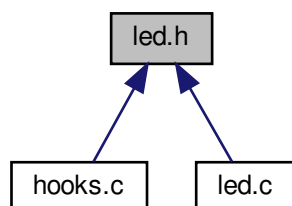
#### Returns

none.

## 7.799 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:



## Functions

- void [LedBlinkInit](#) ([blt\\_int16u](#) interval\_ms)  
*Initializes the LED blink driver.*
- void [LedBlinkTask](#) (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void [LedBlinkExit](#) (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

### 7.799.1 Detailed Description

LED driver header file.

### 7.799.2 Function Documentation

#### 7.799.2.1 LedBlinkExit()

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

#### Returns

none.

#### 7.799.2.2 LedBlinkInit()

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

#### Parameters

|                             |                                                                |
|-----------------------------|----------------------------------------------------------------|
| <a href="#">interval_ms</a> | Specifies the desired LED blink interval time in milliseconds. |
|-----------------------------|----------------------------------------------------------------|

#### Returns

none.

### 7.799.2.3 LedBlinkTask()

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

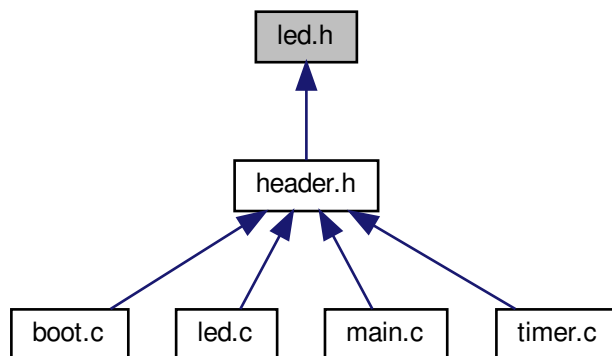
#### Returns

none.

## 7.800 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:



### Functions

- void [LedInit](#) (void)  
*Initializes the LED.*
- void [LedToggle](#) (void)  
*Toggles the LED at a fixed time interval.*

### 7.800.1 Detailed Description

LED driver header file.

### 7.800.2 Function Documentation

### 7.800.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

#### Returns

none.

Initializes the LED.

#### Returns

none.

### 7.800.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

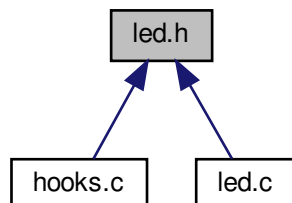
#### Returns

none.

## 7.801 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:



## Functions

- void [LedBlinkInit](#) ([blt\\_int16u](#) interval\_ms)  
*Initializes the LED blink driver.*
- void [LedBlinkTask](#) (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void [LedBlinkExit](#) (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

### 7.801.1 Detailed Description

LED driver header file.

### 7.801.2 Function Documentation

#### 7.801.2.1 LedBlinkExit()

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

#### Returns

none.

#### 7.801.2.2 LedBlinkInit()

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

#### Parameters

|                    |                                                                |
|--------------------|----------------------------------------------------------------|
| <i>interval_ms</i> | Specifies the desired LED blink interval time in milliseconds. |
|--------------------|----------------------------------------------------------------|

#### Returns

none.

### 7.801.2.3 LedBlinkTask()

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

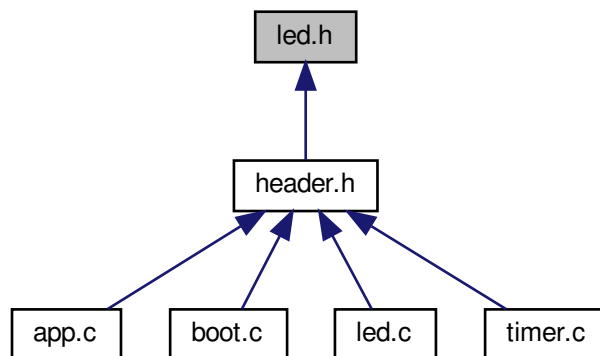
#### Returns

none.

## 7.802 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:



### Functions

- void [LedInit](#) (void)  
*Initializes the LED.*
- void [LedToggle](#) (void)  
*Toggles the LED at a fixed time interval.*

### 7.802.1 Detailed Description

LED driver header file.

### 7.802.2 Function Documentation



### 7.802.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

#### Returns

none.

Initializes the LED.

#### Returns

none.

### 7.802.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

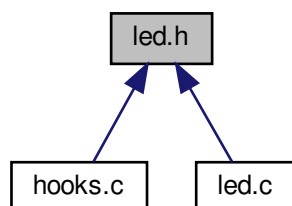
#### Returns

none.

## 7.803 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:



## Functions

- void [LedBlinkInit](#) ([blt\\_int16u](#) interval\_ms)  
*Initializes the LED blink driver.*
- void [LedBlinkTask](#) (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void [LedBlinkExit](#) (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

### 7.803.1 Detailed Description

LED driver header file.

### 7.803.2 Function Documentation

#### 7.803.2.1 LedBlinkExit()

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

#### Returns

none.

#### 7.803.2.2 LedBlinkInit()

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

#### Parameters

|                    |                                                                |
|--------------------|----------------------------------------------------------------|
| <i>interval_ms</i> | Specifies the desired LED blink interval time in milliseconds. |
|--------------------|----------------------------------------------------------------|

#### Returns

none.

### 7.803.2.3 LedBlinkTask()

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

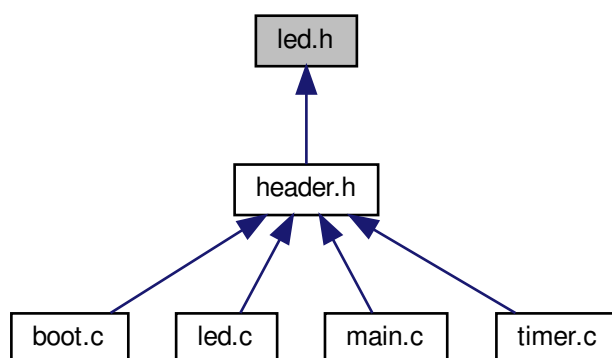
#### Returns

none.

## 7.804 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:



### Functions

- void [LedInit](#) (void)  
*Initializes the LED.*
- void [LedToggle](#) (void)  
*Toggles the LED at a fixed time interval.*

### 7.804.1 Detailed Description

LED driver header file.

### 7.804.2 Function Documentation

#### 7.804.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

##### Returns

none.

Initializes the LED.

##### Returns

none.

#### 7.804.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

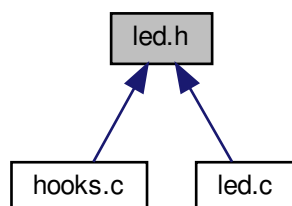
##### Returns

none.

### 7.805 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:



## Functions

- void [LedBlinkInit](#) ([blt\\_int16u](#) interval\_ms)  
*Initializes the LED blink driver.*
- void [LedBlinkTask](#) (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void [LedBlinkExit](#) (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

### 7.805.1 Detailed Description

LED driver header file.

### 7.805.2 Function Documentation

#### 7.805.2.1 LedBlinkExit()

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

#### Returns

none.

#### 7.805.2.2 LedBlinkInit()

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

#### Parameters

|                    |                                                                |
|--------------------|----------------------------------------------------------------|
| <i>interval_ms</i> | Specifies the desired LED blink interval time in milliseconds. |
|--------------------|----------------------------------------------------------------|

#### Returns

none.

### 7.805.2.3 LedBlinkTask()

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

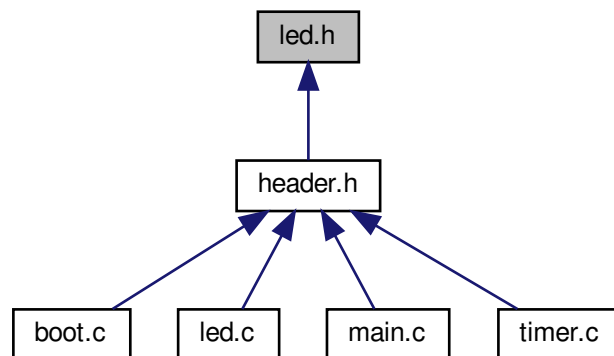
#### Returns

none.

## 7.806 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:



### Functions

- void [LedInit](#) (void)  
*Initializes the LED.*
- void [LedToggle](#) (void)  
*Toggles the LED at a fixed time interval.*

### 7.806.1 Detailed Description

LED driver header file.

### 7.806.2 Function Documentation

### 7.806.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

#### Returns

none.

Initializes the LED.

#### Returns

none.

### 7.806.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

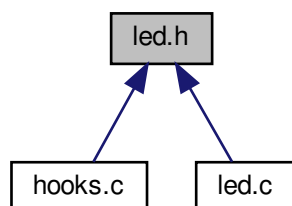
#### Returns

none.

## 7.807 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:



## Functions

- void [LedBlinkInit](#) ([blt\\_int16u](#) interval\_ms)  
*Initializes the LED blink driver.*
- void [LedBlinkTask](#) (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void [LedBlinkExit](#) (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

### 7.807.1 Detailed Description

LED driver header file.

### 7.807.2 Function Documentation

#### 7.807.2.1 LedBlinkExit()

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

#### Returns

none.

#### 7.807.2.2 LedBlinkInit()

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

#### Parameters

|                    |                                                                |
|--------------------|----------------------------------------------------------------|
| <i>interval_ms</i> | Specifies the desired LED blink interval time in milliseconds. |
|--------------------|----------------------------------------------------------------|

#### Returns

none.



### 7.807.2.3 LedBlinkTask()

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

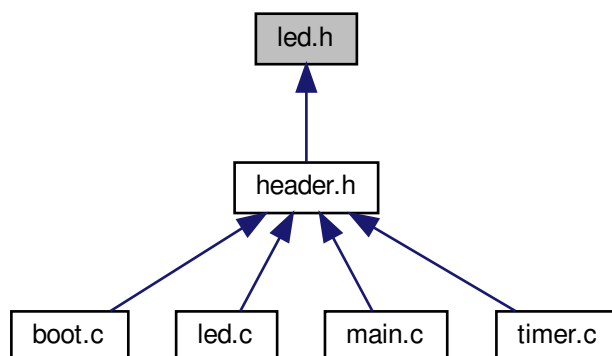
#### Returns

none.

## 7.808 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:



### Functions

- void [LedInit](#) (void)  
*Initializes the LED.*
- void [LedToggle](#) (void)  
*Toggles the LED at a fixed time interval.*

### 7.808.1 Detailed Description

LED driver header file.

### 7.808.2 Function Documentation

### 7.808.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

#### Returns

none.

Initializes the LED.

#### Returns

none.

### 7.808.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

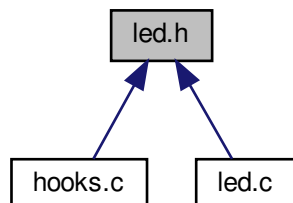
#### Returns

none.

## 7.809 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:



## Functions

- void [LedBlinkInit](#) ([blt\\_int16u](#) interval\_ms)  
*Initializes the LED blink driver.*
- void [LedBlinkTask](#) (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void [LedBlinkExit](#) (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

### 7.809.1 Detailed Description

LED driver header file.

### 7.809.2 Function Documentation

#### 7.809.2.1 LedBlinkExit()

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

#### Returns

none.

#### 7.809.2.2 LedBlinkInit()

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

#### Parameters

|                    |                                                                |
|--------------------|----------------------------------------------------------------|
| <i>interval_ms</i> | Specifies the desired LED blink interval time in milliseconds. |
|--------------------|----------------------------------------------------------------|

#### Returns

none.

### 7.809.2.3 LedBlinkTask()

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

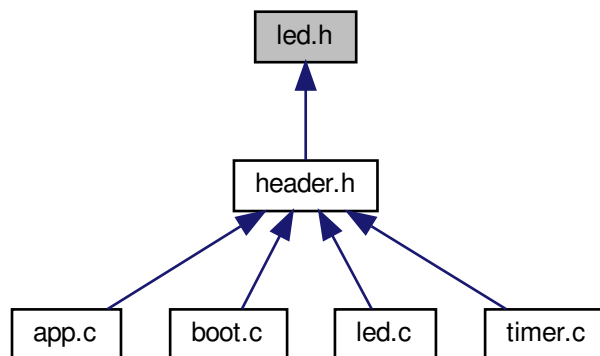
#### Returns

none.

## 7.810 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:



### Functions

- void [LedInit](#) (void)  
*Initializes the LED.*
- void [LedToggle](#) (void)  
*Toggles the LED at a fixed time interval.*

### 7.810.1 Detailed Description

LED driver header file.

### 7.810.2 Function Documentation

### 7.810.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

#### Returns

none.

Initializes the LED.

#### Returns

none.

### 7.810.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

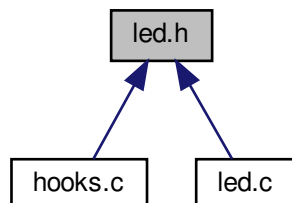
#### Returns

none.

## 7.811 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:



## Functions

- void [LedBlinkInit](#) ([blt\\_int16u](#) interval\_ms)  
*Initializes the LED blink driver.*
- void [LedBlinkTask](#) (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void [LedBlinkExit](#) (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

### 7.811.1 Detailed Description

LED driver header file.

### 7.811.2 Function Documentation

#### 7.811.2.1 LedBlinkExit()

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

#### Returns

none.

#### 7.811.2.2 LedBlinkInit()

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

#### Parameters

|                             |                                                                |
|-----------------------------|----------------------------------------------------------------|
| <a href="#">interval_ms</a> | Specifies the desired LED blink interval time in milliseconds. |
|-----------------------------|----------------------------------------------------------------|

#### Returns

none.

### 7.811.2.3 LedBlinkTask()

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

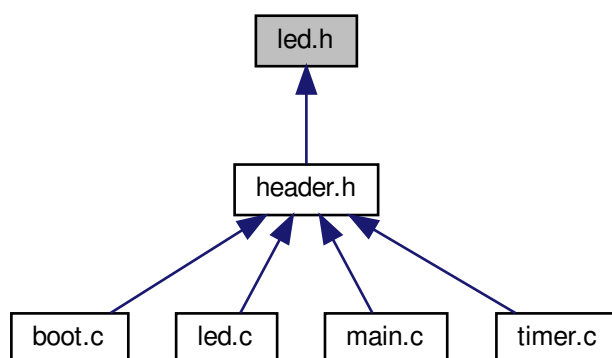
#### Returns

none.

## 7.812 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:



### Functions

- void [LedInit](#) (void)  
*Initializes the LED.*
- void [LedToggle](#) (void)  
*Toggles the LED at a fixed time interval.*

### 7.812.1 Detailed Description

LED driver header file.

### 7.812.2 Function Documentation

### 7.812.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

#### Returns

none.

Initializes the LED.

#### Returns

none.

### 7.812.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

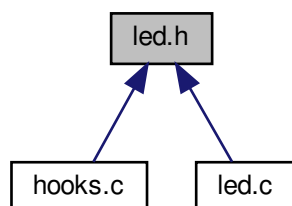
#### Returns

none.

## 7.813 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:





## Functions

- void [LedBlinkInit](#) ([blt\\_int16u](#) interval\_ms)  
*Initializes the LED blink driver.*
- void [LedBlinkTask](#) (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void [LedBlinkExit](#) (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

### 7.813.1 Detailed Description

LED driver header file.

### 7.813.2 Function Documentation

#### 7.813.2.1 LedBlinkExit()

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

#### Returns

none.

#### 7.813.2.2 LedBlinkInit()

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

#### Parameters

|                    |                                                                |
|--------------------|----------------------------------------------------------------|
| <i>interval_ms</i> | Specifies the desired LED blink interval time in milliseconds. |
|--------------------|----------------------------------------------------------------|

#### Returns

none.

### 7.813.2.3 LedBlinkTask()

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

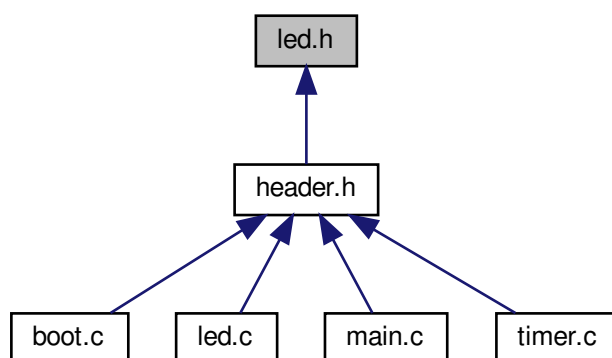
#### Returns

none.

## 7.814 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:



### Functions

- void [LedInit](#) (void)  
*Initializes the LED.*
- void [LedToggle](#) (void)  
*Toggles the LED at a fixed time interval.*

### 7.814.1 Detailed Description

LED driver header file.

### 7.814.2 Function Documentation

#### 7.814.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

##### Returns

none.

Initializes the LED.

##### Returns

none.

#### 7.814.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

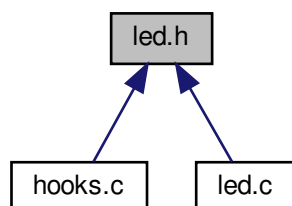
##### Returns

none.

## 7.815 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:



## Functions

- void [LedBlinkInit](#) ([blt\\_int16u](#) interval\_ms)  
*Initializes the LED blink driver.*
- void [LedBlinkTask](#) (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void [LedBlinkExit](#) (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

### 7.815.1 Detailed Description

LED driver header file.

### 7.815.2 Function Documentation

#### 7.815.2.1 LedBlinkExit()

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

#### Returns

none.

#### 7.815.2.2 LedBlinkInit()

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

#### Parameters

|                             |                                                                |
|-----------------------------|----------------------------------------------------------------|
| <a href="#">interval_ms</a> | Specifies the desired LED blink interval time in milliseconds. |
|-----------------------------|----------------------------------------------------------------|

#### Returns

none.

### 7.815.2.3 LedBlinkTask()

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

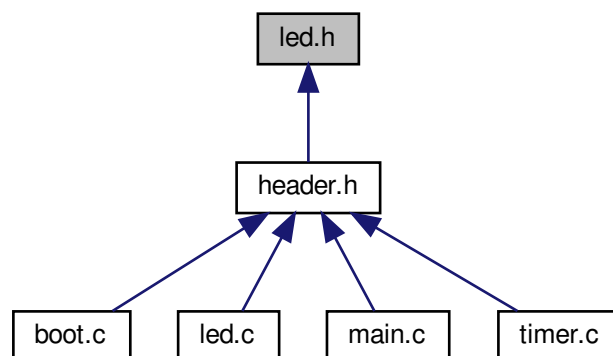
#### Returns

none.

## 7.816 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:



### Functions

- void [LedInit](#) (void)  
*Initializes the LED.*
- void [LedToggle](#) (void)  
*Toggles the LED at a fixed time interval.*

### 7.816.1 Detailed Description

LED driver header file.

### 7.816.2 Function Documentation

### 7.816.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

#### Returns

none.

Initializes the LED.

#### Returns

none.

### 7.816.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

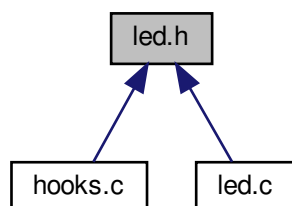
#### Returns

none.

## 7.817 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:



## Functions

- void [LedBlinkInit](#) ([blt\\_int16u](#) interval\_ms)  
*Initializes the LED blink driver.*
- void [LedBlinkTask](#) (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void [LedBlinkExit](#) (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

### 7.817.1 Detailed Description

LED driver header file.

### 7.817.2 Function Documentation

#### 7.817.2.1 LedBlinkExit()

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

#### Returns

none.

#### 7.817.2.2 LedBlinkInit()

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

#### Parameters

|                    |                                                                |
|--------------------|----------------------------------------------------------------|
| <i>interval_ms</i> | Specifies the desired LED blink interval time in milliseconds. |
|--------------------|----------------------------------------------------------------|

#### Returns

none.

### 7.817.2.3 LedBlinkTask()

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

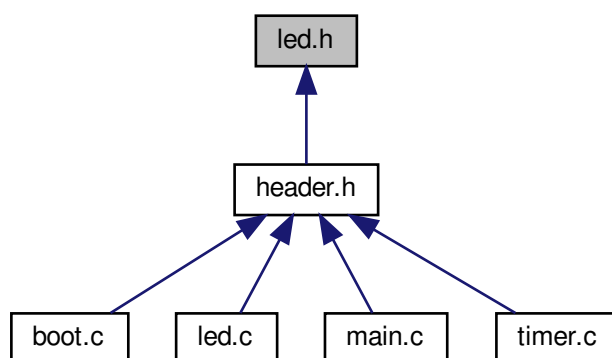
#### Returns

none.

## 7.818 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:



### Functions

- void [LedInit](#) (void)  
*Initializes the LED.*
- void [LedToggle](#) (void)  
*Toggles the LED at a fixed time interval.*

### 7.818.1 Detailed Description

LED driver header file.

### 7.818.2 Function Documentation



### 7.818.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

#### Returns

none.

Initializes the LED.

#### Returns

none.

### 7.818.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

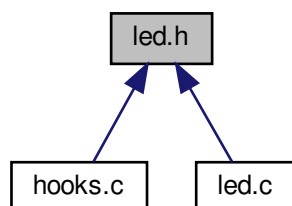
#### Returns

none.

## 7.819 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:



## Functions

- void [LedBlinkInit](#) ([blt\\_int16u](#) interval\_ms)  
*Initializes the LED blink driver.*
- void [LedBlinkTask](#) (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void [LedBlinkExit](#) (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

### 7.819.1 Detailed Description

LED driver header file.

### 7.819.2 Function Documentation

#### 7.819.2.1 LedBlinkExit()

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

#### Returns

none.

#### 7.819.2.2 LedBlinkInit()

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

#### Parameters

|                             |                                                                |
|-----------------------------|----------------------------------------------------------------|
| <a href="#">interval_ms</a> | Specifies the desired LED blink interval time in milliseconds. |
|-----------------------------|----------------------------------------------------------------|

#### Returns

none.

### 7.819.2.3 LedBlinkTask()

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

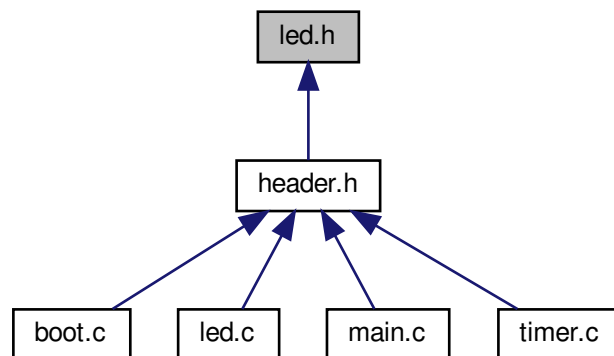
#### Returns

none.

## 7.820 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:



### Functions

- void [LedInit](#) (void)  
*Initializes the LED.*
- void [LedToggle](#) (void)  
*Toggles the LED at a fixed time interval.*

### 7.820.1 Detailed Description

LED driver header file.

### 7.820.2 Function Documentation

### 7.820.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

#### Returns

none.

Initializes the LED.

#### Returns

none.

### 7.820.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

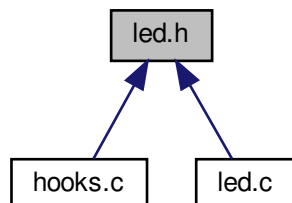
#### Returns

none.

## 7.821 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:



## Functions

- void [LedBlinkInit](#) ([blt\\_int16u](#) interval\_ms)  
*Initializes the LED blink driver.*
- void [LedBlinkTask](#) (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void [LedBlinkExit](#) (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

### 7.821.1 Detailed Description

LED driver header file.

### 7.821.2 Function Documentation

#### 7.821.2.1 LedBlinkExit()

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

#### Returns

none.

#### 7.821.2.2 LedBlinkInit()

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

#### Parameters

|                    |                                                                |
|--------------------|----------------------------------------------------------------|
| <i>interval_ms</i> | Specifies the desired LED blink interval time in milliseconds. |
|--------------------|----------------------------------------------------------------|

#### Returns

none.

### 7.821.2.3 LedBlinkTask()

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

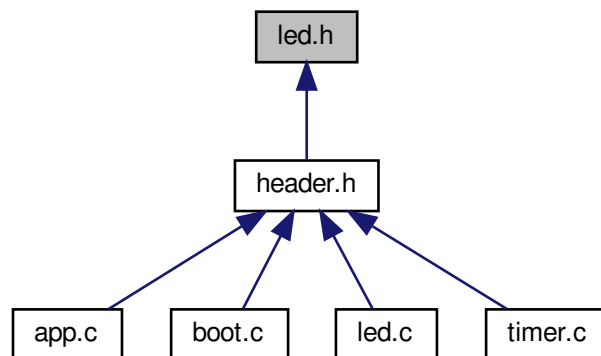
#### Returns

none.

## 7.822 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:



### Functions

- void [LedInit](#) (void)  
*Initializes the LED.*
- void [LedToggle](#) (void)  
*Toggles the LED at a fixed time interval.*

### 7.822.1 Detailed Description

LED driver header file.

### 7.822.2 Function Documentation

### 7.822.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

#### Returns

none.

Initializes the LED.

#### Returns

none.

### 7.822.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

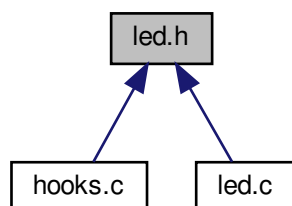
#### Returns

none.

## 7.823 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:



## Functions

- void [LedBlinkInit](#) ([blt\\_int16u](#) interval\_ms)  
*Initializes the LED blink driver.*
- void [LedBlinkTask](#) (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void [LedBlinkExit](#) (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

### 7.823.1 Detailed Description

LED driver header file.

### 7.823.2 Function Documentation

#### 7.823.2.1 LedBlinkExit()

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

#### Returns

none.

#### 7.823.2.2 LedBlinkInit()

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

#### Parameters

|                    |                                                                |
|--------------------|----------------------------------------------------------------|
| <i>interval_ms</i> | Specifies the desired LED blink interval time in milliseconds. |
|--------------------|----------------------------------------------------------------|

#### Returns

none.



### 7.823.2.3 LedBlinkTask()

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

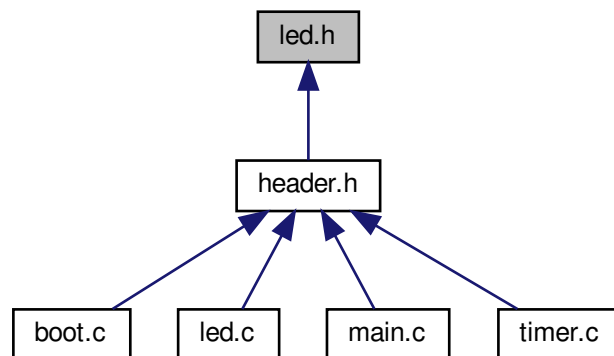
#### Returns

none.

## 7.824 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:



### Functions

- void [LedInit](#) (void)  
*Initializes the LED.*
- void [LedToggle](#) (void)  
*Toggles the LED at a fixed time interval.*

### 7.824.1 Detailed Description

LED driver header file.

### 7.824.2 Function Documentation

### 7.824.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

#### Returns

none.

Initializes the LED.

#### Returns

none.

### 7.824.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

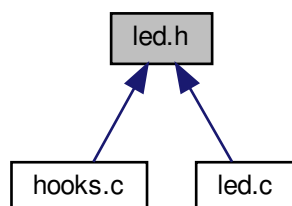
#### Returns

none.

## 7.825 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:



## Functions

- void [LedBlinkInit](#) ([blt\\_int16u](#) interval\_ms)  
*Initializes the LED blink driver.*
- void [LedBlinkTask](#) (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void [LedBlinkExit](#) (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

### 7.825.1 Detailed Description

LED driver header file.

### 7.825.2 Function Documentation

#### 7.825.2.1 LedBlinkExit()

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

#### Returns

none.

#### 7.825.2.2 LedBlinkInit()

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

#### Parameters

|                    |                                                                |
|--------------------|----------------------------------------------------------------|
| <i>interval_ms</i> | Specifies the desired LED blink interval time in milliseconds. |
|--------------------|----------------------------------------------------------------|

#### Returns

none.

### 7.825.2.3 LedBlinkTask()

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

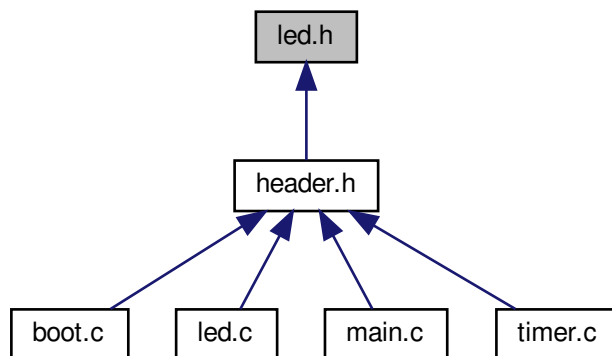
#### Returns

none.

## 7.826 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:



### Functions

- void [LedInit](#) (void)  
*Initializes the LED.*
- void [LedToggle](#) (void)  
*Toggles the LED at a fixed time interval.*

### 7.826.1 Detailed Description

LED driver header file.

### 7.826.2 Function Documentation

### 7.826.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

#### Returns

none.

Initializes the LED.

#### Returns

none.

### 7.826.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

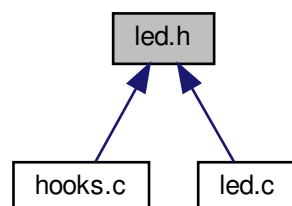
#### Returns

none.

## 7.827 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:



## Functions

- void [LedBlinkInit](#) ([blt\\_int16u](#) interval\_ms)  
*Initializes the LED blink driver.*
- void [LedBlinkTask](#) (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void [LedBlinkExit](#) (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

### 7.827.1 Detailed Description

LED driver header file.

### 7.827.2 Function Documentation

#### 7.827.2.1 LedBlinkExit()

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

#### Returns

none.

#### 7.827.2.2 LedBlinkInit()

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

#### Parameters

|                             |                                                                |
|-----------------------------|----------------------------------------------------------------|
| <a href="#">interval_ms</a> | Specifies the desired LED blink interval time in milliseconds. |
|-----------------------------|----------------------------------------------------------------|

#### Returns

none.

### 7.827.2.3 LedBlinkTask()

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

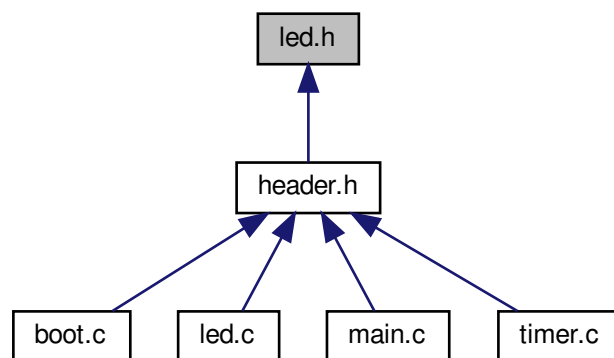
#### Returns

none.

## 7.828 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:



### Functions

- void [LedInit](#) (void)  
*Initializes the LED.*
- void [LedToggle](#) (void)  
*Toggles the LED at a fixed time interval.*

### 7.828.1 Detailed Description

LED driver header file.

### 7.828.2 Function Documentation

### 7.828.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

#### Returns

none.

Initializes the LED.

#### Returns

none.

### 7.828.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

#### Returns

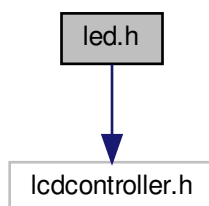
none.

## 7.829 led.h File Reference

LED driver header file.

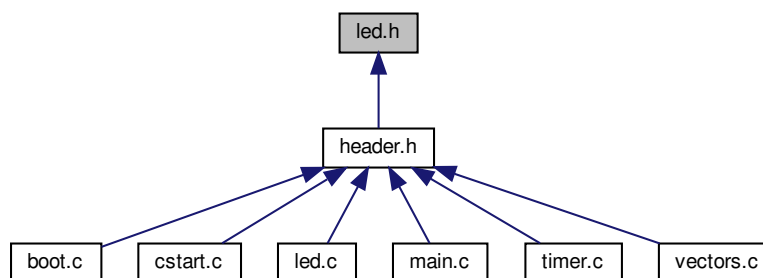
```
#include "lcdcontroller.h"
```

Include dependency graph for ARMCM3\_EFM32\_Olimex\_EM32G880F128STK\_GCC/Prog/led.h:





This graph shows which files directly or indirectly include this file:



## Functions

- void [LedInit](#) (void)  
*Initializes the LED.*
- void [LedToggle](#) (void)  
*Toggles the LED at a fixed time interval.*

### 7.829.1 Detailed Description

LED driver header file.

### 7.829.2 Function Documentation

#### 7.829.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

#### Returns

none.

Initializes the LED.

#### Returns

none.

### 7.829.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

#### Returns

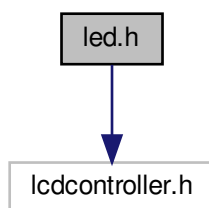
none.

## 7.830 led.h File Reference

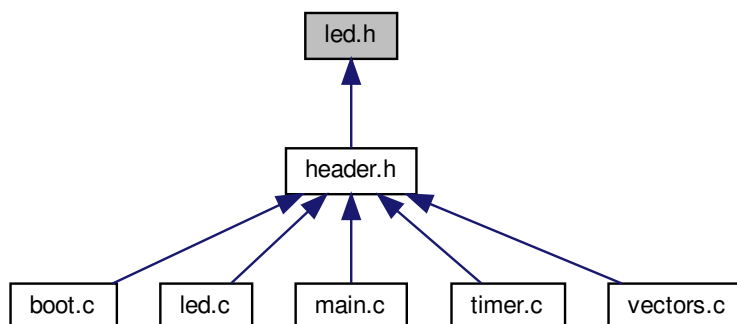
LED driver header file.

```
#include "lcdcontroller.h"
```

Include dependency graph for ARMC32\_EFM32\_Olimex\_EM32G880F128STK\_IAR/Prog/led.h:



This graph shows which files directly or indirectly include this file:



## Functions

- void [LedInit](#) (void)  
*Initializes the LED.*
- void [LedToggle](#) (void)  
*Toggles the LED at a fixed time interval.*

### 7.830.1 Detailed Description

LED driver header file.

### 7.830.2 Function Documentation

#### 7.830.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

##### Returns

none.

Initializes the LED.

##### Returns

none.

#### 7.830.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

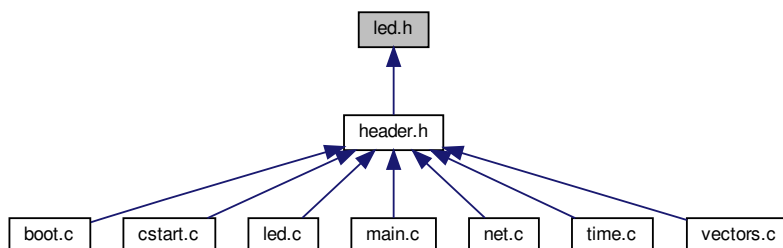
##### Returns

none.

## 7.831 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:



### Functions

- void [LedInit](#) (void)  
*Initializes the LED.*
- void [LedToggle](#) (void)  
*Toggles the LED at a fixed time interval.*

### 7.831.1 Detailed Description

LED driver header file.

### 7.831.2 Function Documentation

#### 7.831.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

#### Returns

none.

Initializes the LED.

#### Returns

none.

### 7.831.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

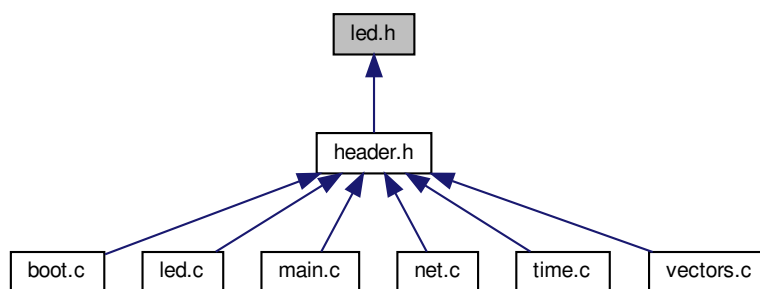
#### Returns

none.

## 7.832 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:



### Functions

- void [LedInit](#) (void)  
*Initializes the LED.*
- void [LedToggle](#) (void)  
*Toggles the LED at a fixed time interval.*

### 7.832.1 Detailed Description

LED driver header file.

### 7.832.2 Function Documentation

### 7.832.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

#### Returns

none.

Initializes the LED.

#### Returns

none.

### 7.832.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

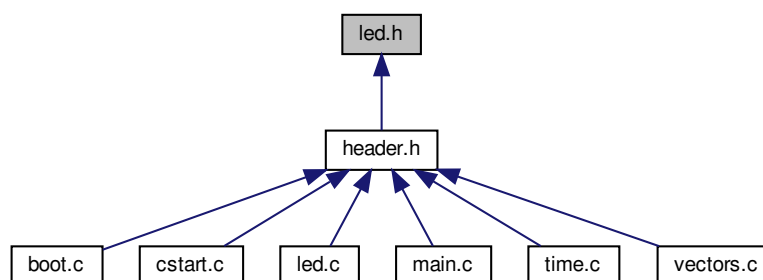
#### Returns

none.

## 7.833 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:



## Functions

- void [LedInit](#) (void)  
*Initializes the LED.*
- void [LedToggle](#) (void)  
*Toggles the LED at a fixed time interval.*

### 7.833.1 Detailed Description

LED driver header file.

### 7.833.2 Function Documentation

#### 7.833.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

##### Returns

none.

Initializes the LED.

##### Returns

none.

#### 7.833.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

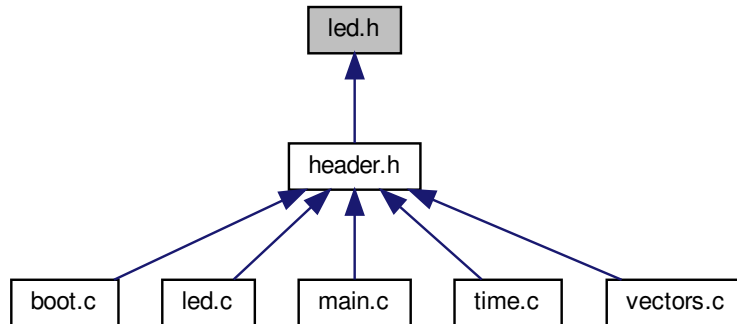
##### Returns

none.

## 7.834 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:



### Functions

- void [LedInit](#) (void)  
*Initializes the LED.*
- void [LedToggle](#) (void)  
*Toggles the LED at a fixed time interval.*

### 7.834.1 Detailed Description

LED driver header file.

### 7.834.2 Function Documentation

#### 7.834.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

**Returns**

none.

Initializes the LED.

**Returns**

none.



### 7.834.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

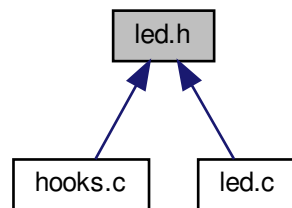
#### Returns

none.

## 7.835 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:



## Functions

- void [LedBlinkInit](#) ([blt\\_int16u](#) interval\_ms)  
*Initializes the LED blink driver.*
- void [LedBlinkTask](#) (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void [LedBlinkExit](#) (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

### 7.835.1 Detailed Description

LED driver header file.

### 7.835.2 Function Documentation

### 7.835.2.1 LedBlinkExit()

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

#### Returns

none.

### 7.835.2.2 LedBlinkInit()

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

#### Parameters

|                    |                                                                |
|--------------------|----------------------------------------------------------------|
| <i>interval_ms</i> | Specifies the desired LED blink interval time in milliseconds. |
|--------------------|----------------------------------------------------------------|

#### Returns

none.

### 7.835.2.3 LedBlinkTask()

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

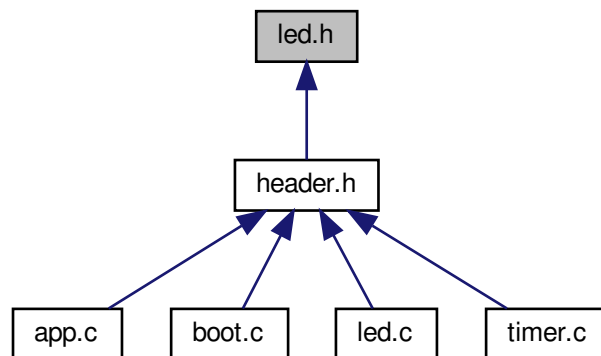
#### Returns

none.

## 7.836 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:



## Functions

- void [LedInit](#) (void)  
*Initializes the LED.*
- void [LedToggle](#) (void)  
*Toggles the LED at a fixed time interval.*

### 7.836.1 Detailed Description

LED driver header file.

### 7.836.2 Function Documentation

#### 7.836.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

#### Returns

none.

Initializes the LED.

#### Returns

none.

### 7.836.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

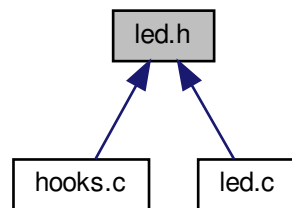
#### Returns

none.

## 7.837 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:



## Functions

- void [LedBlinkInit](#) ([blt\\_int16u](#) interval\_ms)  
*Initializes the LED blink driver.*
- void [LedBlinkTask](#) (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void [LedBlinkExit](#) (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

### 7.837.1 Detailed Description

LED driver header file.

### 7.837.2 Function Documentation

### 7.837.2.1 LedBlinkExit()

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

#### Returns

none.

### 7.837.2.2 LedBlinkInit()

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

#### Parameters

|                    |                                                                |
|--------------------|----------------------------------------------------------------|
| <i>interval_ms</i> | Specifies the desired LED blink interval time in milliseconds. |
|--------------------|----------------------------------------------------------------|

#### Returns

none.

### 7.837.2.3 LedBlinkTask()

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

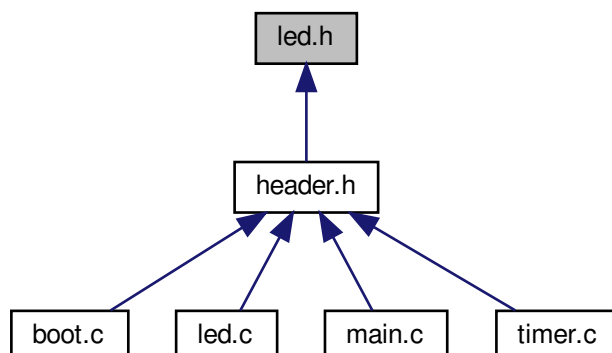
#### Returns

none.

## 7.838 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:



## Functions

- void [LedInit](#) (void)  
*Initializes the LED.*
- void [LedToggle](#) (void)  
*Toggles the LED at a fixed time interval.*

### 7.838.1 Detailed Description

LED driver header file.

### 7.838.2 Function Documentation

#### 7.838.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

#### Returns

none.

Initializes the LED.

#### Returns

none.

### 7.838.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

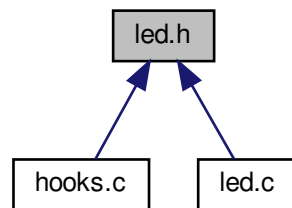
#### Returns

none.

## 7.839 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:



## Functions

- void [LedBlinkInit](#) ([blt\\_int16u](#) interval\_ms)  
*Initializes the LED blink driver.*
- void [LedBlinkTask](#) (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void [LedBlinkExit](#) (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

### 7.839.1 Detailed Description

LED driver header file.

### 7.839.2 Function Documentation

### 7.839.2.1 LedBlinkExit()

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

#### Returns

none.

### 7.839.2.2 LedBlinkInit()

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

#### Parameters

|                    |                                                                |
|--------------------|----------------------------------------------------------------|
| <i>interval_ms</i> | Specifies the desired LED blink interval time in milliseconds. |
|--------------------|----------------------------------------------------------------|

#### Returns

none.

### 7.839.2.3 LedBlinkTask()

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

#### Returns

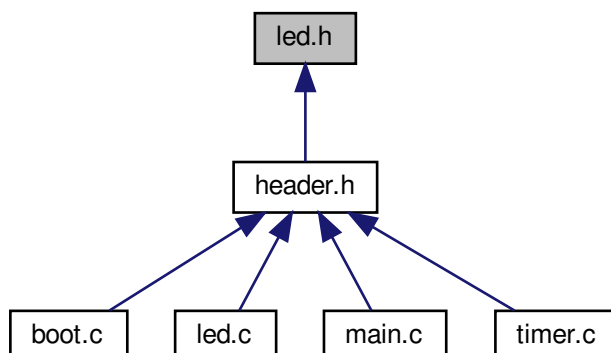
none.

## 7.840 led.h File Reference

LED driver header file.



This graph shows which files directly or indirectly include this file:



## Functions

- void [LedInit](#) (void)  
*Initializes the LED.*
- void [LedToggle](#) (void)  
*Toggles the LED at a fixed time interval.*

### 7.840.1 Detailed Description

LED driver header file.

### 7.840.2 Function Documentation

#### 7.840.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

#### Returns

none.

Initializes the LED.

#### Returns

none.

### 7.840.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

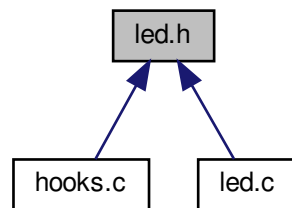
#### Returns

none.

## 7.841 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:



### Functions

- void [LedBlinkInit](#) ([blt\\_int16u](#) interval\_ms)  
*Initializes the LED blink driver.*
- void [LedBlinkTask](#) (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void [LedBlinkExit](#) (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

### 7.841.1 Detailed Description

LED driver header file.

### 7.841.2 Function Documentation

### 7.841.2.1 LedBlinkExit()

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

#### Returns

none.

### 7.841.2.2 LedBlinkInit()

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

#### Parameters

|                    |                                                                |
|--------------------|----------------------------------------------------------------|
| <i>interval_ms</i> | Specifies the desired LED blink interval time in milliseconds. |
|--------------------|----------------------------------------------------------------|

#### Returns

none.

### 7.841.2.3 LedBlinkTask()

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

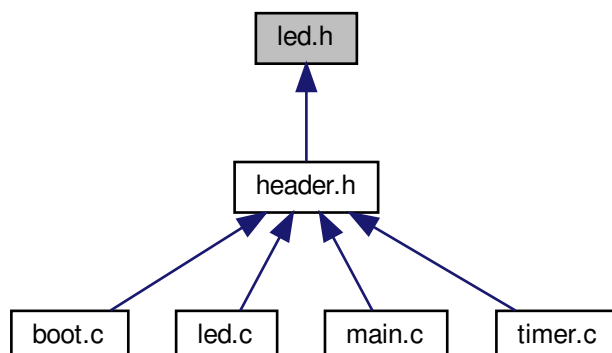
#### Returns

none.

## 7.842 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:



## Functions

- void [LedInit](#) (void)  
*Initializes the LED.*
- void [LedToggle](#) (void)  
*Toggles the LED at a fixed time interval.*

### 7.842.1 Detailed Description

LED driver header file.

### 7.842.2 Function Documentation

#### 7.842.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

#### Returns

none.

Initializes the LED.

#### Returns

none.

### 7.842.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

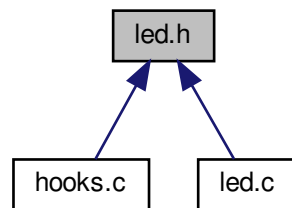
#### Returns

none.

## 7.843 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:



### Functions

- void [LedBlinkInit](#) ([blt\\_int16u](#) interval\_ms)  
*Initializes the LED blink driver.*
- void [LedBlinkTask](#) (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void [LedBlinkExit](#) (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

### 7.843.1 Detailed Description

LED driver header file.

### 7.843.2 Function Documentation

### 7.843.2.1 LedBlinkExit()

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

#### Returns

none.

### 7.843.2.2 LedBlinkInit()

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

#### Parameters

|                    |                                                                |
|--------------------|----------------------------------------------------------------|
| <i>interval_ms</i> | Specifies the desired LED blink interval time in milliseconds. |
|--------------------|----------------------------------------------------------------|

#### Returns

none.

### 7.843.2.3 LedBlinkTask()

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

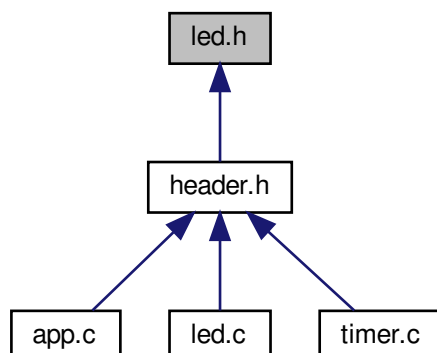
#### Returns

none.

## 7.844 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:



## Functions

- void [LedInit](#) (void)  
*Initializes the LED.*
- void [LedToggle](#) (void)  
*Toggles the LED at a fixed time interval.*

### 7.844.1 Detailed Description

LED driver header file.

### 7.844.2 Function Documentation

#### 7.844.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

#### Returns

none.

Initializes the LED.

#### Returns

none.

### 7.844.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

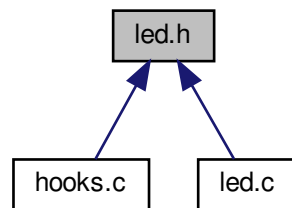
#### Returns

none.

## 7.845 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:



## Functions

- void [LedBlinkInit](#) ([blt\\_int16u](#) interval\_ms)  
*Initializes the LED blink driver.*
- void [LedBlinkTask](#) (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void [LedBlinkExit](#) (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

### 7.845.1 Detailed Description

LED driver header file.

### 7.845.2 Function Documentation



### 7.845.2.1 LedBlinkExit()

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

#### Returns

none.

### 7.845.2.2 LedBlinkInit()

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

#### Parameters

|                    |                                                                |
|--------------------|----------------------------------------------------------------|
| <i>interval_ms</i> | Specifies the desired LED blink interval time in milliseconds. |
|--------------------|----------------------------------------------------------------|

#### Returns

none.

### 7.845.2.3 LedBlinkTask()

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

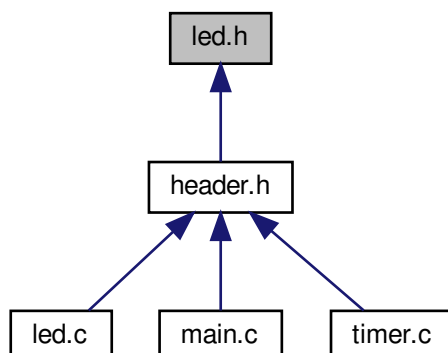
#### Returns

none.

## 7.846 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:



## Functions

- void [LedInit](#) (void)  
*Initializes the LED.*
- void [LedToggle](#) (void)  
*Toggles the LED at a fixed time interval.*

### 7.846.1 Detailed Description

LED driver header file.

### 7.846.2 Function Documentation

#### 7.846.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

#### Returns

none.

Initializes the LED.

#### Returns

none.

### 7.846.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

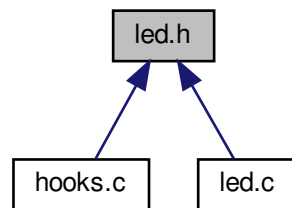
#### Returns

none.

## 7.847 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:



### Functions

- void [LedBlinkInit](#) ([blt\\_int16u](#) interval\_ms)  
*Initializes the LED blink driver.*
- void [LedBlinkTask](#) (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void [LedBlinkExit](#) (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

### 7.847.1 Detailed Description

LED driver header file.

### 7.847.2 Function Documentation

### 7.847.2.1 LedBlinkExit()

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

#### Returns

none.

### 7.847.2.2 LedBlinkInit()

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

#### Parameters

|                    |                                                                |
|--------------------|----------------------------------------------------------------|
| <i>interval_ms</i> | Specifies the desired LED blink interval time in milliseconds. |
|--------------------|----------------------------------------------------------------|

#### Returns

none.

### 7.847.2.3 LedBlinkTask()

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

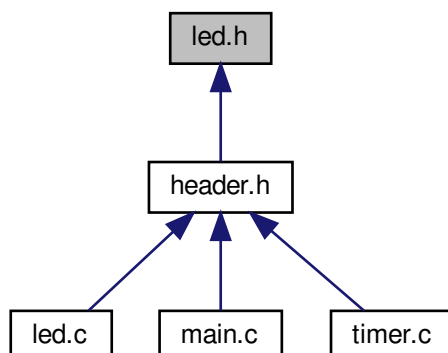
#### Returns

none.

## 7.848 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:



## Functions

- void [LedInit](#) (void)  
*Initializes the LED.*
- void [LedToggle](#) (void)  
*Toggles the LED at a fixed time interval.*

### 7.848.1 Detailed Description

LED driver header file.

### 7.848.2 Function Documentation

#### 7.848.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

#### Returns

none.

Initializes the LED.

#### Returns

none.

### 7.848.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

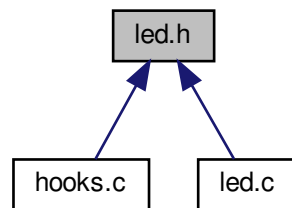
#### Returns

none.

## 7.849 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:



### Functions

- void [LedBlinkInit](#) ([blt\\_int16u](#) interval\_ms)  
*Initializes the LED blink driver.*
- void [LedBlinkTask](#) (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void [LedBlinkExit](#) (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

### 7.849.1 Detailed Description

LED driver header file.

### 7.849.2 Function Documentation

### 7.849.2.1 LedBlinkExit()

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

#### Returns

none.

### 7.849.2.2 LedBlinkInit()

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

#### Parameters

|                    |                                                                |
|--------------------|----------------------------------------------------------------|
| <i>interval_ms</i> | Specifies the desired LED blink interval time in milliseconds. |
|--------------------|----------------------------------------------------------------|

#### Returns

none.

### 7.849.2.3 LedBlinkTask()

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

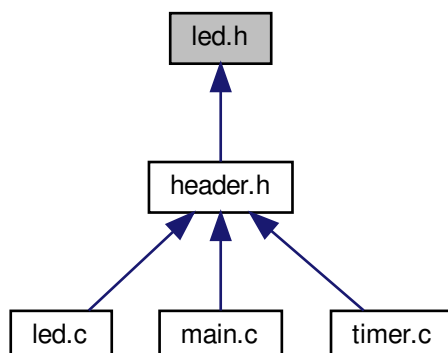
#### Returns

none.

## 7.850 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:



## Functions

- void [LedInit](#) (void)  
*Initializes the LED.*
- void [LedToggle](#) (void)  
*Toggles the LED at a fixed time interval.*

### 7.850.1 Detailed Description

LED driver header file.

### 7.850.2 Function Documentation

#### 7.850.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

#### Returns

none.

Initializes the LED.

#### Returns

none.



### 7.850.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

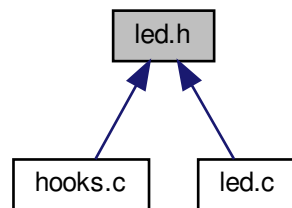
#### Returns

none.

## 7.851 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:



### Functions

- void [LedBlinkInit](#) ([blt\\_int16u](#) interval\_ms)  
*Initializes the LED blink driver.*
- void [LedBlinkTask](#) (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void [LedBlinkExit](#) (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

### 7.851.1 Detailed Description

LED driver header file.

### 7.851.2 Function Documentation

### 7.851.2.1 LedBlinkExit()

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

#### Returns

none.

### 7.851.2.2 LedBlinkInit()

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

#### Parameters

|                    |                                                                |
|--------------------|----------------------------------------------------------------|
| <i>interval_ms</i> | Specifies the desired LED blink interval time in milliseconds. |
|--------------------|----------------------------------------------------------------|

#### Returns

none.

### 7.851.2.3 LedBlinkTask()

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

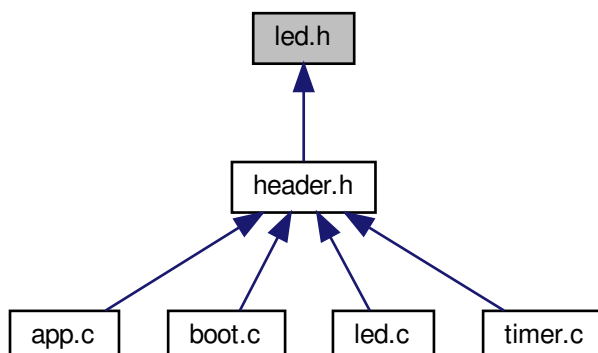
#### Returns

none.

## 7.852 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:



## Functions

- void [LedInit](#) (void)  
*Initializes the LED.*
- void [LedToggle](#) (void)  
*Toggles the LED at a fixed time interval.*

### 7.852.1 Detailed Description

LED driver header file.

### 7.852.2 Function Documentation

#### 7.852.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

#### Returns

none.

Initializes the LED.

#### Returns

none.

### 7.852.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

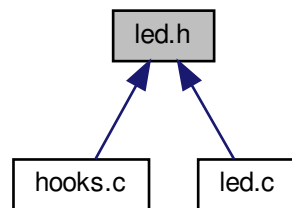
#### Returns

none.

## 7.853 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:



### Functions

- void [LedBlinkInit](#) ([blt\\_int16u](#) interval\_ms)  
*Initializes the LED blink driver.*
- void [LedBlinkTask](#) (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void [LedBlinkExit](#) (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

### 7.853.1 Detailed Description

LED driver header file.

### 7.853.2 Function Documentation

### 7.853.2.1 LedBlinkExit()

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

#### Returns

none.

### 7.853.2.2 LedBlinkInit()

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

#### Parameters

|                    |                                                                |
|--------------------|----------------------------------------------------------------|
| <i>interval_ms</i> | Specifies the desired LED blink interval time in milliseconds. |
|--------------------|----------------------------------------------------------------|

#### Returns

none.

### 7.853.2.3 LedBlinkTask()

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

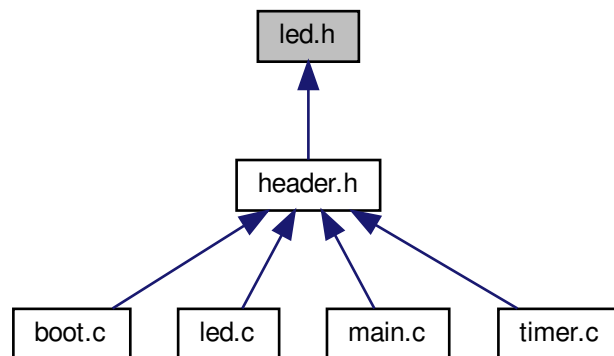
#### Returns

none.

## 7.854 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:



## Functions

- void [LedInit](#) (void)  
*Initializes the LED.*
- void [LedToggle](#) (void)  
*Toggles the LED at a fixed time interval.*

### 7.854.1 Detailed Description

LED driver header file.

### 7.854.2 Function Documentation

#### 7.854.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

#### Returns

none.

Initializes the LED.

#### Returns

none.

### 7.854.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

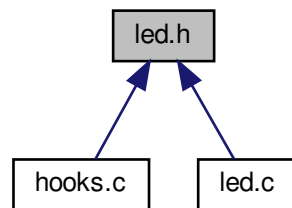
#### Returns

none.

## 7.855 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:



### Functions

- void [LedBlinkInit](#) ([blt\\_int16u](#) interval\_ms)  
*Initializes the LED blink driver.*
- void [LedBlinkTask](#) (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void [LedBlinkExit](#) (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

### 7.855.1 Detailed Description

LED driver header file.

### 7.855.2 Function Documentation

### 7.855.2.1 LedBlinkExit()

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

#### Returns

none.

### 7.855.2.2 LedBlinkInit()

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

#### Parameters

|                    |                                                                |
|--------------------|----------------------------------------------------------------|
| <i>interval_ms</i> | Specifies the desired LED blink interval time in milliseconds. |
|--------------------|----------------------------------------------------------------|

#### Returns

none.

### 7.855.2.3 LedBlinkTask()

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

#### Returns

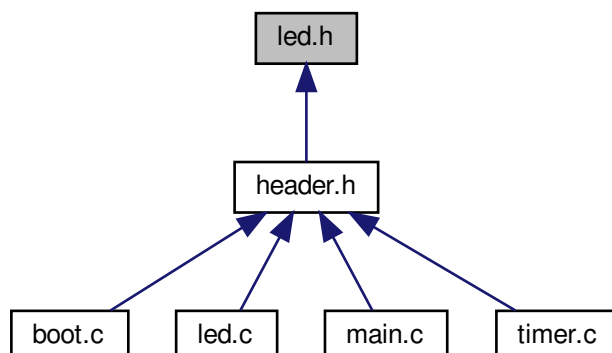
none.

## 7.856 led.h File Reference

LED driver header file.



This graph shows which files directly or indirectly include this file:



## Functions

- void [LedInit](#) (void)  
*Initializes the LED.*
- void [LedToggle](#) (void)  
*Toggles the LED at a fixed time interval.*

### 7.856.1 Detailed Description

LED driver header file.

### 7.856.2 Function Documentation

#### 7.856.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

#### Returns

none.

Initializes the LED.

#### Returns

none.

### 7.856.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

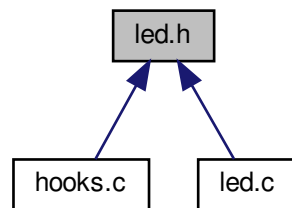
#### Returns

none.

## 7.857 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:



### Functions

- void [LedBlinkInit](#) ([blt\\_int16u](#) interval\_ms)  
*Initializes the LED blink driver.*
- void [LedBlinkTask](#) (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void [LedBlinkExit](#) (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

### 7.857.1 Detailed Description

LED driver header file.

### 7.857.2 Function Documentation

### 7.857.2.1 LedBlinkExit()

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

#### Returns

none.

### 7.857.2.2 LedBlinkInit()

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

#### Parameters

|                    |                                                                |
|--------------------|----------------------------------------------------------------|
| <i>interval_ms</i> | Specifies the desired LED blink interval time in milliseconds. |
|--------------------|----------------------------------------------------------------|

#### Returns

none.

### 7.857.2.3 LedBlinkTask()

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

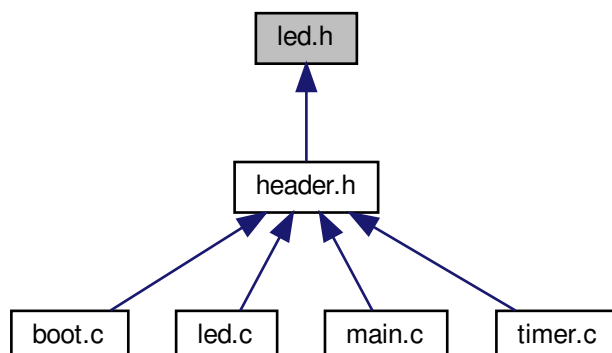
#### Returns

none.

## 7.858 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:



## Functions

- void [LedInit](#) (void)  
*Initializes the LED.*
- void [LedToggle](#) (void)  
*Toggles the LED at a fixed time interval.*

### 7.858.1 Detailed Description

LED driver header file.

### 7.858.2 Function Documentation

#### 7.858.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

#### Returns

none.

Initializes the LED.

#### Returns

none.

### 7.858.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

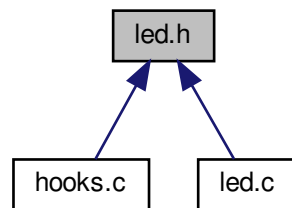
#### Returns

none.

## 7.859 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:



### Functions

- void [LedBlinkInit](#) ([blt\\_int16u](#) interval\_ms)  
*Initializes the LED blink driver.*
- void [LedBlinkTask](#) (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void [LedBlinkExit](#) (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

### 7.859.1 Detailed Description

LED driver header file.

### 7.859.2 Function Documentation

### 7.859.2.1 LedBlinkExit()

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

#### Returns

none.

### 7.859.2.2 LedBlinkInit()

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

#### Parameters

|                    |                                                                |
|--------------------|----------------------------------------------------------------|
| <i>interval_ms</i> | Specifies the desired LED blink interval time in milliseconds. |
|--------------------|----------------------------------------------------------------|

#### Returns

none.

### 7.859.2.3 LedBlinkTask()

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

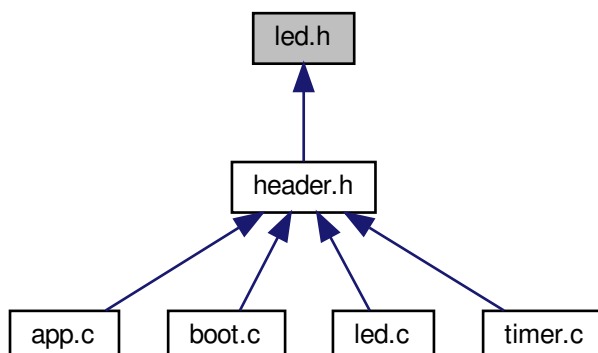
#### Returns

none.

## 7.860 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:



## Functions

- void [LedInit](#) (void)  
*Initializes the LED.*
- void [LedToggle](#) (void)  
*Toggles the LED at a fixed time interval.*

### 7.860.1 Detailed Description

LED driver header file.

### 7.860.2 Function Documentation

#### 7.860.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

#### Returns

none.

Initializes the LED.

#### Returns

none.

### 7.860.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

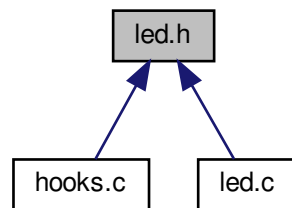
#### Returns

none.

## 7.861 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:



### Functions

- void [LedBlinkInit](#) ([blt\\_int16u](#) interval\_ms)  
*Initializes the LED blink driver.*
- void [LedBlinkTask](#) (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void [LedBlinkExit](#) (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

### 7.861.1 Detailed Description

LED driver header file.

### 7.861.2 Function Documentation



### 7.861.2.1 LedBlinkExit()

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

#### Returns

none.

### 7.861.2.2 LedBlinkInit()

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

#### Parameters

|                    |                                                                |
|--------------------|----------------------------------------------------------------|
| <i>interval_ms</i> | Specifies the desired LED blink interval time in milliseconds. |
|--------------------|----------------------------------------------------------------|

#### Returns

none.

### 7.861.2.3 LedBlinkTask()

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

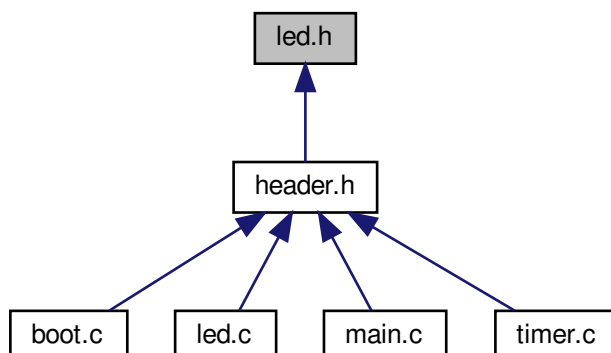
#### Returns

none.

## 7.862 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:



## Functions

- void [LedInit](#) (void)  
*Initializes the LED.*
- void [LedToggle](#) (void)  
*Toggles the LED at a fixed time interval.*

### 7.862.1 Detailed Description

LED driver header file.

### 7.862.2 Function Documentation

#### 7.862.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

#### Returns

none.

Initializes the LED.

#### Returns

none.

### 7.862.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

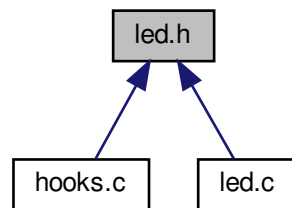
#### Returns

none.

## 7.863 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:



## Functions

- void [LedBlinkInit](#) ([blt\\_int16u](#) interval\_ms)  
*Initializes the LED blink driver.*
- void [LedBlinkTask](#) (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void [LedBlinkExit](#) (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

### 7.863.1 Detailed Description

LED driver header file.

### 7.863.2 Function Documentation

### 7.863.2.1 LedBlinkExit()

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

#### Returns

none.

### 7.863.2.2 LedBlinkInit()

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

#### Parameters

|                    |                                                                |
|--------------------|----------------------------------------------------------------|
| <i>interval_ms</i> | Specifies the desired LED blink interval time in milliseconds. |
|--------------------|----------------------------------------------------------------|

#### Returns

none.

### 7.863.2.3 LedBlinkTask()

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

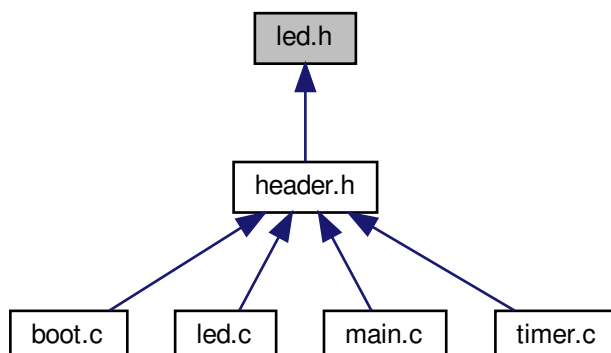
#### Returns

none.

## 7.864 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:



## Functions

- void [LedInit](#) (void)  
*Initializes the LED.*
- void [LedToggle](#) (void)  
*Toggles the LED at a fixed time interval.*

### 7.864.1 Detailed Description

LED driver header file.

### 7.864.2 Function Documentation

#### 7.864.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

#### Returns

none.

Initializes the LED.

#### Returns

none.

### 7.864.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

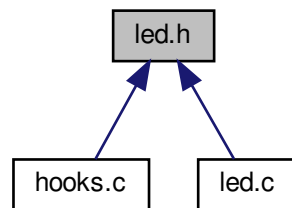
#### Returns

none.

## 7.865 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:



## Functions

- void [LedBlinkInit](#) ([blt\\_int16u](#) interval\_ms)  
*Initializes the LED blink driver.*
- void [LedBlinkTask](#) (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void [LedBlinkExit](#) (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

### 7.865.1 Detailed Description

LED driver header file.

### 7.865.2 Function Documentation

### 7.865.2.1 LedBlinkExit()

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

#### Returns

none.

### 7.865.2.2 LedBlinkInit()

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

#### Parameters

|                    |                                                                |
|--------------------|----------------------------------------------------------------|
| <i>interval_ms</i> | Specifies the desired LED blink interval time in milliseconds. |
|--------------------|----------------------------------------------------------------|

#### Returns

none.

### 7.865.2.3 LedBlinkTask()

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

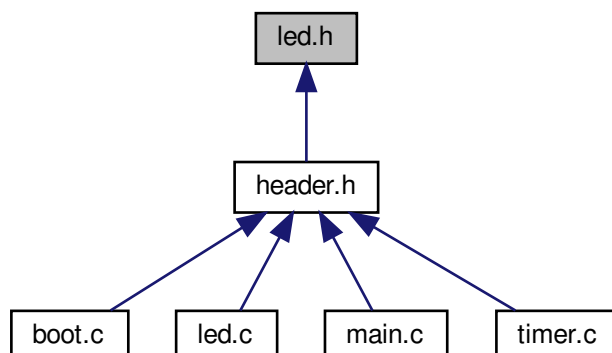
#### Returns

none.

## 7.866 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:



## Functions

- void [LedInit](#) (void)  
*Initializes the LED.*
- void [LedToggle](#) (void)  
*Toggles the LED at a fixed time interval.*

### 7.866.1 Detailed Description

LED driver header file.

### 7.866.2 Function Documentation

#### 7.866.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

#### Returns

none.

Initializes the LED.

#### Returns

none.



### 7.866.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

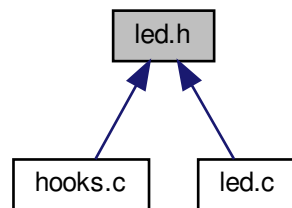
#### Returns

none.

## 7.867 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:



### Functions

- void [LedBlinkInit](#) ([blt\\_int16u](#) interval\_ms)  
*Initializes the LED blink driver.*
- void [LedBlinkTask](#) (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void [LedBlinkExit](#) (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

### 7.867.1 Detailed Description

LED driver header file.

### 7.867.2 Function Documentation

### 7.867.2.1 LedBlinkExit()

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

#### Returns

none.

### 7.867.2.2 LedBlinkInit()

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

#### Parameters

|                    |                                                                |
|--------------------|----------------------------------------------------------------|
| <i>interval_ms</i> | Specifies the desired LED blink interval time in milliseconds. |
|--------------------|----------------------------------------------------------------|

#### Returns

none.

### 7.867.2.3 LedBlinkTask()

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

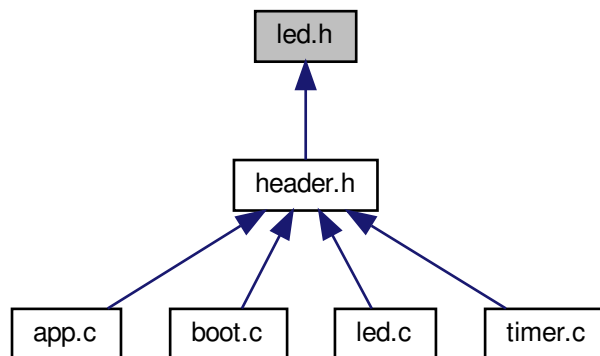
#### Returns

none.

## 7.868 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:



## Functions

- void [LedInit](#) (void)  
*Initializes the LED.*
- void [LedToggle](#) (void)  
*Toggles the LED at a fixed time interval.*

### 7.868.1 Detailed Description

LED driver header file.

### 7.868.2 Function Documentation

#### 7.868.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

#### Returns

none.

Initializes the LED.

#### Returns

none.

### 7.868.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

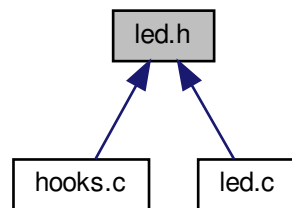
#### Returns

none.

## 7.869 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:



### Functions

- void [LedBlinkInit](#) ([blt\\_int16u](#) interval\_ms)  
*Initializes the LED blink driver.*
- void [LedBlinkTask](#) (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void [LedBlinkExit](#) (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

### 7.869.1 Detailed Description

LED driver header file.

### 7.869.2 Function Documentation

### 7.869.2.1 LedBlinkExit()

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

#### Returns

none.

### 7.869.2.2 LedBlinkInit()

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

#### Parameters

|                    |                                                                |
|--------------------|----------------------------------------------------------------|
| <i>interval_ms</i> | Specifies the desired LED blink interval time in milliseconds. |
|--------------------|----------------------------------------------------------------|

#### Returns

none.

### 7.869.2.3 LedBlinkTask()

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

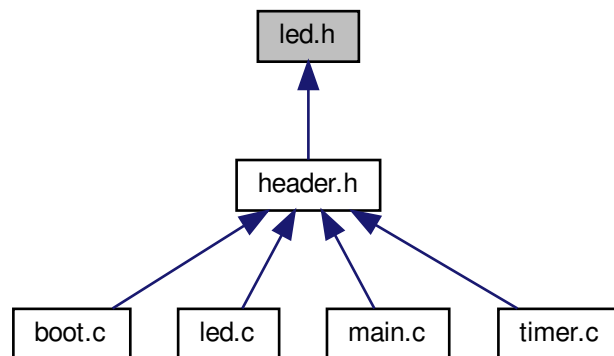
#### Returns

none.

## 7.870 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:



## Functions

- void [LedInit](#) (void)  
*Initializes the LED.*
- void [LedToggle](#) (void)  
*Toggles the LED at a fixed time interval.*

### 7.870.1 Detailed Description

LED driver header file.

### 7.870.2 Function Documentation

#### 7.870.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

#### Returns

none.

Initializes the LED.

#### Returns

none.

### 7.870.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

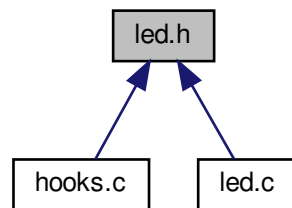
#### Returns

none.

## 7.871 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:



### Functions

- void [LedBlinkInit](#) ([blt\\_int16u](#) interval\_ms)  
*Initializes the LED blink driver.*
- void [LedBlinkTask](#) (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void [LedBlinkExit](#) (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

### 7.871.1 Detailed Description

LED driver header file.

### 7.871.2 Function Documentation

### 7.871.2.1 LedBlinkExit()

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

#### Returns

none.

### 7.871.2.2 LedBlinkInit()

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

#### Parameters

|                    |                                                                |
|--------------------|----------------------------------------------------------------|
| <i>interval_ms</i> | Specifies the desired LED blink interval time in milliseconds. |
|--------------------|----------------------------------------------------------------|

#### Returns

none.

### 7.871.2.3 LedBlinkTask()

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

#### Returns

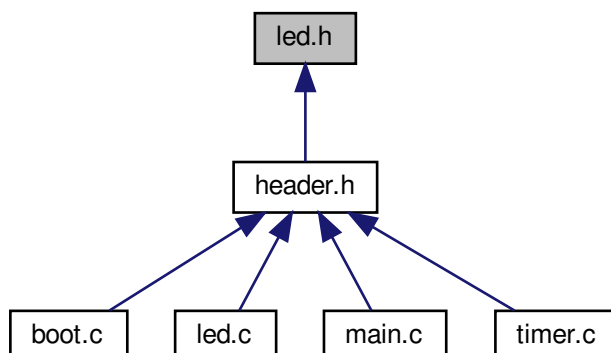
none.

## 7.872 led.h File Reference

LED driver header file.



This graph shows which files directly or indirectly include this file:



## Functions

- void [LedInit](#) (void)  
*Initializes the LED.*
- void [LedToggle](#) (void)  
*Toggles the LED at a fixed time interval.*

### 7.872.1 Detailed Description

LED driver header file.

### 7.872.2 Function Documentation

#### 7.872.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

#### Returns

none.

Initializes the LED.

#### Returns

none.

### 7.872.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

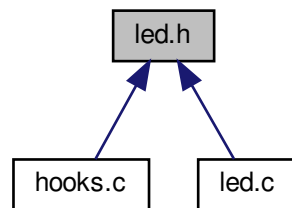
#### Returns

none.

## 7.873 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:



### Functions

- void [LedBlinkInit](#) ([blt\\_int16u](#) interval\_ms)  
*Initializes the LED blink driver.*
- void [LedBlinkTask](#) (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void [LedBlinkExit](#) (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

### 7.873.1 Detailed Description

LED driver header file.

### 7.873.2 Function Documentation

### 7.873.2.1 LedBlinkExit()

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

#### Returns

none.

### 7.873.2.2 LedBlinkInit()

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

#### Parameters

|                    |                                                                |
|--------------------|----------------------------------------------------------------|
| <i>interval_ms</i> | Specifies the desired LED blink interval time in milliseconds. |
|--------------------|----------------------------------------------------------------|

#### Returns

none.

### 7.873.2.3 LedBlinkTask()

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

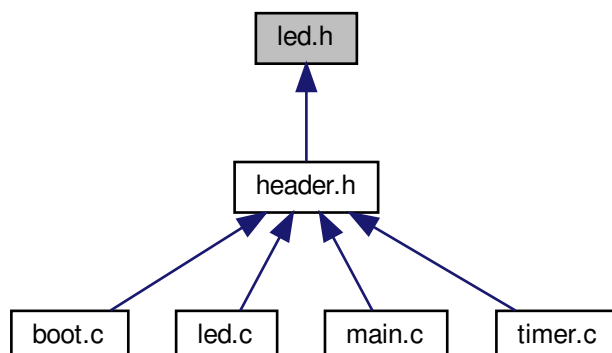
#### Returns

none.

## 7.874 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:



## Functions

- void [LedInit](#) (void)  
*Initializes the LED.*
- void [LedToggle](#) (void)  
*Toggles the LED at a fixed time interval.*

### 7.874.1 Detailed Description

LED driver header file.

### 7.874.2 Function Documentation

#### 7.874.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

#### Returns

none.

Initializes the LED.

#### Returns

none.

### 7.874.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

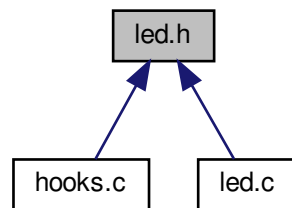
#### Returns

none.

## 7.875 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:



## Functions

- void [LedBlinkInit](#) ([blt\\_int16u](#) interval\_ms)  
*Initializes the LED blink driver.*
- void [LedBlinkTask](#) (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void [LedBlinkExit](#) (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

### 7.875.1 Detailed Description

LED driver header file.

### 7.875.2 Function Documentation

### 7.875.2.1 LedBlinkExit()

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

#### Returns

none.

### 7.875.2.2 LedBlinkInit()

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

#### Parameters

|                    |                                                                |
|--------------------|----------------------------------------------------------------|
| <i>interval_ms</i> | Specifies the desired LED blink interval time in milliseconds. |
|--------------------|----------------------------------------------------------------|

#### Returns

none.

### 7.875.2.3 LedBlinkTask()

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

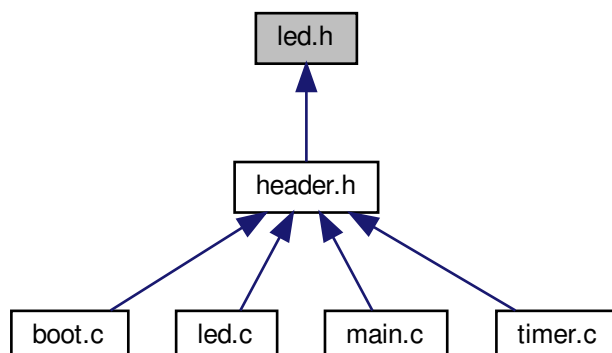
#### Returns

none.

## 7.876 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:



## Functions

- void [LedInit](#) (void)  
*Initializes the LED.*
- void [LedToggle](#) (void)  
*Toggles the LED at a fixed time interval.*

### 7.876.1 Detailed Description

LED driver header file.

### 7.876.2 Function Documentation

#### 7.876.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

#### Returns

none.

Initializes the LED.

#### Returns

none.

### 7.876.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

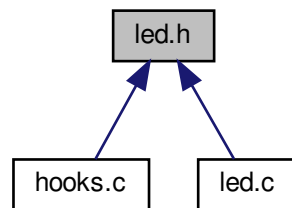
#### Returns

none.

## 7.877 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:



## Functions

- void [LedBlinkInit](#) ([blt\\_int16u](#) interval\_ms)  
*Initializes the LED blink driver.*
- void [LedBlinkTask](#) (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void [LedBlinkExit](#) (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

### 7.877.1 Detailed Description

LED driver header file.

### 7.877.2 Function Documentation



### 7.877.2.1 LedBlinkExit()

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

#### Returns

none.

### 7.877.2.2 LedBlinkInit()

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

#### Parameters

|                    |                                                                |
|--------------------|----------------------------------------------------------------|
| <i>interval_ms</i> | Specifies the desired LED blink interval time in milliseconds. |
|--------------------|----------------------------------------------------------------|

#### Returns

none.

### 7.877.2.3 LedBlinkTask()

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

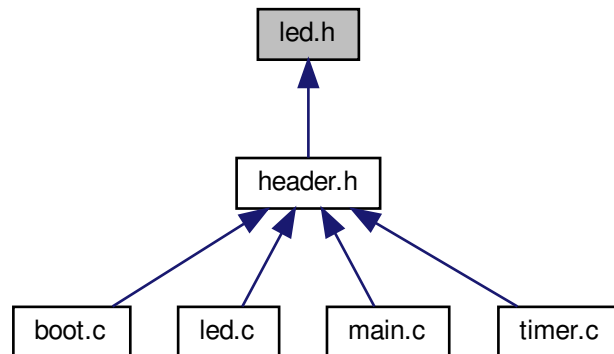
#### Returns

none.

## 7.878 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:



## Functions

- void [LedInit](#) (void)  
*Initializes the LED.*
- void [LedToggle](#) (void)  
*Toggles the LED at a fixed time interval.*

### 7.878.1 Detailed Description

LED driver header file.

### 7.878.2 Function Documentation

#### 7.878.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

#### Returns

none.

Initializes the LED.

#### Returns

none.

### 7.878.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

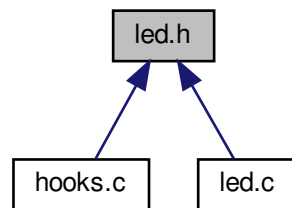
#### Returns

none.

## 7.879 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:



### Functions

- void [LedBlinkInit](#) ([blt\\_int16u](#) interval\_ms)  
*Initializes the LED blink driver.*
- void [LedBlinkTask](#) (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void [LedBlinkExit](#) (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

### 7.879.1 Detailed Description

LED driver header file.

### 7.879.2 Function Documentation

### 7.879.2.1 LedBlinkExit()

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

#### Returns

none.

### 7.879.2.2 LedBlinkInit()

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

#### Parameters

|                    |                                                                |
|--------------------|----------------------------------------------------------------|
| <i>interval_ms</i> | Specifies the desired LED blink interval time in milliseconds. |
|--------------------|----------------------------------------------------------------|

#### Returns

none.

### 7.879.2.3 LedBlinkTask()

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

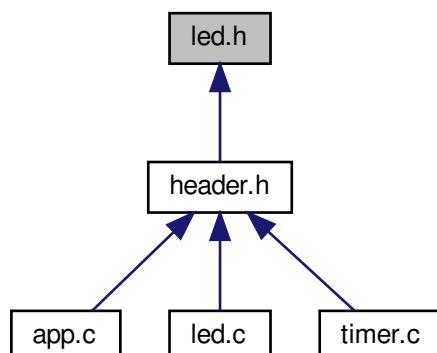
#### Returns

none.

## 7.880 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:



## Functions

- void [LedInit](#) (void)  
*Initializes the LED.*
- void [LedToggle](#) (void)  
*Toggles the LED at a fixed time interval.*

### 7.880.1 Detailed Description

LED driver header file.

### 7.880.2 Function Documentation

#### 7.880.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

#### Returns

none.

Initializes the LED.

#### Returns

none.

### 7.880.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

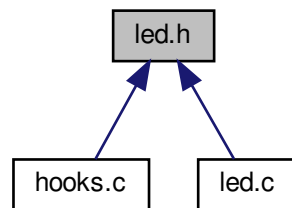
#### Returns

none.

## 7.881 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:



### Functions

- void [LedBlinkInit](#) ([blt\\_int16u](#) interval\_ms)  
*Initializes the LED blink driver.*
- void [LedBlinkTask](#) (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void [LedBlinkExit](#) (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

### 7.881.1 Detailed Description

LED driver header file.

### 7.881.2 Function Documentation

### 7.881.2.1 LedBlinkExit()

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

#### Returns

none.

### 7.881.2.2 LedBlinkInit()

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

#### Parameters

|                    |                                                                |
|--------------------|----------------------------------------------------------------|
| <i>interval_ms</i> | Specifies the desired LED blink interval time in milliseconds. |
|--------------------|----------------------------------------------------------------|

#### Returns

none.

### 7.881.2.3 LedBlinkTask()

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

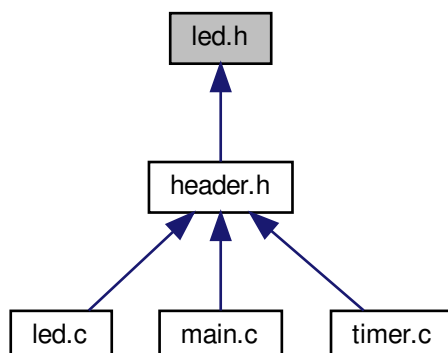
#### Returns

none.

## 7.882 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:



## Functions

- void [LedInit](#) (void)  
*Initializes the LED.*
- void [LedToggle](#) (void)  
*Toggles the LED at a fixed time interval.*

### 7.882.1 Detailed Description

LED driver header file.

### 7.882.2 Function Documentation

#### 7.882.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

#### Returns

none.

Initializes the LED.

#### Returns

none.



### 7.882.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

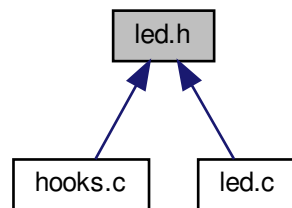
#### Returns

none.

## 7.883 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:



### Functions

- void [LedBlinkInit](#) ([blt\\_int16u](#) interval\_ms)  
*Initializes the LED blink driver.*
- void [LedBlinkTask](#) (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void [LedBlinkExit](#) (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

### 7.883.1 Detailed Description

LED driver header file.

### 7.883.2 Function Documentation

### 7.883.2.1 LedBlinkExit()

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

#### Returns

none.

### 7.883.2.2 LedBlinkInit()

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

#### Parameters

|                    |                                                                |
|--------------------|----------------------------------------------------------------|
| <i>interval_ms</i> | Specifies the desired LED blink interval time in milliseconds. |
|--------------------|----------------------------------------------------------------|

#### Returns

none.

### 7.883.2.3 LedBlinkTask()

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

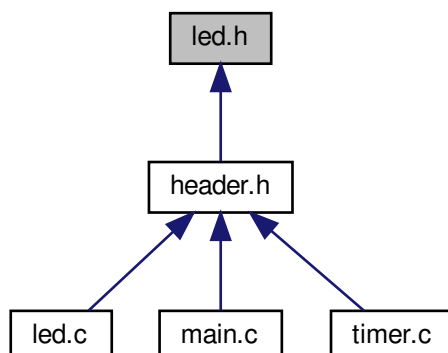
#### Returns

none.

## 7.884 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:



## Functions

- void [LedInit](#) (void)  
*Initializes the LED.*
- void [LedToggle](#) (void)  
*Toggles the LED at a fixed time interval.*

### 7.884.1 Detailed Description

LED driver header file.

### 7.884.2 Function Documentation

#### 7.884.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

#### Returns

none.

Initializes the LED.

#### Returns

none.

### 7.884.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

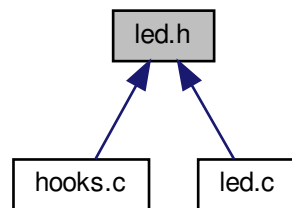
#### Returns

none.

## 7.885 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:



## Functions

- void [LedBlinkInit](#) ([blt\\_int16u](#) interval\_ms)  
*Initializes the LED blink driver.*
- void [LedBlinkTask](#) (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void [LedBlinkExit](#) (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

### 7.885.1 Detailed Description

LED driver header file.

### 7.885.2 Function Documentation

### 7.885.2.1 LedBlinkExit()

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

#### Returns

none.

### 7.885.2.2 LedBlinkInit()

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

#### Parameters

|                    |                                                                |
|--------------------|----------------------------------------------------------------|
| <i>interval_ms</i> | Specifies the desired LED blink interval time in milliseconds. |
|--------------------|----------------------------------------------------------------|

#### Returns

none.

### 7.885.2.3 LedBlinkTask()

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

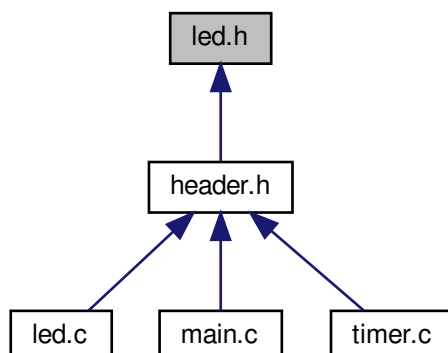
#### Returns

none.

## 7.886 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:



## Functions

- void [LedInit](#) (void)  
*Initializes the LED.*
- void [LedToggle](#) (void)  
*Toggles the LED at a fixed time interval.*

### 7.886.1 Detailed Description

LED driver header file.

### 7.886.2 Function Documentation

#### 7.886.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

#### Returns

none.

Initializes the LED.

#### Returns

none.

### 7.886.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

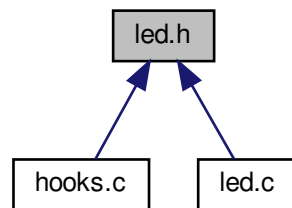
#### Returns

none.

## 7.887 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:



### Functions

- void [LedBlinkInit](#) ([blt\\_int16u](#) interval\_ms)  
*Initializes the LED blink driver.*
- void [LedBlinkTask](#) (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void [LedBlinkExit](#) (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

### 7.887.1 Detailed Description

LED driver header file.

### 7.887.2 Function Documentation

### 7.887.2.1 LedBlinkExit()

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

#### Returns

none.

### 7.887.2.2 LedBlinkInit()

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

#### Parameters

|                    |                                                                |
|--------------------|----------------------------------------------------------------|
| <i>interval_ms</i> | Specifies the desired LED blink interval time in milliseconds. |
|--------------------|----------------------------------------------------------------|

#### Returns

none.

### 7.887.2.3 LedBlinkTask()

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

#### Returns

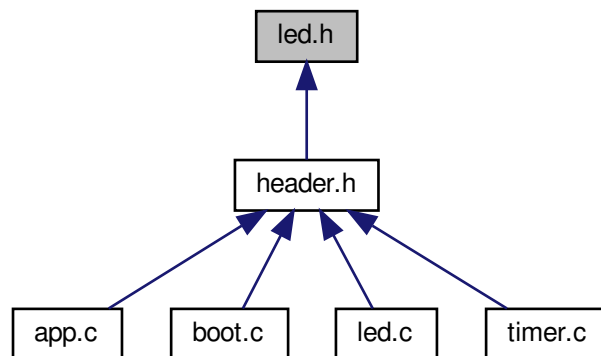
none.

## 7.888 led.h File Reference

LED driver header file.



This graph shows which files directly or indirectly include this file:



## Functions

- void [LedInit](#) (void)  
*Initializes the LED.*
- void [LedToggle](#) (void)  
*Toggles the LED at a fixed time interval.*

### 7.888.1 Detailed Description

LED driver header file.

### 7.888.2 Function Documentation

#### 7.888.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

#### Returns

none.

Initializes the LED.

#### Returns

none.

### 7.888.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

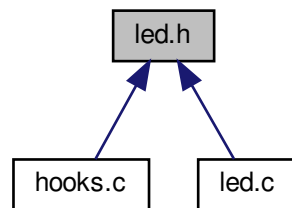
#### Returns

none.

## 7.889 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:



## Functions

- void [LedBlinkInit](#) ([blt\\_int16u](#) interval\_ms)  
*Initializes the LED blink driver.*
- void [LedBlinkTask](#) (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void [LedBlinkExit](#) (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

### 7.889.1 Detailed Description

LED driver header file.

### 7.889.2 Function Documentation

### 7.889.2.1 LedBlinkExit()

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

#### Returns

none.

### 7.889.2.2 LedBlinkInit()

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

#### Parameters

|                    |                                                                |
|--------------------|----------------------------------------------------------------|
| <i>interval_ms</i> | Specifies the desired LED blink interval time in milliseconds. |
|--------------------|----------------------------------------------------------------|

#### Returns

none.

### 7.889.2.3 LedBlinkTask()

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

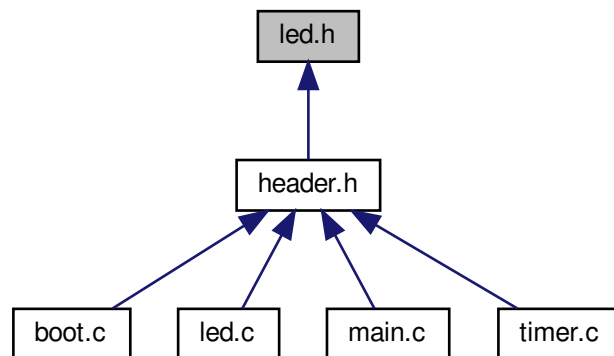
#### Returns

none.

## 7.890 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:



## Functions

- void [LedInit](#) (void)  
*Initializes the LED.*
- void [LedToggle](#) (void)  
*Toggles the LED at a fixed time interval.*

### 7.890.1 Detailed Description

LED driver header file.

### 7.890.2 Function Documentation

#### 7.890.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

#### Returns

none.

Initializes the LED.

#### Returns

none.

### 7.890.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

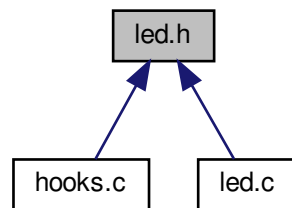
#### Returns

none.

## 7.891 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:



### Functions

- void [LedBlinkInit](#) ([blt\\_int16u](#) interval\_ms)  
*Initializes the LED blink driver.*
- void [LedBlinkTask](#) (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void [LedBlinkExit](#) (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

### 7.891.1 Detailed Description

LED driver header file.

### 7.891.2 Function Documentation

### 7.891.2.1 LedBlinkExit()

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

#### Returns

none.

### 7.891.2.2 LedBlinkInit()

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

#### Parameters

|                    |                                                                |
|--------------------|----------------------------------------------------------------|
| <i>interval_ms</i> | Specifies the desired LED blink interval time in milliseconds. |
|--------------------|----------------------------------------------------------------|

#### Returns

none.

### 7.891.2.3 LedBlinkTask()

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

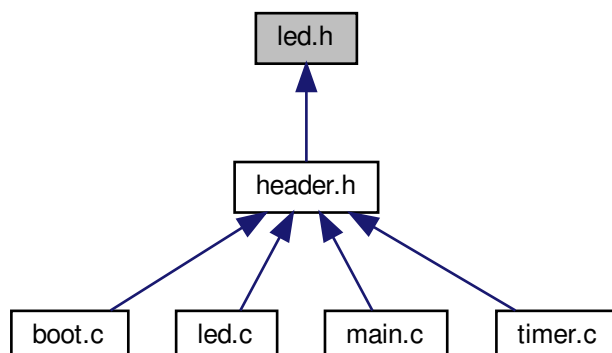
#### Returns

none.

## 7.892 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:



## Functions

- void [LedInit](#) (void)  
*Initializes the LED.*
- void [LedToggle](#) (void)  
*Toggles the LED at a fixed time interval.*

### 7.892.1 Detailed Description

LED driver header file.

### 7.892.2 Function Documentation

#### 7.892.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

#### Returns

none.

Initializes the LED.

#### Returns

none.

### 7.892.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

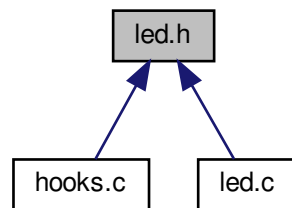
#### Returns

none.

## 7.893 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:



## Functions

- void [LedBlinkInit](#) ([blt\\_int16u](#) interval\_ms)  
*Initializes the LED blink driver.*
- void [LedBlinkTask](#) (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void [LedBlinkExit](#) (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

### 7.893.1 Detailed Description

LED driver header file.

### 7.893.2 Function Documentation



### 7.893.2.1 LedBlinkExit()

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

#### Returns

none.

### 7.893.2.2 LedBlinkInit()

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

#### Parameters

|                    |                                                                |
|--------------------|----------------------------------------------------------------|
| <i>interval_ms</i> | Specifies the desired LED blink interval time in milliseconds. |
|--------------------|----------------------------------------------------------------|

#### Returns

none.

### 7.893.2.3 LedBlinkTask()

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

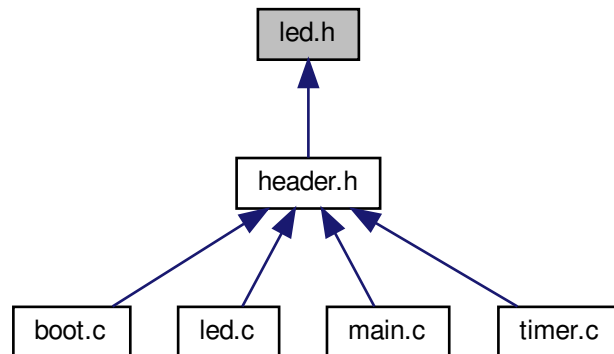
#### Returns

none.

## 7.894 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:



## Functions

- void [LedInit](#) (void)  
*Initializes the LED.*
- void [LedToggle](#) (void)  
*Toggles the LED at a fixed time interval.*

### 7.894.1 Detailed Description

LED driver header file.

### 7.894.2 Function Documentation

#### 7.894.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

#### Returns

none.

Initializes the LED.

#### Returns

none.

### 7.894.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

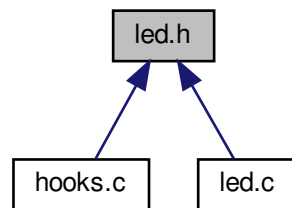
#### Returns

none.

## 7.895 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:



### Functions

- void [LedBlinkInit](#) ([blt\\_int16u](#) interval\_ms)  
*Initializes the LED blink driver.*
- void [LedBlinkTask](#) (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void [LedBlinkExit](#) (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

### 7.895.1 Detailed Description

LED driver header file.

### 7.895.2 Function Documentation

### 7.895.2.1 LedBlinkExit()

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

#### Returns

none.

### 7.895.2.2 LedBlinkInit()

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

#### Parameters

|                    |                                                                |
|--------------------|----------------------------------------------------------------|
| <i>interval_ms</i> | Specifies the desired LED blink interval time in milliseconds. |
|--------------------|----------------------------------------------------------------|

#### Returns

none.

### 7.895.2.3 LedBlinkTask()

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

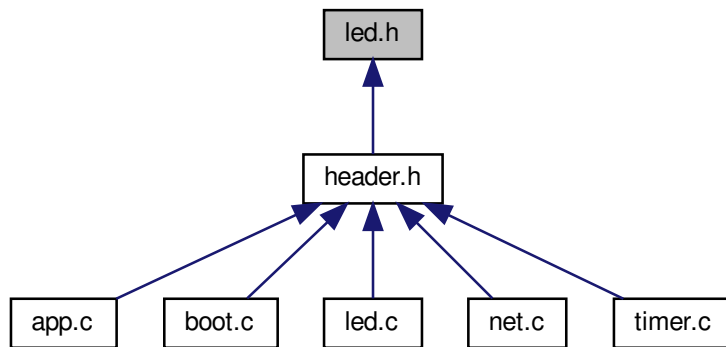
#### Returns

none.

## 7.896 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:



## Functions

- void [LedInit](#) (void)  
*Initializes the LED.*
- void [LedToggle](#) (void)  
*Toggles the LED at a fixed time interval.*

### 7.896.1 Detailed Description

LED driver header file.

### 7.896.2 Function Documentation

#### 7.896.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

#### Returns

none.

Initializes the LED.

#### Returns

none.

### 7.896.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

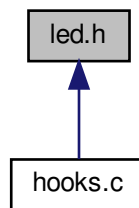
#### Returns

none.

## 7.897 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:



### Functions

- void [LedBlinkInit](#) ([blt\\_int16u](#) interval\_ms)  
*Initializes the LED blink driver.*
- void [LedBlinkTask](#) (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void [LedBlinkExit](#) (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

### 7.897.1 Detailed Description

LED driver header file.

### 7.897.2 Function Documentation

### 7.897.2.1 LedBlinkExit()

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

#### Returns

none.

### 7.897.2.2 LedBlinkInit()

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

#### Parameters

|                    |                                                                |
|--------------------|----------------------------------------------------------------|
| <i>interval_ms</i> | Specifies the desired LED blink interval time in milliseconds. |
|--------------------|----------------------------------------------------------------|

#### Returns

none.

### 7.897.2.3 LedBlinkTask()

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

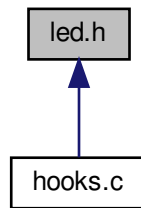
#### Returns

none.

## 7.898 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:



## Functions

- void [LedBlinkInit](#) ([blt\\_int16u](#) interval\_ms)  
*Initializes the LED blink driver.*
- void [LedBlinkTask](#) (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void [LedBlinkExit](#) (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

### 7.898.1 Detailed Description

LED driver header file.

### 7.898.2 Function Documentation

#### 7.898.2.1 LedBlinkExit()

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

#### Returns

none.

#### 7.898.2.2 LedBlinkInit()

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.



## Parameters

|                          |                                                                |
|--------------------------|----------------------------------------------------------------|
| <code>interval_ms</code> | Specifies the desired LED blink interval time in milliseconds. |
|--------------------------|----------------------------------------------------------------|

## Returns

none.

## 7.898.2.3 LedBlinkTask()

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

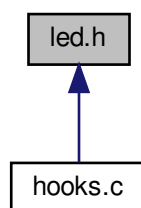
## Returns

none.

## 7.899 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:



## Functions

- void [LedBlinkInit](#) ([blt\\_int16u](#) interval\_ms)  
*Initializes the LED blink driver.*
- void [LedBlinkTask](#) (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void [LedBlinkExit](#) (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

## 7.899.1 Detailed Description

LED driver header file.

## 7.899.2 Function Documentation

### 7.899.2.1 LedBlinkExit()

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

#### Returns

none.

### 7.899.2.2 LedBlinkInit()

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

#### Parameters

|                    |                                                                |
|--------------------|----------------------------------------------------------------|
| <i>interval_ms</i> | Specifies the desired LED blink interval time in milliseconds. |
|--------------------|----------------------------------------------------------------|

#### Returns

none.

### 7.899.2.3 LedBlinkTask()

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

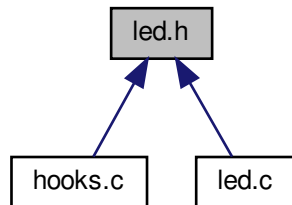
#### Returns

none.

## 7.900 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:



### Functions

- void [LedBlinkInit](#) ([blt\\_int16u](#) interval\_ms)  
*Initializes the LED blink driver.*
- void [LedBlinkTask](#) (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void [LedBlinkExit](#) (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

### 7.900.1 Detailed Description

LED driver header file.

### 7.900.2 Function Documentation

#### 7.900.2.1 LedBlinkExit()

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

#### Returns

none.

#### 7.900.2.2 LedBlinkInit()

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

**Parameters**

|                    |                                                                |
|--------------------|----------------------------------------------------------------|
| <i>interval_ms</i> | Specifies the desired LED blink interval time in milliseconds. |
|--------------------|----------------------------------------------------------------|

**Returns**

none.

**7.900.2.3 LedBlinkTask()**

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

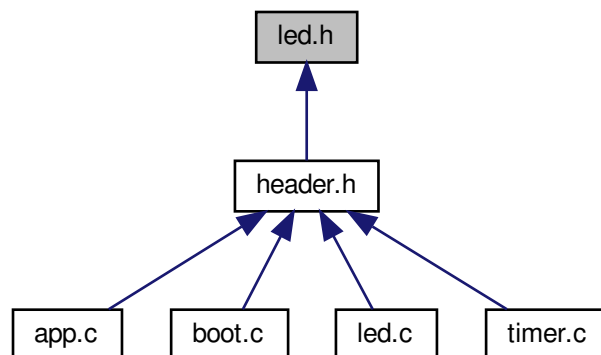
**Returns**

none.

**7.901 led.h File Reference**

LED driver header file.

This graph shows which files directly or indirectly include this file:

**Functions**

- void [LedInit](#) (void)  
*Initializes the LED.*
- void [LedToggle](#) (void)  
*Toggles the LED at a fixed time interval.*

### 7.901.1 Detailed Description

LED driver header file.

### 7.901.2 Function Documentation

#### 7.901.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

##### Returns

none.

Initializes the LED.

##### Returns

none.

#### 7.901.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

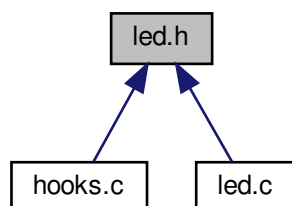
##### Returns

none.

## 7.902 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:



## Functions

- void [LedBlinkInit](#) ([blt\\_int16u](#) interval\_ms)  
*Initializes the LED blink driver.*
- void [LedBlinkTask](#) (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void [LedBlinkExit](#) (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

### 7.902.1 Detailed Description

LED driver header file.

### 7.902.2 Function Documentation

#### 7.902.2.1 LedBlinkExit()

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

#### Returns

none.

#### 7.902.2.2 LedBlinkInit()

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

#### Parameters

|                    |                                                                |
|--------------------|----------------------------------------------------------------|
| <i>interval_ms</i> | Specifies the desired LED blink interval time in milliseconds. |
|--------------------|----------------------------------------------------------------|

#### Returns

none.

### 7.902.2.3 LedBlinkTask()

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

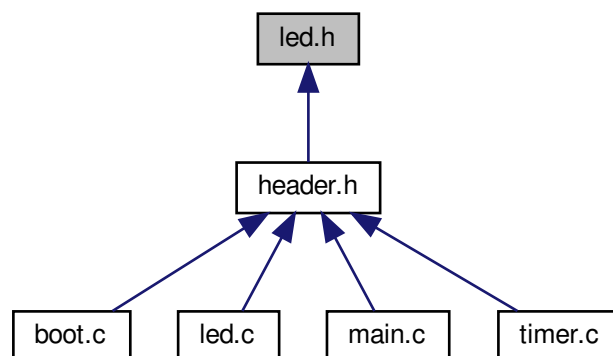
#### Returns

none.

## 7.903 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:



### Functions

- void [LedInit](#) (void)  
*Initializes the LED.*
- void [LedToggle](#) (void)  
*Toggles the LED at a fixed time interval.*

### 7.903.1 Detailed Description

LED driver header file.

### 7.903.2 Function Documentation

### 7.903.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

#### Returns

none.

Initializes the LED.

#### Returns

none.

### 7.903.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

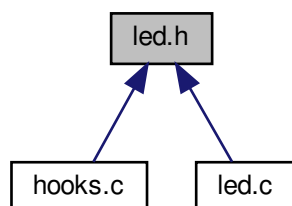
#### Returns

none.

## 7.904 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:





## Functions

- void [LedBlinkInit](#) ([blt\\_int16u](#) interval\_ms)  
*Initializes the LED blink driver.*
- void [LedBlinkTask](#) (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void [LedBlinkExit](#) (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

### 7.904.1 Detailed Description

LED driver header file.

### 7.904.2 Function Documentation

#### 7.904.2.1 LedBlinkExit()

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

#### Returns

none.

#### 7.904.2.2 LedBlinkInit()

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

#### Parameters

|                    |                                                                |
|--------------------|----------------------------------------------------------------|
| <i>interval_ms</i> | Specifies the desired LED blink interval time in milliseconds. |
|--------------------|----------------------------------------------------------------|

#### Returns

none.

### 7.904.2.3 LedBlinkTask()

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

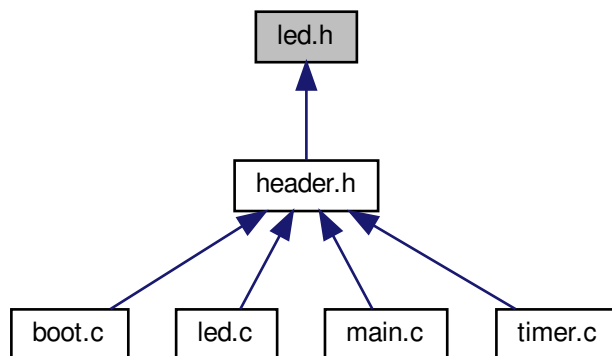
#### Returns

none.

## 7.905 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:



### Functions

- void [LedInit](#) (void)  
*Initializes the LED.*
- void [LedToggle](#) (void)  
*Toggles the LED at a fixed time interval.*

### 7.905.1 Detailed Description

LED driver header file.

### 7.905.2 Function Documentation

### 7.905.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

#### Returns

none.

Initializes the LED.

#### Returns

none.

### 7.905.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

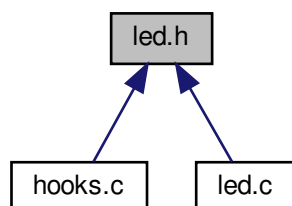
#### Returns

none.

## 7.906 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:



## Functions

- void [LedBlinkInit](#) ([blt\\_int16u](#) interval\_ms)  
*Initializes the LED blink driver.*
- void [LedBlinkTask](#) (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void [LedBlinkExit](#) (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

### 7.906.1 Detailed Description

LED driver header file.

### 7.906.2 Function Documentation

#### 7.906.2.1 LedBlinkExit()

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

#### Returns

none.

#### 7.906.2.2 LedBlinkInit()

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

#### Parameters

|                             |                                                                |
|-----------------------------|----------------------------------------------------------------|
| <a href="#">interval_ms</a> | Specifies the desired LED blink interval time in milliseconds. |
|-----------------------------|----------------------------------------------------------------|

#### Returns

none.

### 7.906.2.3 LedBlinkTask()

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

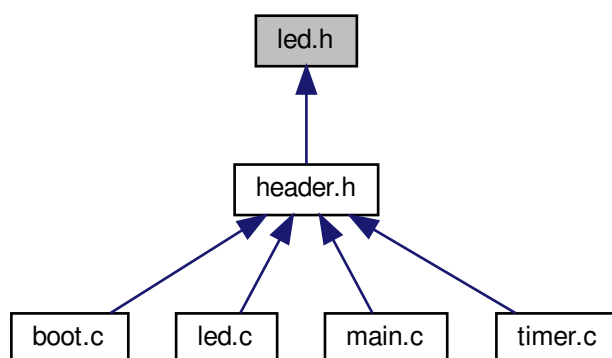
#### Returns

none.

## 7.907 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:



### Functions

- void [LedInit](#) (void)  
*Initializes the LED.*
- void [LedToggle](#) (void)  
*Toggles the LED at a fixed time interval.*

### 7.907.1 Detailed Description

LED driver header file.

### 7.907.2 Function Documentation

### 7.907.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

#### Returns

none.

Initializes the LED.

#### Returns

none.

### 7.907.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

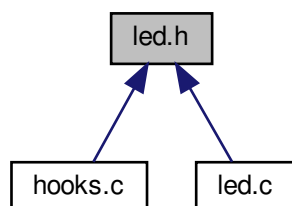
#### Returns

none.

## 7.908 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:



## Functions

- void [LedBlinkInit](#) ([blt\\_int16u](#) interval\_ms)  
*Initializes the LED blink driver.*
- void [LedBlinkTask](#) (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void [LedBlinkExit](#) (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

### 7.908.1 Detailed Description

LED driver header file.

### 7.908.2 Function Documentation

#### 7.908.2.1 LedBlinkExit()

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

#### Returns

none.

#### 7.908.2.2 LedBlinkInit()

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

#### Parameters

|                    |                                                                |
|--------------------|----------------------------------------------------------------|
| <i>interval_ms</i> | Specifies the desired LED blink interval time in milliseconds. |
|--------------------|----------------------------------------------------------------|

#### Returns

none.

### 7.908.2.3 LedBlinkTask()

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

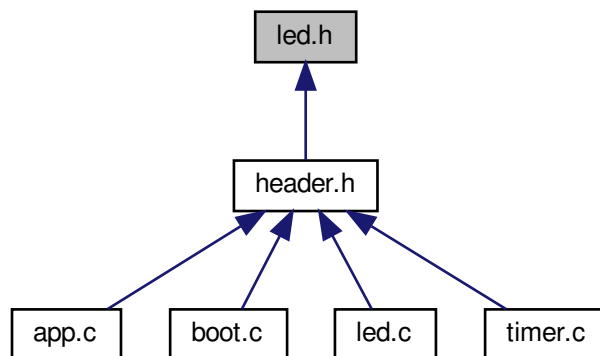
#### Returns

none.

## 7.909 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:



### Functions

- void [LedInit](#) (void)  
*Initializes the LED.*
- void [LedToggle](#) (void)  
*Toggles the LED at a fixed time interval.*

### 7.909.1 Detailed Description

LED driver header file.

### 7.909.2 Function Documentation



### 7.909.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

#### Returns

none.

Initializes the LED.

#### Returns

none.

### 7.909.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

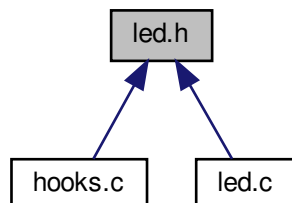
#### Returns

none.

## 7.910 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:



## Functions

- void [LedBlinkInit](#) ([blt\\_int16u](#) interval\_ms)  
*Initializes the LED blink driver.*
- void [LedBlinkTask](#) (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void [LedBlinkExit](#) (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

### 7.910.1 Detailed Description

LED driver header file.

### 7.910.2 Function Documentation

#### 7.910.2.1 LedBlinkExit()

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

#### Returns

none.

#### 7.910.2.2 LedBlinkInit()

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

#### Parameters

|                    |                                                                |
|--------------------|----------------------------------------------------------------|
| <i>interval_ms</i> | Specifies the desired LED blink interval time in milliseconds. |
|--------------------|----------------------------------------------------------------|

#### Returns

none.

### 7.910.2.3 LedBlinkTask()

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

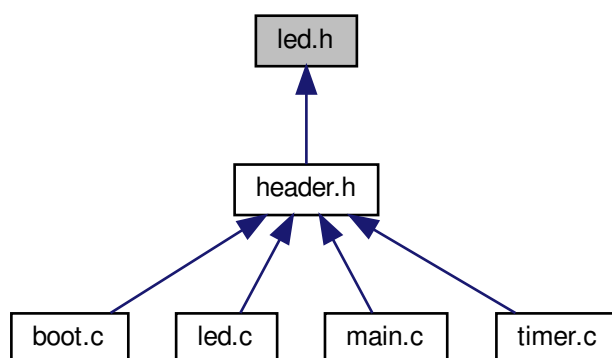
#### Returns

none.

## 7.911 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:



### Functions

- void [LedInit](#) (void)  
*Initializes the LED.*
- void [LedToggle](#) (void)  
*Toggles the LED at a fixed time interval.*

### 7.911.1 Detailed Description

LED driver header file.

### 7.911.2 Function Documentation

### 7.911.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

#### Returns

none.

Initializes the LED.

#### Returns

none.

### 7.911.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

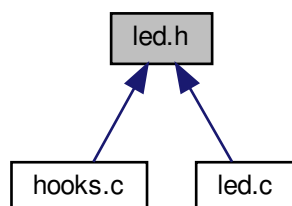
#### Returns

none.

## 7.912 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:



## Functions

- void [LedBlinkInit](#) ([blt\\_int16u](#) interval\_ms)  
*Initializes the LED blink driver.*
- void [LedBlinkTask](#) (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void [LedBlinkExit](#) (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

### 7.912.1 Detailed Description

LED driver header file.

### 7.912.2 Function Documentation

#### 7.912.2.1 LedBlinkExit()

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

#### Returns

none.

#### 7.912.2.2 LedBlinkInit()

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

#### Parameters

|                    |                                                                |
|--------------------|----------------------------------------------------------------|
| <i>interval_ms</i> | Specifies the desired LED blink interval time in milliseconds. |
|--------------------|----------------------------------------------------------------|

#### Returns

none.

### 7.912.2.3 LedBlinkTask()

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

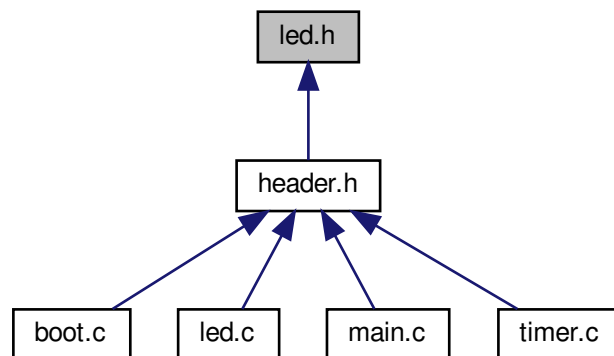
#### Returns

none.

## 7.913 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:



### Functions

- void [LedInit](#) (void)  
*Initializes the LED.*
- void [LedToggle](#) (void)  
*Toggles the LED at a fixed time interval.*

### 7.913.1 Detailed Description

LED driver header file.

### 7.913.2 Function Documentation

### 7.913.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

#### Returns

none.

Initializes the LED.

#### Returns

none.

### 7.913.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

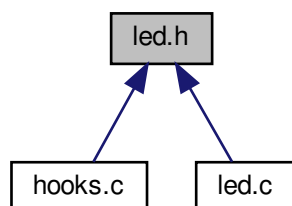
#### Returns

none.

## 7.914 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:



## Functions

- void [LedBlinkInit](#) ([blt\\_int16u](#) interval\_ms)  
*Initializes the LED blink driver.*
- void [LedBlinkTask](#) (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void [LedBlinkExit](#) (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

### 7.914.1 Detailed Description

LED driver header file.

### 7.914.2 Function Documentation

#### 7.914.2.1 LedBlinkExit()

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

#### Returns

none.

#### 7.914.2.2 LedBlinkInit()

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

#### Parameters

|                    |                                                                |
|--------------------|----------------------------------------------------------------|
| <i>interval_ms</i> | Specifies the desired LED blink interval time in milliseconds. |
|--------------------|----------------------------------------------------------------|

#### Returns

none.



### 7.914.2.3 LedBlinkTask()

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

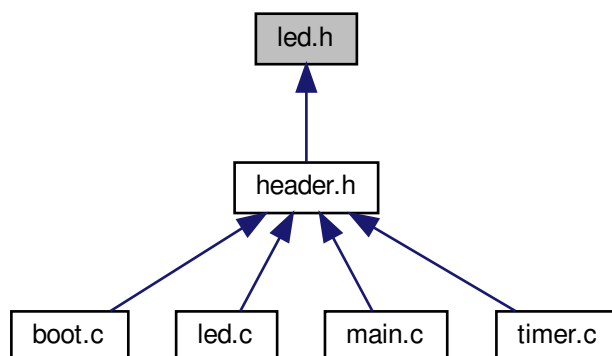
#### Returns

none.

## 7.915 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:



### Functions

- void [LedInit](#) (void)  
*Initializes the LED.*
- void [LedToggle](#) (void)  
*Toggles the LED at a fixed time interval.*

### 7.915.1 Detailed Description

LED driver header file.

### 7.915.2 Function Documentation

### 7.915.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

#### Returns

none.

Initializes the LED.

#### Returns

none.

### 7.915.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

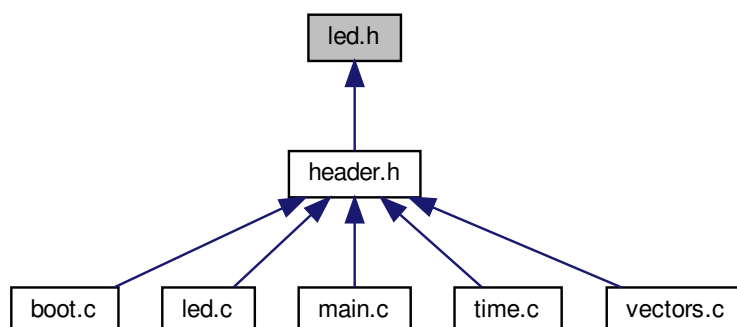
#### Returns

none.

## 7.916 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:



## Functions

- void [LedInit](#) (void)  
*Initializes the LED.*
- void [LedToggle](#) (void)  
*Toggles the LED at a fixed time interval.*

### 7.916.1 Detailed Description

LED driver header file.

### 7.916.2 Function Documentation

#### 7.916.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

##### Returns

none.

Initializes the LED.

##### Returns

none.

#### 7.916.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

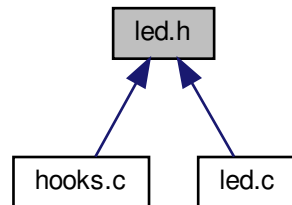
##### Returns

none.

## 7.917 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:



### Functions

- void [LedBlinkInit](#) ([blt\\_int16u](#) interval\_ms)  
*Initializes the LED blink driver.*
- void [LedBlinkTask](#) (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void [LedBlinkExit](#) (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

### 7.917.1 Detailed Description

LED driver header file.

### 7.917.2 Function Documentation

#### 7.917.2.1 LedBlinkExit()

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

#### Returns

none.

#### 7.917.2.2 LedBlinkInit()

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

**Parameters**

|                          |                                                                |
|--------------------------|----------------------------------------------------------------|
| <code>interval_ms</code> | Specifies the desired LED blink interval time in milliseconds. |
|--------------------------|----------------------------------------------------------------|

**Returns**

none.

**7.917.2.3 LedBlinkTask()**

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

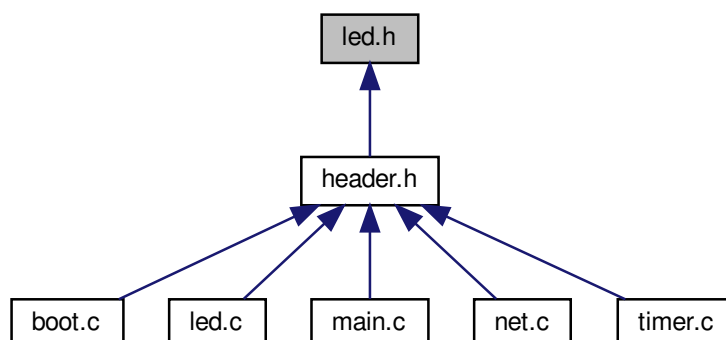
**Returns**

none.

## 7.918 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:

**Functions**

- void [LedInit](#) (void)  
*Initializes the LED.*
- void [LedToggle](#) (void)  
*Toggles the LED at a fixed time interval.*

### 7.918.1 Detailed Description

LED driver header file.

### 7.918.2 Function Documentation

#### 7.918.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

##### Returns

none.

Initializes the LED.

##### Returns

none.

#### 7.918.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

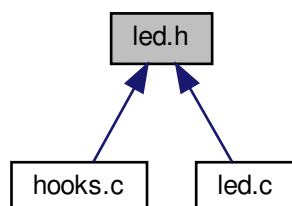
##### Returns

none.

## 7.919 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:



## Functions

- void [LedBlinkInit](#) ([blt\\_int16u](#) interval\_ms)  
*Initializes the LED blink driver.*
- void [LedBlinkTask](#) (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void [LedBlinkExit](#) (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

### 7.919.1 Detailed Description

LED driver header file.

### 7.919.2 Function Documentation

#### 7.919.2.1 LedBlinkExit()

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

#### Returns

none.

#### 7.919.2.2 LedBlinkInit()

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

#### Parameters

|                    |                                                                |
|--------------------|----------------------------------------------------------------|
| <i>interval_ms</i> | Specifies the desired LED blink interval time in milliseconds. |
|--------------------|----------------------------------------------------------------|

#### Returns

none.

### 7.919.2.3 LedBlinkTask()

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

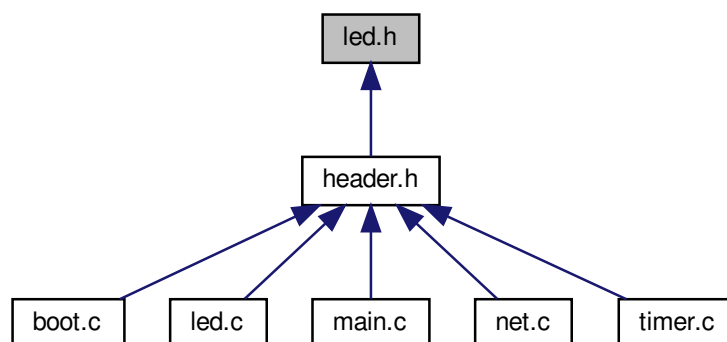
#### Returns

none.

## 7.920 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:



### Functions

- void [LedInit](#) (void)  
*Initializes the LED.*
- void [LedToggle](#) (void)  
*Toggles the LED at a fixed time interval.*

### 7.920.1 Detailed Description

LED driver header file.

### 7.920.2 Function Documentation



### 7.920.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

#### Returns

none.

Initializes the LED.

#### Returns

none.

### 7.920.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

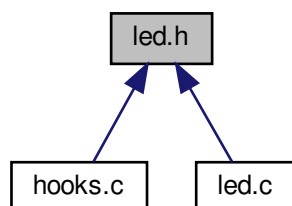
#### Returns

none.

## 7.921 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:



## Functions

- void [LedBlinkInit](#) ([blt\\_int16u](#) interval\_ms)  
*Initializes the LED blink driver.*
- void [LedBlinkTask](#) (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void [LedBlinkExit](#) (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

### 7.921.1 Detailed Description

LED driver header file.

### 7.921.2 Function Documentation

#### 7.921.2.1 LedBlinkExit()

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

#### Returns

none.

#### 7.921.2.2 LedBlinkInit()

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

#### Parameters

|                    |                                                                |
|--------------------|----------------------------------------------------------------|
| <i>interval_ms</i> | Specifies the desired LED blink interval time in milliseconds. |
|--------------------|----------------------------------------------------------------|

#### Returns

none.

### 7.921.2.3 LedBlinkTask()

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

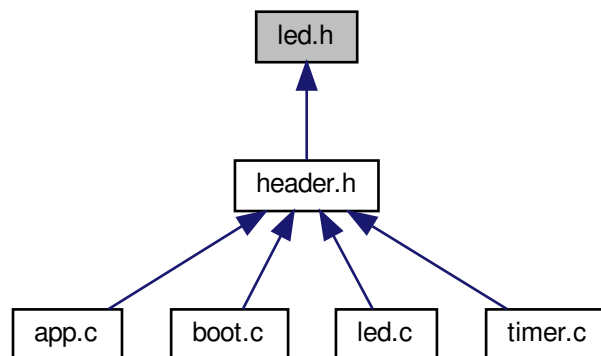
#### Returns

none.

## 7.922 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:



### Functions

- void [LedInit](#) (void)  
*Initializes the LED.*
- void [LedToggle](#) (void)  
*Toggles the LED at a fixed time interval.*

### 7.922.1 Detailed Description

LED driver header file.

### 7.922.2 Function Documentation

### 7.922.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

#### Returns

none.

Initializes the LED.

#### Returns

none.

### 7.922.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

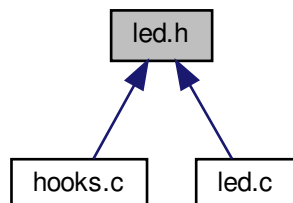
#### Returns

none.

## 7.923 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:



## Functions

- void [LedBlinkInit](#) ([blt\\_int16u](#) interval\_ms)  
*Initializes the LED blink driver.*
- void [LedBlinkTask](#) (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void [LedBlinkExit](#) (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

### 7.923.1 Detailed Description

LED driver header file.

### 7.923.2 Function Documentation

#### 7.923.2.1 LedBlinkExit()

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

#### Returns

none.

#### 7.923.2.2 LedBlinkInit()

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

#### Parameters

|                    |                                                                |
|--------------------|----------------------------------------------------------------|
| <i>interval_ms</i> | Specifies the desired LED blink interval time in milliseconds. |
|--------------------|----------------------------------------------------------------|

#### Returns

none.

### 7.923.2.3 LedBlinkTask()

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

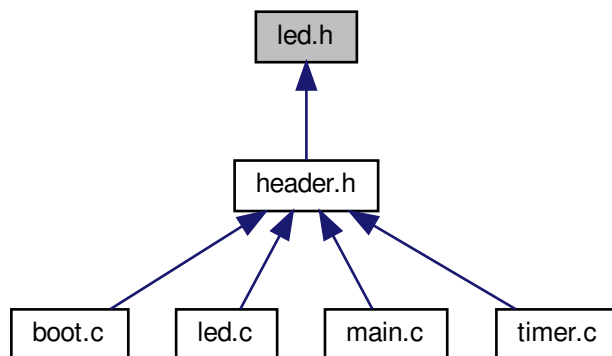
#### Returns

none.

## 7.924 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:



### Functions

- void [LedInit](#) (void)  
*Initializes the LED.*
- void [LedToggle](#) (void)  
*Toggles the LED at a fixed time interval.*

### 7.924.1 Detailed Description

LED driver header file.

### 7.924.2 Function Documentation

### 7.924.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

#### Returns

none.

Initializes the LED.

#### Returns

none.

### 7.924.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

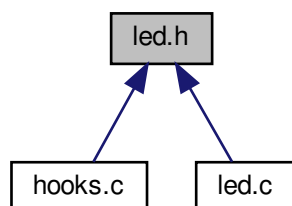
#### Returns

none.

## 7.925 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:



## Functions

- void [LedBlinkInit](#) ([blt\\_int16u](#) interval\_ms)  
*Initializes the LED blink driver.*
- void [LedBlinkTask](#) (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void [LedBlinkExit](#) (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

### 7.925.1 Detailed Description

LED driver header file.

### 7.925.2 Function Documentation

#### 7.925.2.1 LedBlinkExit()

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

#### Returns

none.

#### 7.925.2.2 LedBlinkInit()

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

#### Parameters

|                    |                                                                |
|--------------------|----------------------------------------------------------------|
| <i>interval_ms</i> | Specifies the desired LED blink interval time in milliseconds. |
|--------------------|----------------------------------------------------------------|

#### Returns

none.



### 7.925.2.3 LedBlinkTask()

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

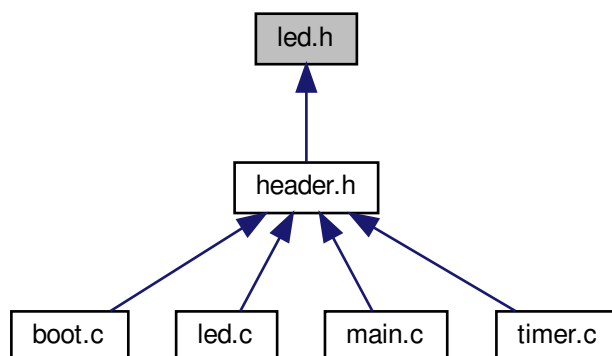
#### Returns

none.

## 7.926 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:



### Functions

- void [LedInit](#) (void)  
*Initializes the LED.*
- void [LedToggle](#) (void)  
*Toggles the LED at a fixed time interval.*

### 7.926.1 Detailed Description

LED driver header file.

### 7.926.2 Function Documentation

### 7.926.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

#### Returns

none.

Initializes the LED.

#### Returns

none.

### 7.926.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

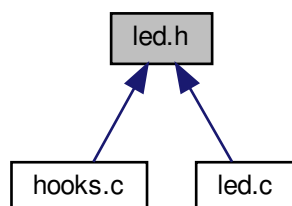
#### Returns

none.

## 7.927 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:



## Functions

- void [LedBlinkInit](#) ([blt\\_int16u](#) interval\_ms)  
*Initializes the LED blink driver.*
- void [LedBlinkTask](#) (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void [LedBlinkExit](#) (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

### 7.927.1 Detailed Description

LED driver header file.

### 7.927.2 Function Documentation

#### 7.927.2.1 LedBlinkExit()

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

#### Returns

none.

#### 7.927.2.2 LedBlinkInit()

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

#### Parameters

|                    |                                                                |
|--------------------|----------------------------------------------------------------|
| <i>interval_ms</i> | Specifies the desired LED blink interval time in milliseconds. |
|--------------------|----------------------------------------------------------------|

#### Returns

none.

### 7.927.2.3 LedBlinkTask()

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

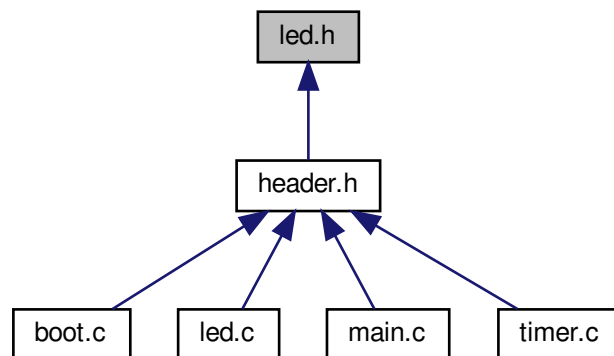
#### Returns

none.

## 7.928 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:



### Functions

- void [LedInit](#) (void)  
*Initializes the LED.*
- void [LedToggle](#) (void)  
*Toggles the LED at a fixed time interval.*

### 7.928.1 Detailed Description

LED driver header file.

### 7.928.2 Function Documentation

### 7.928.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

#### Returns

none.

Initializes the LED.

#### Returns

none.

### 7.928.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

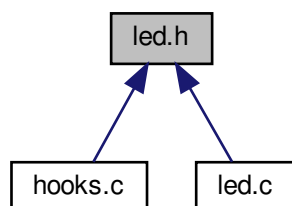
#### Returns

none.

## 7.929 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:



## Functions

- void [LedBlinkInit](#) ([blt\\_int16u](#) interval\_ms)  
*Initializes the LED blink driver.*
- void [LedBlinkTask](#) (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void [LedBlinkExit](#) (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

### 7.929.1 Detailed Description

LED driver header file.

### 7.929.2 Function Documentation

#### 7.929.2.1 LedBlinkExit()

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

#### Returns

none.

#### 7.929.2.2 LedBlinkInit()

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

#### Parameters

|                    |                                                                |
|--------------------|----------------------------------------------------------------|
| <i>interval_ms</i> | Specifies the desired LED blink interval time in milliseconds. |
|--------------------|----------------------------------------------------------------|

#### Returns

none.

### 7.929.2.3 LedBlinkTask()

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

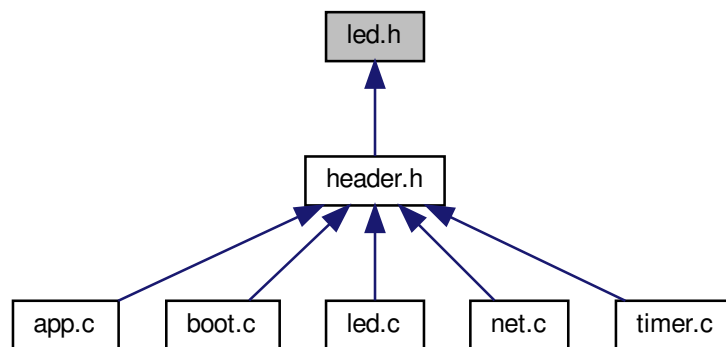
#### Returns

none.

## 7.930 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:



### Functions

- void [LedInit](#) (void)  
*Initializes the LED.*
- void [LedToggle](#) (void)  
*Toggles the LED at a fixed time interval.*

### 7.930.1 Detailed Description

LED driver header file.

### 7.930.2 Function Documentation

### 7.930.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

#### Returns

none.

Initializes the LED.

#### Returns

none.

### 7.930.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

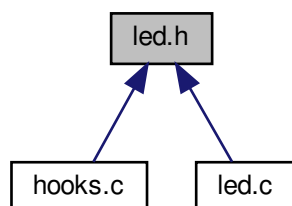
#### Returns

none.

## 7.931 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:





## Functions

- void [LedBlinkInit](#) ([blt\\_int16u](#) interval\_ms)  
*Initializes the LED blink driver.*
- void [LedBlinkTask](#) (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void [LedBlinkExit](#) (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

### 7.931.1 Detailed Description

LED driver header file.

### 7.931.2 Function Documentation

#### 7.931.2.1 LedBlinkExit()

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

#### Returns

none.

#### 7.931.2.2 LedBlinkInit()

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

#### Parameters

|                    |                                                                |
|--------------------|----------------------------------------------------------------|
| <i>interval_ms</i> | Specifies the desired LED blink interval time in milliseconds. |
|--------------------|----------------------------------------------------------------|

#### Returns

none.

### 7.931.2.3 LedBlinkTask()

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

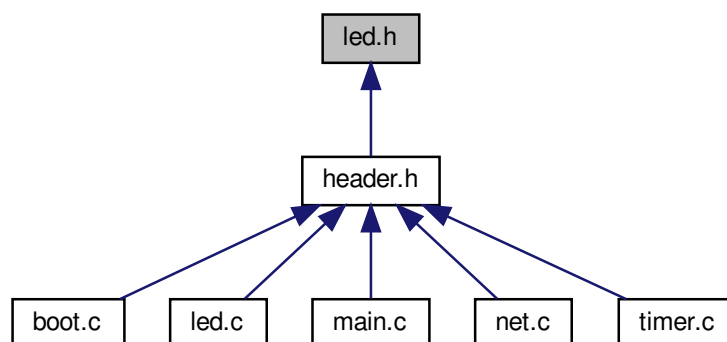
#### Returns

none.

## 7.932 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:



#### Functions

- void [LedInit](#) (void)  
*Initializes the LED.*
- void [LedToggle](#) (void)  
*Toggles the LED at a fixed time interval.*

### 7.932.1 Detailed Description

LED driver header file.

### 7.932.2 Function Documentation

### 7.932.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

#### Returns

none.

Initializes the LED.

#### Returns

none.

### 7.932.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

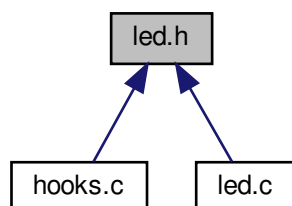
#### Returns

none.

## 7.933 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:



## Functions

- void [LedBlinkInit](#) ([blt\\_int16u](#) interval\_ms)  
*Initializes the LED blink driver.*
- void [LedBlinkTask](#) (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void [LedBlinkExit](#) (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

### 7.933.1 Detailed Description

LED driver header file.

### 7.933.2 Function Documentation

#### 7.933.2.1 LedBlinkExit()

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

#### Returns

none.

#### 7.933.2.2 LedBlinkInit()

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

#### Parameters

|                    |                                                                |
|--------------------|----------------------------------------------------------------|
| <i>interval_ms</i> | Specifies the desired LED blink interval time in milliseconds. |
|--------------------|----------------------------------------------------------------|

#### Returns

none.

### 7.933.2.3 LedBlinkTask()

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

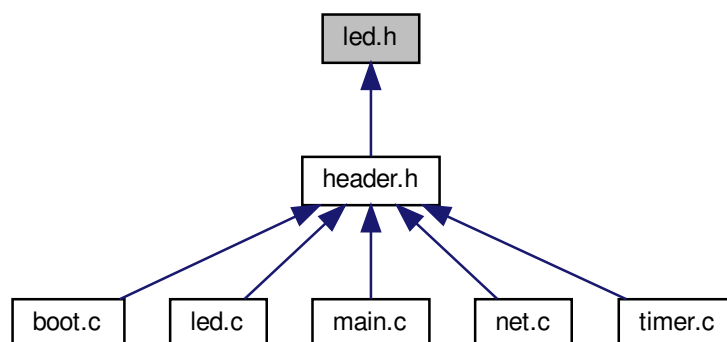
#### Returns

none.

## 7.934 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:



### Functions

- void [LedInit](#) (void)  
*Initializes the LED.*
- void [LedToggle](#) (void)  
*Toggles the LED at a fixed time interval.*

### 7.934.1 Detailed Description

LED driver header file.

### 7.934.2 Function Documentation

### 7.934.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

#### Returns

none.

Initializes the LED.

#### Returns

none.

### 7.934.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

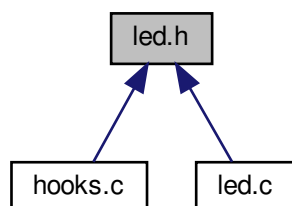
#### Returns

none.

## 7.935 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:



## Functions

- void [LedBlinkInit](#) ([blt\\_int16u](#) interval\_ms)  
*Initializes the LED blink driver.*
- void [LedBlinkTask](#) (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void [LedBlinkExit](#) (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

### 7.935.1 Detailed Description

LED driver header file.

### 7.935.2 Function Documentation

#### 7.935.2.1 LedBlinkExit()

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

#### Returns

none.

#### 7.935.2.2 LedBlinkInit()

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

#### Parameters

|                    |                                                                |
|--------------------|----------------------------------------------------------------|
| <i>interval_ms</i> | Specifies the desired LED blink interval time in milliseconds. |
|--------------------|----------------------------------------------------------------|

#### Returns

none.

### 7.935.2.3 LedBlinkTask()

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

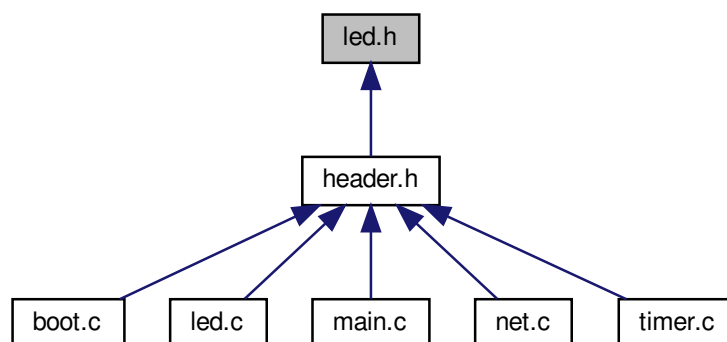
#### Returns

none.

## 7.936 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:



### Functions

- void [LedInit](#) (void)  
*Initializes the LED.*
- void [LedToggle](#) (void)  
*Toggles the LED at a fixed time interval.*

### 7.936.1 Detailed Description

LED driver header file.

### 7.936.2 Function Documentation



### 7.936.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

#### Returns

none.

Initializes the LED.

#### Returns

none.

### 7.936.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

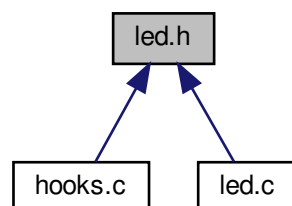
#### Returns

none.

## 7.937 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:



## Functions

- void [LedBlinkInit](#) ([blt\\_int16u](#) interval\_ms)  
*Initializes the LED blink driver.*
- void [LedBlinkTask](#) (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void [LedBlinkExit](#) (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

### 7.937.1 Detailed Description

LED driver header file.

### 7.937.2 Function Documentation

#### 7.937.2.1 LedBlinkExit()

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

#### Returns

none.

#### 7.937.2.2 LedBlinkInit()

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

#### Parameters

|                    |                                                                |
|--------------------|----------------------------------------------------------------|
| <i>interval_ms</i> | Specifies the desired LED blink interval time in milliseconds. |
|--------------------|----------------------------------------------------------------|

#### Returns

none.

### 7.937.2.3 LedBlinkTask()

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

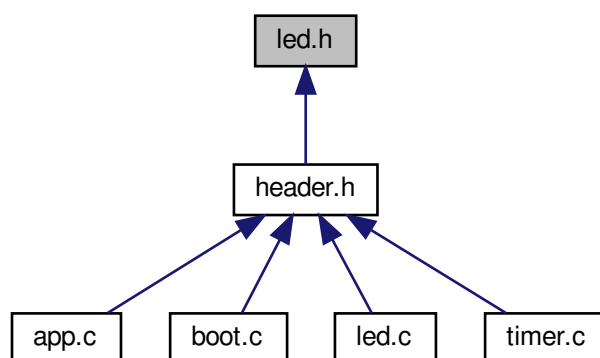
#### Returns

none.

## 7.938 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:



### Functions

- void [LedInit](#) (void)  
*Initializes the LED.*
- void [LedToggle](#) (void)  
*Toggles the LED at a fixed time interval.*

### 7.938.1 Detailed Description

LED driver header file.

### 7.938.2 Function Documentation

### 7.938.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

#### Returns

none.

Initializes the LED.

#### Returns

none.

### 7.938.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

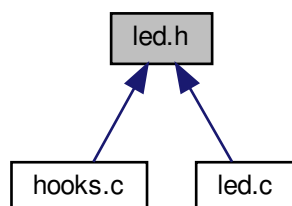
#### Returns

none.

## 7.939 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:



## Functions

- void [LedBlinkInit](#) ([blt\\_int16u](#) interval\_ms)  
*Initializes the LED blink driver.*
- void [LedBlinkTask](#) (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void [LedBlinkExit](#) (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

### 7.939.1 Detailed Description

LED driver header file.

### 7.939.2 Function Documentation

#### 7.939.2.1 LedBlinkExit()

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

#### Returns

none.

#### 7.939.2.2 LedBlinkInit()

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

#### Parameters

|                    |                                                                |
|--------------------|----------------------------------------------------------------|
| <i>interval_ms</i> | Specifies the desired LED blink interval time in milliseconds. |
|--------------------|----------------------------------------------------------------|

#### Returns

none.

### 7.939.2.3 LedBlinkTask()

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

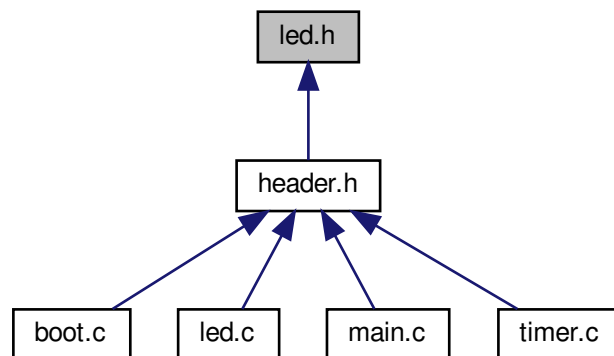
#### Returns

none.

## 7.940 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:



### Functions

- void [LedInit](#) (void)  
*Initializes the LED.*
- void [LedToggle](#) (void)  
*Toggles the LED at a fixed time interval.*

### 7.940.1 Detailed Description

LED driver header file.

### 7.940.2 Function Documentation

### 7.940.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

#### Returns

none.

Initializes the LED.

#### Returns

none.

### 7.940.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

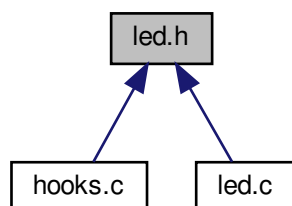
#### Returns

none.

## 7.941 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:



## Functions

- void [LedBlinkInit](#) ([blt\\_int16u](#) interval\_ms)  
*Initializes the LED blink driver.*
- void [LedBlinkTask](#) (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void [LedBlinkExit](#) (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

### 7.941.1 Detailed Description

LED driver header file.

### 7.941.2 Function Documentation

#### 7.941.2.1 LedBlinkExit()

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

#### Returns

none.

#### 7.941.2.2 LedBlinkInit()

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

#### Parameters

|                    |                                                                |
|--------------------|----------------------------------------------------------------|
| <i>interval_ms</i> | Specifies the desired LED blink interval time in milliseconds. |
|--------------------|----------------------------------------------------------------|

#### Returns

none.



### 7.941.2.3 LedBlinkTask()

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

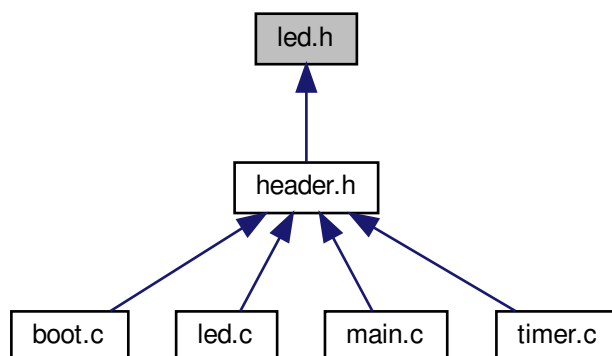
#### Returns

none.

## 7.942 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:



### Functions

- void [LedInit](#) (void)  
*Initializes the LED.*
- void [LedToggle](#) (void)  
*Toggles the LED at a fixed time interval.*

### 7.942.1 Detailed Description

LED driver header file.

### 7.942.2 Function Documentation

### 7.942.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

#### Returns

none.

Initializes the LED.

#### Returns

none.

### 7.942.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

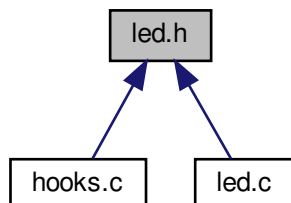
#### Returns

none.

## 7.943 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:



## Functions

- void [LedBlinkInit](#) ([blt\\_int16u](#) interval\_ms)  
*Initializes the LED blink driver.*
- void [LedBlinkTask](#) (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void [LedBlinkExit](#) (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

### 7.943.1 Detailed Description

LED driver header file.

### 7.943.2 Function Documentation

#### 7.943.2.1 LedBlinkExit()

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

#### Returns

none.

#### 7.943.2.2 LedBlinkInit()

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

#### Parameters

|                    |                                                                |
|--------------------|----------------------------------------------------------------|
| <i>interval_ms</i> | Specifies the desired LED blink interval time in milliseconds. |
|--------------------|----------------------------------------------------------------|

#### Returns

none.

### 7.943.2.3 LedBlinkTask()

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

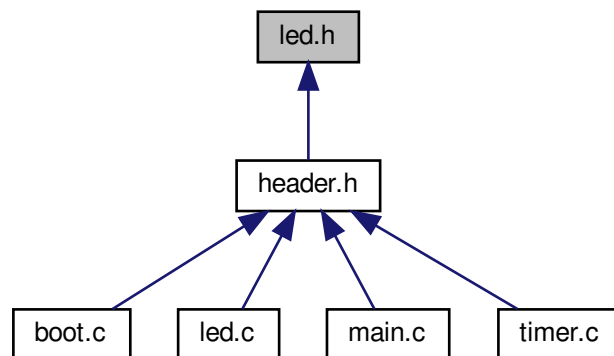
#### Returns

none.

## 7.944 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:



### Functions

- void [LedInit](#) (void)  
*Initializes the LED.*
- void [LedToggle](#) (void)  
*Toggles the LED at a fixed time interval.*

### 7.944.1 Detailed Description

LED driver header file.

### 7.944.2 Function Documentation

### 7.944.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

#### Returns

none.

Initializes the LED.

#### Returns

none.

### 7.944.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

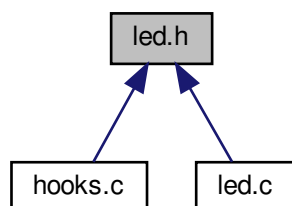
#### Returns

none.

## 7.945 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:



## Functions

- void [LedBlinkInit](#) ([blt\\_int16u](#) interval\_ms)  
*Initializes the LED blink driver.*
- void [LedBlinkTask](#) (void)  
*Task function for blinking the LED as a fixed timer interval.*
- void [LedBlinkExit](#) (void)  
*Cleans up the LED blink driver. This is intended to be used upon program exit.*

### 7.945.1 Detailed Description

LED driver header file.

### 7.945.2 Function Documentation

#### 7.945.2.1 LedBlinkExit()

```
void LedBlinkExit (
 void)
```

Cleans up the LED blink driver. This is intended to be used upon program exit.

#### Returns

none.

Referenced by [CpuUserProgramStartHook\(\)](#).

#### 7.945.2.2 LedBlinkInit()

```
void LedBlinkInit (
 blt_int16u interval_ms)
```

Initializes the LED blink driver.

#### Parameters

|                    |                                                                |
|--------------------|----------------------------------------------------------------|
| <i>interval_ms</i> | Specifies the desired LED blink interval time in milliseconds. |
|--------------------|----------------------------------------------------------------|

#### Returns

none.

Referenced by CopInitHook().

### 7.945.2.3 LedBlinkTask()

```
void LedBlinkTask (
 void)
```

Task function for blinking the LED as a fixed timer interval.

#### Returns

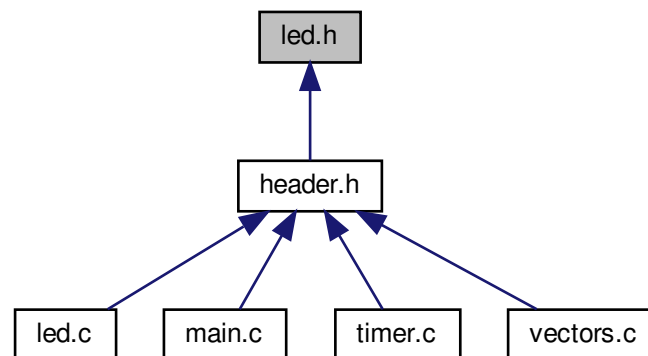
none.

Referenced by CopServiceHook().

## 7.946 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:



## Functions

- void [LedInit](#) (void)  
*Initializes the LED.*
- void [LedToggle](#) (void)  
*Toggles the LED at a fixed time interval.*

### 7.946.1 Detailed Description

LED driver header file.

### 7.946.2 Function Documentation

#### 7.946.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

**Returns**

none.

Initializes the LED.

**Returns**

none.

#### 7.946.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

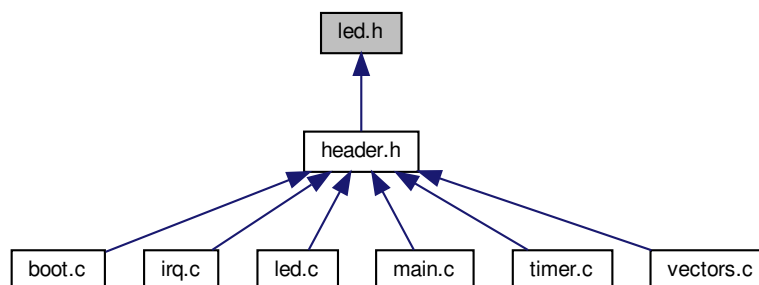
**Returns**

none.

## 7.947 led.h File Reference

LED driver header file.

This graph shows which files directly or indirectly include this file:





## Functions

- void [LedInit](#) (void)  
*Initializes the LED.*
- void [LedToggle](#) (void)  
*Toggles the LED at a fixed time interval.*

### 7.947.1 Detailed Description

LED driver header file.

### 7.947.2 Function Documentation

#### 7.947.2.1 LedInit()

```
void LedInit (
 void)
```

Initializes the LED.

##### Returns

none.

Initializes the LED.

##### Returns

none.

Referenced by [Applnit\(\)](#), and [Init\(\)](#).

#### 7.947.2.2 LedToggle()

```
void LedToggle (
 void)
```

Toggles the LED at a fixed time interval.

##### Returns

none.

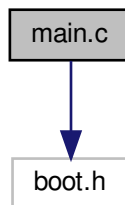
Referenced by [AppTask\(\)](#), and [main\(\)](#).

## 7.948 main.c File Reference

Bootloader application source file.

```
#include "boot.h"
```

Include dependency graph for \_template/Boot/main.c:



### Functions

- static void [Init](#) (void)  
*Initializes the microcontroller.*
- int [main](#) (void)  
*This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.*

### 7.948.1 Detailed Description

Bootloader application source file.

### 7.948.2 Function Documentation

#### 7.948.2.1 Init()

```
static void Init (
 void) [static]
```

Initializes the microcontroller.

#### Returns

none.

Referenced by [main\(\)](#).

### 7.948.2.2 main()

```
int main (
 void)
```

This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.

#### Returns

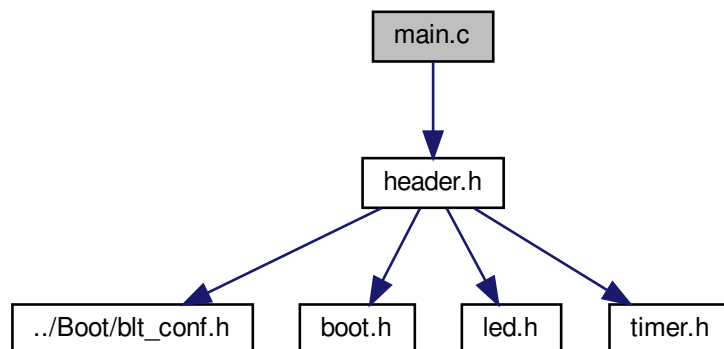
Program return code.

## 7.949 main.c File Reference

Demo program application source file.

```
#include "header.h"
```

Include dependency graph for `_template/Prog/main.c`:



#### Functions

- static void `Init` (void)  
*Initializes the microcontroller.*
- int `main` (void)  
*This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.*

### 7.949.1 Detailed Description

Demo program application source file.

## 7.949.2 Function Documentation

### 7.949.2.1 Init()

```
static void Init (
 void) [static]
```

Initializes the microcontroller.

#### Returns

none.

Referenced by main().

### 7.949.2.2 main()

```
int main (
 void)
```

This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.

#### Returns

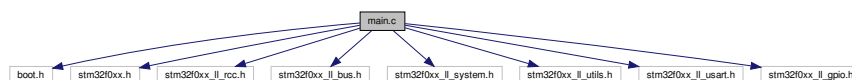
Program return code.

## 7.950 main.c File Reference

Bootloader application source file.

```
#include "boot.h"
#include "stm32f0xx.h"
#include "stm32f0xx_ll_rcc.h"
#include "stm32f0xx_ll_bus.h"
#include "stm32f0xx_ll_system.h"
#include "stm32f0xx_ll_utils.h"
#include "stm32f0xx_ll_usart.h"
#include "stm32f0xx_ll_gpio.h"
```

Include dependency graph for ARMCM0\_STM32F0\_Discovery\_STM32F051\_GCC/Boot/main.c:



## Functions

- static void `Init` (void)  
*Initializes the microcontroller.*
- static void `SystemClock_Config` (void)  
*System Clock Configuration. This code was created by CubeMX and configures the system clock to match the configuration in the bootloader's configuration (blt\_conf.h), specifically the macros: BOOT\_CPU\_SYSTEM\_SPEED\_KHZ and BOOT\_CPU\_XTAL\_SPEED\_KHZ. Note that the Lower Layer drivers were selected in CubeMX for the RCC subsystem.*
- int `main` (void)  
*This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.*
- void `HAL_MspInit` (void)  
*Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).*
- void `HAL_MspDeInit` (void)  
*DeInitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level de-initialization (GPIOs, clock, DMA, interrupt).*

### 7.950.1 Detailed Description

Bootloader application source file.

### 7.950.2 Function Documentation

#### 7.950.2.1 HAL\_MspDeInit()

```
void HAL_MspDeInit (
 void)
```

DeInitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level de-initialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

#### 7.950.2.2 HAL\_MspInit()

```
void HAL_MspInit (
 void)
```

Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

### 7.950.2.3 Init()

```
static void Init (
 void) [static]
```

Initializes the microcontroller.

#### Returns

none.

Referenced by main().

### 7.950.2.4 main()

```
int main (
 void)
```

This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.

#### Returns

Program return code.

### 7.950.2.5 SystemClock\_Config()

```
static void SystemClock_Config (
 void) [static]
```

System Clock Configuration. This code was created by CubeMX and configures the system clock to match the configuration in the bootloader's configuration (blt\_conf.h), specifically the macros: BOOT\_CPU\_SYSTEM\_SPEED\_KHZ and BOOT\_CPU\_XTAL\_SPEED\_KHZ. Note that the Lower Layer drivers were selected in CubeMX for the RCC subsystem.

#### Returns

none.

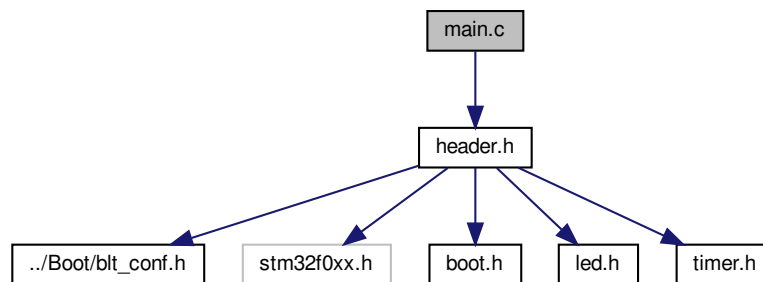
Referenced by Init().

## 7.951 main.c File Reference

Demo program application source file.

```
#include "header.h"
```

Include dependency graph for ARMCM0\_STM32F0\_Discovery\_STM32F051\_GCC/Prog/main.c:



### Functions

- static void `Init` (void)  
*Initializes the microcontroller.*
- static void `SystemClock_Config` (void)  
*System Clock Configuration. This code was created by CubeMX and configures the system clock.*
- int `main` (void)  
*This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.*
- void `HAL_MspInit` (void)  
*Initializes the Global MSP. This function is called from `HAL_Init()` function to perform system level initialization (GPIOs, clock, DMA, interrupt).*
- void `HAL_MspDeInit` (void)  
*Deinitializes the Global MSP. This function is called from `HAL_DeInit()` function to perform system level Deinitialization (GPIOs, clock, DMA, interrupt).*

### 7.951.1 Detailed Description

Demo program application source file.

### 7.951.2 Function Documentation

#### 7.951.2.1 HAL\_MspDeInit()

```
void HAL_MspDeInit (
 void)
```

Deinitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level Deinitialization (GPIOs, clock, DMA, interrupt).

##### Returns

none.

#### 7.951.2.2 HAL\_MspInit()

```
void HAL_MspInit (
 void)
```

Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).

##### Returns

none.

#### 7.951.2.3 Init()

```
static void Init (
 void) [static]
```

Initializes the microcontroller.

##### Returns

none.

Referenced by main().

#### 7.951.2.4 main()

```
int main (
 void)
```

This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.

##### Returns

Program exit code.



## 7.951.2.5 SystemClock\_Config()

```
static void SystemClock_Config (
 void) [static]
```

System Clock Configuration. This code was created by CubeMX and configures the system clock.

## Returns

none.

Referenced by Init().

## 7.952 main.c File Reference

Bootloader application source file.

```
#include "boot.h"
#include "stm32f0xx.h"
#include "stm32f0xx_ll_rcc.h"
#include "stm32f0xx_ll_bus.h"
#include "stm32f0xx_ll_system.h"
#include "stm32f0xx_ll_utils.h"
#include "stm32f0xx_ll_usart.h"
#include "stm32f0xx_ll_gpio.h"
```

Include dependency graph for ARMCM0\_STM32F0\_Discovery\_STM32F051\_IAR/Boot/main.c:



## Functions

- static void [Init](#) (void)  
*Initializes the microcontroller.*
- static void [SystemClock\\_Config](#) (void)  
*System Clock Configuration. This code was created by CubeMX and configures the system clock to match the configuration in the bootloader's configuration (blt\_conf.h), specifically the macros: BOOT\_CPU\_SYSTEM\_SPEED\_KHZ and BOOT\_CPU\_XTAL\_SPEED\_KHZ. Note that the Lower Layer drivers were selected in CubeMX for the RCC subsystem.*
- void [main](#) (void)  
*This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.*
- void [HAL\\_MspInit](#) (void)  
*Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).*
- void [HAL\\_MspDeInit](#) (void)  
*DeInitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level de-initialization (GPIOs, clock, DMA, interrupt).*

### 7.952.1 Detailed Description

Bootloader application source file.

### 7.952.2 Function Documentation

#### 7.952.2.1 HAL\_MspDeInit()

```
void HAL_MspDeInit (
 void)
```

DeInitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level de-initialization (GPIOs, clock, DMA, interrupt).

##### Returns

none.

#### 7.952.2.2 HAL\_MspInit()

```
void HAL_MspInit (
 void)
```

Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).

##### Returns

none.

#### 7.952.2.3 Init()

```
static void Init (
 void) [static]
```

Initializes the microcontroller.

##### Returns

none.

Referenced by main().

#### 7.952.2.4 main()

```
void main (
 void)
```

This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.

##### Returns

none.

#### 7.952.2.5 SystemClock\_Config()

```
static void SystemClock_Config (
 void) [static]
```

System Clock Configuration. This code was created by CubeMX and configures the system clock to match the configuration in the bootloader's configuration (blt\_conf.h), specifically the macros: BOOT\_CPU\_SYSTEM\_SPEED\_KHZ and BOOT\_CPU\_XTAL\_SPEED\_KHZ. Note that the Lower Layer drivers were selected in CubeMX for the RCC subsystem.

##### Returns

none.

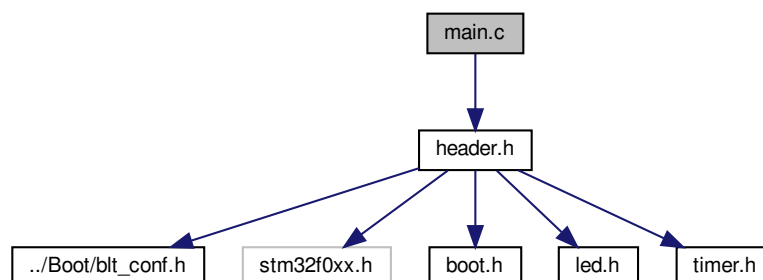
Referenced by Init().

## 7.953 main.c File Reference

Demo program application source file.

```
#include "header.h"
```

Include dependency graph for ARMCM0\_STM32F0\_Discovery\_STM32F051\_IAR/Prog/main.c:



## Functions

- static void `Init` (void)  
*Initializes the microcontroller.*
- static void `SystemClock_Config` (void)  
*System Clock Configuration. This code was created by CubeMX and configures the system clock.*
- void `main` (void)  
*This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.*
- void `HAL_MspInit` (void)  
*Initializes the Global MSP. This function is called from `HAL_Init()` function to perform system level initialization (GPIOs, clock, DMA, interrupt).*
- void `HAL_MspDeInit` (void)  
*Deinitializes the Global MSP. This function is called from `HAL_DeInit()` function to perform system level Deinitialization (GPIOs, clock, DMA, interrupt).*

### 7.953.1 Detailed Description

Demo program application source file.

### 7.953.2 Function Documentation

#### 7.953.2.1 HAL\_MspDeInit()

```
void HAL_MspDeInit (
 void)
```

Deinitializes the Global MSP. This function is called from `HAL_DeInit()` function to perform system level Deinitialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

#### 7.953.2.2 HAL\_MspInit()

```
void HAL_MspInit (
 void)
```

Initializes the Global MSP. This function is called from `HAL_Init()` function to perform system level initialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

### 7.953.2.3 Init()

```
static void Init (
 void) [static]
```

Initializes the microcontroller.

#### Returns

none.

Referenced by main().

### 7.953.2.4 main()

```
void main (
 void)
```

This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.

#### Returns

none.

### 7.953.2.5 SystemClock\_Config()

```
static void SystemClock_Config (
 void) [static]
```

System Clock Configuration. This code was created by CubeMX and configures the system clock.

#### Returns

none.

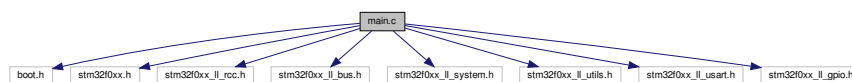
Referenced by Init().

## 7.954 main.c File Reference

Bootloader application source file.

```
#include "boot.h"
#include "stm32f0xx.h"
#include "stm32f0xx_ll_rcc.h"
#include "stm32f0xx_ll_bus.h"
#include "stm32f0xx_ll_system.h"
#include "stm32f0xx_ll_utils.h"
#include "stm32f0xx_ll_usart.h"
#include "stm32f0xx_ll_gpio.h"
```

Include dependency graph for ARMCM0\_STM32F0\_Discovery\_STM32F051\_Keil/Boot/main.c:



### Functions

- static void [Init](#) (void)  
*Initializes the microcontroller.*
- static void [SystemClock\\_Config](#) (void)  
*System Clock Configuration. This code was created by CubeMX and configures the system clock to match the configuration in the bootloader's configuration (blt\_conf.h), specifically the macros: BOOT\_CPU\_SYSTEM\_SPEED\_KHZ and BOOT\_CPU\_XTAL\_SPEED\_KHZ. Note that the Lower Layer drivers were selected in CubeMX for the RCC subsystem.*
- int [main](#) (void)  
*This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.*
- void [HAL\\_MspInit](#) (void)  
*Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).*
- void [HAL\\_MspDeInit](#) (void)  
*DeInitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level de-initialization (GPIOs, clock, DMA, interrupt).*

### 7.954.1 Detailed Description

Bootloader application source file.

### 7.954.2 Function Documentation

#### 7.954.2.1 HAL\_MspDeInit()

```
void HAL_MspDeInit (
 void)
```

DeInitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level de-initialization (GPIOs, clock, DMA, interrupt).

##### Returns

none.

#### 7.954.2.2 HAL\_MspInit()

```
void HAL_MspInit (
 void)
```

Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).

##### Returns

none.

#### 7.954.2.3 Init()

```
static void Init (
 void) [static]
```

Initializes the microcontroller.

##### Returns

none.

Referenced by main().

#### 7.954.2.4 main()

```
int main (
 void)
```

This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.

##### Returns

Program return code.

### 7.954.2.5 SystemClock\_Config()

```
static void SystemClock_Config (
 void) [static]
```

System Clock Configuration. This code was created by CubeMX and configures the system clock to match the configuration in the bootloader's configuration (blt\_conf.h), specifically the macros: BOOT\_CPU\_SYSTEM\_SPEED\_KHZ and BOOT\_CPU\_XTAL\_SPEED\_KHZ. Note that the Lower Layer drivers were selected in CubeMX for the RCC subsystem.

#### Returns

none.

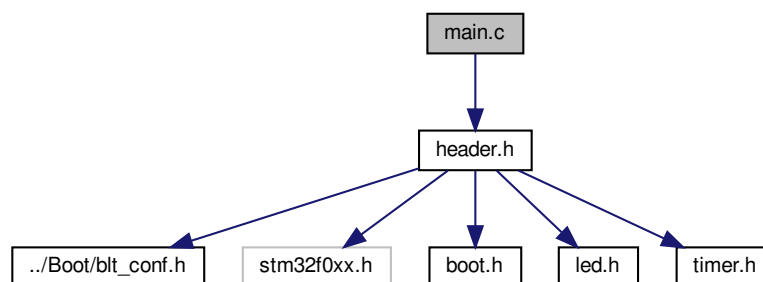
Referenced by Init().

## 7.955 main.c File Reference

Demo program application source file.

```
#include "header.h"
```

Include dependency graph for ARMCM0\_STM32F0\_Discovery\_STM32F051\_Keil/Prog/main.c:



## Functions

- static void [Init](#) (void)  
*Initializes the microcontroller.*
- static void [SystemClock\\_Config](#) (void)  
*System Clock Configuration. This code was created by CubeMX and configures the system clock.*
- int [main](#) (void)  
*This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.*
- void [HAL\\_MspInit](#) (void)  
*Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).*
- void [HAL\\_MspDeInit](#) (void)  
*Deinitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level Deinitialization (GPIOs, clock, DMA, interrupt).*



## 7.955.1 Detailed Description

Demo program application source file.

## 7.955.2 Function Documentation

### 7.955.2.1 HAL\_MspDeInit()

```
void HAL_MspDeInit (
 void)
```

Deinitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level Deinitialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

### 7.955.2.2 HAL\_MspInit()

```
void HAL_MspInit (
 void)
```

Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

### 7.955.2.3 Init()

```
static void Init (
 void) [static]
```

Initializes the microcontroller.

#### Returns

none.

Referenced by main().

#### 7.955.2.4 main()

```
int main (
 void)
```

This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.

##### Returns

none.

#### 7.955.2.5 SystemClock\_Config()

```
static void SystemClock_Config (
 void) [static]
```

System Clock Configuration. This code was created by CubeMX and configures the system clock.

##### Returns

none.

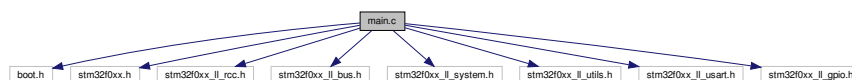
Referenced by Init().

## 7.956 main.c File Reference

Bootloader application source file.

```
#include "boot.h"
#include "stm32f0xx.h"
#include "stm32f0xx_ll_rcc.h"
#include "stm32f0xx_ll_bus.h"
#include "stm32f0xx_ll_system.h"
#include "stm32f0xx_ll_utils.h"
#include "stm32f0xx_ll_usart.h"
#include "stm32f0xx_ll_gpio.h"
```

Include dependency graph for ARMCM0\_STM32F0\_Nucleo\_F091RC\_GCC/Boot/main.c:



## Functions

- static void `Init` (void)  
*Initializes the microcontroller.*
- static void `SystemClock_Config` (void)  
*System Clock Configuration. This code was created by CubeMX and configures the system clock to match the configuration in the bootloader's configuration (blt\_conf.h), specifically the macros: BOOT\_CPU\_SYSTEM\_SPEED\_KHZ and BOOT\_CPU\_XTAL\_SPEED\_KHZ. Note that the Lower Layer drivers were selected in CubeMX for the RCC subsystem.*
- int `main` (void)  
*This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.*
- void `HAL_MspInit` (void)  
*Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).*
- void `HAL_MspDeInit` (void)  
*DeInitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level de-initialization (GPIOs, clock, DMA, interrupt).*

### 7.956.1 Detailed Description

Bootloader application source file.

### 7.956.2 Function Documentation

#### 7.956.2.1 HAL\_MspDeInit()

```
void HAL_MspDeInit (
 void)
```

DeInitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level de-initialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

#### 7.956.2.2 HAL\_MspInit()

```
void HAL_MspInit (
 void)
```

Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

### 7.956.2.3 Init()

```
static void Init (
 void) [static]
```

Initializes the microcontroller.

#### Returns

none.

Referenced by main().

### 7.956.2.4 main()

```
int main (
 void)
```

This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.

#### Returns

Program return code.

### 7.956.2.5 SystemClock\_Config()

```
static void SystemClock_Config (
 void) [static]
```

System Clock Configuration. This code was created by CubeMX and configures the system clock to match the configuration in the bootloader's configuration (blt\_conf.h), specifically the macros: BOOT\_CPU\_SYSTEM\_SPEED\_KHZ and BOOT\_CPU\_XTAL\_SPEED\_KHZ. Note that the Lower Layer drivers were selected in CubeMX for the RCC subsystem.

#### Returns

none.

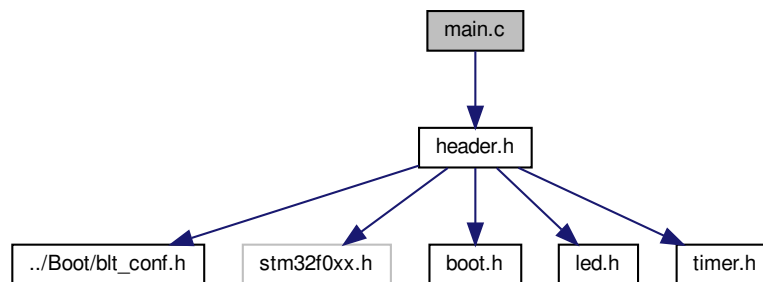
Referenced by Init().

## 7.957 main.c File Reference

Demo program application source file.

```
#include "header.h"
```

Include dependency graph for ARMCM0\_STM32F0\_Nucleo\_F091RC\_GCC/Prog/main.c:



### Functions

- static void [Init](#) (void)  
*Initializes the microcontroller.*
- static void [SystemClock\\_Config](#) (void)  
*System Clock Configuration. This code was created by CubeMX and configures the system clock.*
- int [main](#) (void)  
*This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.*
- void [HAL\\_MspInit](#) (void)  
*Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).*
- void [HAL\\_MspDeInit](#) (void)  
*Deinitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level Deinitialization (GPIOs, clock, DMA, interrupt).*

### 7.957.1 Detailed Description

Demo program application source file.

### 7.957.2 Function Documentation

#### 7.957.2.1 HAL\_MspDeInit()

```
void HAL_MspDeInit (
 void)
```

Deinitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level Deinitialization (GPIOs, clock, DMA, interrupt).

##### Returns

none.

#### 7.957.2.2 HAL\_MspInit()

```
void HAL_MspInit (
 void)
```

Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).

##### Returns

none.

#### 7.957.2.3 Init()

```
static void Init (
 void) [static]
```

Initializes the microcontroller.

##### Returns

none.

Referenced by main().

#### 7.957.2.4 main()

```
int main (
 void)
```

This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.

##### Returns

Program exit code.

## 7.957.2.5 SystemClock\_Config()

```
static void SystemClock_Config (
 void) [static]
```

System Clock Configuration. This code was created by CubeMX and configures the system clock.

## Returns

none.

Referenced by Init().

## 7.958 main.c File Reference

Bootloader application source file.

```
#include "boot.h"
#include "stm32f0xx.h"
#include "stm32f0xx_ll_rcc.h"
#include "stm32f0xx_ll_bus.h"
#include "stm32f0xx_ll_system.h"
#include "stm32f0xx_ll_utils.h"
#include "stm32f0xx_ll_usart.h"
#include "stm32f0xx_ll_gpio.h"
```

Include dependency graph for ARMCM0\_STM32F0\_Nucleo\_F091RC\_IAR/Boot/main.c:



## Functions

- static void [Init](#) (void)  
*Initializes the microcontroller.*
- static void [SystemClock\\_Config](#) (void)  
*System Clock Configuration. This code was created by CubeMX and configures the system clock to match the configuration in the bootloader's configuration (blt\_conf.h), specifically the macros: BOOT\_CPU\_SYSTEM\_SPEED\_KHZ and BOOT\_CPU\_XTAL\_SPEED\_KHZ. Note that the Lower Layer drivers were selected in CubeMX for the RCC subsystem.*
- void [main](#) (void)  
*This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.*
- void [HAL\\_MspInit](#) (void)  
*Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).*
- void [HAL\\_MspDeInit](#) (void)  
*DeInitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level de-initialization (GPIOs, clock, DMA, interrupt).*

## 7.958.1 Detailed Description

Bootloader application source file.

## 7.958.2 Function Documentation

### 7.958.2.1 HAL\_MspDeInit()

```
void HAL_MspDeInit (
 void)
```

DeInitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level de-initialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

### 7.958.2.2 HAL\_MspInit()

```
void HAL_MspInit (
 void)
```

Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

### 7.958.2.3 Init()

```
static void Init (
 void) [static]
```

Initializes the microcontroller.

#### Returns

none.

Referenced by main().



#### 7.958.2.4 main()

```
void main (
 void)
```

This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.

##### Returns

none.

#### 7.958.2.5 SystemClock\_Config()

```
static void SystemClock_Config (
 void) [static]
```

System Clock Configuration. This code was created by CubeMX and configures the system clock to match the configuration in the bootloader's configuration (blt\_conf.h), specifically the macros: BOOT\_CPU\_SYSTEM\_SPEED\_KHZ and BOOT\_CPU\_XTAL\_SPEED\_KHZ. Note that the Lower Layer drivers were selected in CubeMX for the RCC subsystem.

##### Returns

none.

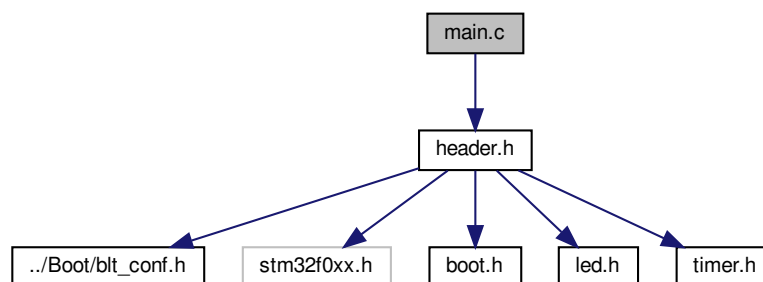
Referenced by Init().

## 7.959 main.c File Reference

Demo program application source file.

```
#include "header.h"
```

Include dependency graph for ARMCM0\_STM32F0\_Nucleo\_F091RC\_IAR/Prog/main.c:



## Functions

- static void `Init` (void)  
*Initializes the microcontroller.*
- static void `SystemClock_Config` (void)  
*System Clock Configuration. This code was created by CubeMX and configures the system clock.*
- void `main` (void)  
*This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.*
- void `HAL_MspInit` (void)  
*Initializes the Global MSP. This function is called from `HAL_Init()` function to perform system level initialization (GPIOs, clock, DMA, interrupt).*
- void `HAL_MspDeInit` (void)  
*Deinitializes the Global MSP. This function is called from `HAL_DeInit()` function to perform system level Deinitialization (GPIOs, clock, DMA, interrupt).*

### 7.959.1 Detailed Description

Demo program application source file.

### 7.959.2 Function Documentation

#### 7.959.2.1 `HAL_MspDeInit()`

```
void HAL_MspDeInit (
 void)
```

Deinitializes the Global MSP. This function is called from `HAL_DeInit()` function to perform system level Deinitialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

#### 7.959.2.2 `HAL_MspInit()`

```
void HAL_MspInit (
 void)
```

Initializes the Global MSP. This function is called from `HAL_Init()` function to perform system level initialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

### 7.959.2.3 Init()

```
static void Init (
 void) [static]
```

Initializes the microcontroller.

#### Returns

none.

Referenced by main().

### 7.959.2.4 main()

```
void main (
 void)
```

This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.

#### Returns

none.

### 7.959.2.5 SystemClock\_Config()

```
static void SystemClock_Config (
 void) [static]
```

System Clock Configuration. This code was created by CubeMX and configures the system clock.

#### Returns

none.

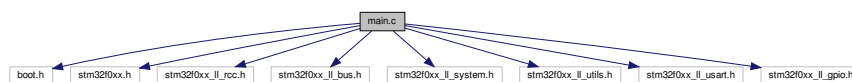
Referenced by Init().

## 7.960 main.c File Reference

Bootloader application source file.

```
#include "boot.h"
#include "stm32f0xx.h"
#include "stm32f0xx_ll_rcc.h"
#include "stm32f0xx_ll_bus.h"
#include "stm32f0xx_ll_system.h"
#include "stm32f0xx_ll_utils.h"
#include "stm32f0xx_ll_usart.h"
#include "stm32f0xx_ll_gpio.h"
```

Include dependency graph for ARMCM0\_STM32F0\_Nucleo\_F091RC\_Keil/Boot/main.c:



### Functions

- static void [Init](#) (void)  
*Initializes the microcontroller.*
- static void [SystemClock\\_Config](#) (void)  
*System Clock Configuration. This code was created by CubeMX and configures the system clock to match the configuration in the bootloader's configuration (blt\_conf.h), specifically the macros: BOOT\_CPU\_SYSTEM\_SPEED\_KHZ and BOOT\_CPU\_XTAL\_SPEED\_KHZ. Note that the Lower Layer drivers were selected in CubeMX for the RCC subsystem.*
- int [main](#) (void)  
*This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.*
- void [HAL\\_MspInit](#) (void)  
*Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).*
- void [HAL\\_MspDeInit](#) (void)  
*DeInitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level de-initialization (GPIOs, clock, DMA, interrupt).*

### 7.960.1 Detailed Description

Bootloader application source file.

### 7.960.2 Function Documentation

### 7.960.2.1 HAL\_MspDeInit()

```
void HAL_MspDeInit (
 void)
```

DeInitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level de-initialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

### 7.960.2.2 HAL\_MspInit()

```
void HAL_MspInit (
 void)
```

Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

### 7.960.2.3 Init()

```
static void Init (
 void) [static]
```

Initializes the microcontroller.

#### Returns

none.

Referenced by main().

### 7.960.2.4 main()

```
int main (
 void)
```

This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.

#### Returns

Program return code.

### 7.960.2.5 SystemClock\_Config()

```
static void SystemClock_Config (
 void) [static]
```

System Clock Configuration. This code was created by CubeMX and configures the system clock to match the configuration in the bootloader's configuration (blt\_conf.h), specifically the macros: BOOT\_CPU\_SYSTEM\_SPEED\_KHZ and BOOT\_CPU\_XTAL\_SPEED\_KHZ. Note that the Lower Layer drivers were selected in CubeMX for the RCC subsystem.

#### Returns

none.

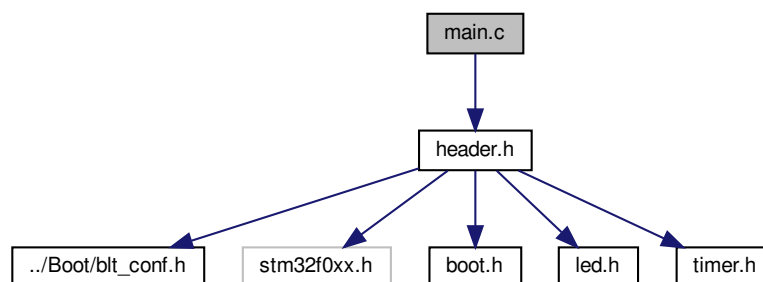
Referenced by Init().

## 7.961 main.c File Reference

Demo program application source file.

```
#include "header.h"
```

Include dependency graph for ARMCM0\_STM32F0\_Nucleo\_F091RC\_Keil/Prog/main.c:



## Functions

- static void [Init](#) (void)  
*Initializes the microcontroller.*
- static void [SystemClock\\_Config](#) (void)  
*System Clock Configuration. This code was created by CubeMX and configures the system clock.*
- int [main](#) (void)  
*This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.*
- void [HAL\\_MspInit](#) (void)  
*Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).*
- void [HAL\\_MspDeInit](#) (void)  
*Deinitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level Deinitialization (GPIOs, clock, DMA, interrupt).*

## 7.961.1 Detailed Description

Demo program application source file.

## 7.961.2 Function Documentation

### 7.961.2.1 HAL\_MspDeInit()

```
void HAL_MspDeInit (
 void)
```

Deinitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level Deinitialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

### 7.961.2.2 HAL\_MspInit()

```
void HAL_MspInit (
 void)
```

Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

### 7.961.2.3 Init()

```
static void Init (
 void) [static]
```

Initializes the microcontroller.

#### Returns

none.

Referenced by main().

#### 7.961.2.4 main()

```
int main (
 void)
```

This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.

##### Returns

Program exit code.

#### 7.961.2.5 SystemClock\_Config()

```
static void SystemClock_Config (
 void) [static]
```

System Clock Configuration. This code was created by CubeMX and configures the system clock.

##### Returns

none.

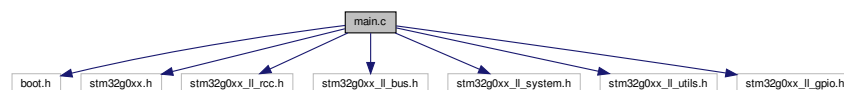
Referenced by Init().

## 7.962 main.c File Reference

Bootloader application source file.

```
#include "boot.h"
#include "stm32g0xx.h"
#include "stm32g0xx_ll_rcc.h"
#include "stm32g0xx_ll_bus.h"
#include "stm32g0xx_ll_system.h"
#include "stm32g0xx_ll_utils.h"
#include "stm32g0xx_ll_gpio.h"
```

Include dependency graph for ARMC00\_STM32G0\_Nucleo\_G071RB\_GCC/Boot/main.c:





## Functions

- static void `Init` (void)  
*Initializes the microcontroller.*
- static void `SystemClock_Config` (void)  
*System Clock Configuration. This code was created by CubeMX and configures the system clock to match the configuration in the bootloader's configuration (blt\_conf.h), specifically the macros: BOOT\_CPU\_SYSTEM\_SPEED\_KHZ and BOOT\_CPU\_XTAL\_SPEED\_KHZ. Note that the Lower Layer drivers were selected in CubeMX for the RCC subsystem.*
- int `main` (void)  
*This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.*
- void `HAL_MspInit` (void)  
*Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).*
- void `HAL_MspDeInit` (void)  
*DeInitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level de-initialization (GPIOs, clock, DMA, interrupt).*

### 7.962.1 Detailed Description

Bootloader application source file.

### 7.962.2 Function Documentation

#### 7.962.2.1 HAL\_MspDeInit()

```
void HAL_MspDeInit (
 void)
```

DeInitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level de-initialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

#### 7.962.2.2 HAL\_MspInit()

```
void HAL_MspInit (
 void)
```

Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

### 7.962.2.3 Init()

```
static void Init (
 void) [static]
```

Initializes the microcontroller.

#### Returns

none.

Referenced by main().

### 7.962.2.4 main()

```
int main (
 void)
```

This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.

#### Returns

Program return code.

### 7.962.2.5 SystemClock\_Config()

```
static void SystemClock_Config (
 void) [static]
```

System Clock Configuration. This code was created by CubeMX and configures the system clock to match the configuration in the bootloader's configuration (blt\_conf.h), specifically the macros: BOOT\_CPU\_SYSTEM\_SPEED\_KHZ and BOOT\_CPU\_XTAL\_SPEED\_KHZ. Note that the Lower Layer drivers were selected in CubeMX for the RCC subsystem.

#### Returns

none.

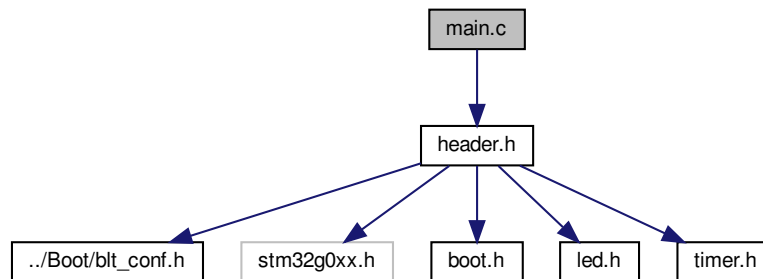
Referenced by Init().

## 7.963 main.c File Reference

Demo program application source file.

```
#include "header.h"
```

Include dependency graph for ARMCM0\_STM32G0\_Nucleo\_G071RB\_GCC/Prog/main.c:



### Functions

- static void [Init](#) (void)  
*Initializes the microcontroller.*
- static void [SystemClock\\_Config](#) (void)  
*System Clock Configuration. This code was created by CubeMX and configures the system clock.*
- static void [VectorBase\\_Config](#) (void)  
*Vector base address configuration. It should no longer be at the start of flash memory but moved forward because the first part of flash is reserved for the bootloader. Note that this is already done by the bootloader before starting this program. Unfortunately, function SystemInit() overwrites this change again.*
- int [main](#) (void)  
*This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.*
- void [HAL\\_MspInit](#) (void)  
*Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).*
- void [HAL\\_MspDeInit](#) (void)  
*Deinitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level Deinitialization (GPIOs, clock, DMA, interrupt).*

### 7.963.1 Detailed Description

Demo program application source file.

### 7.963.2 Function Documentation

#### 7.963.2.1 HAL\_MspDeInit()

```
void HAL_MspDeInit (
 void)
```

Deinitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level Deinitialization (GPIOs, clock, DMA, interrupt).

##### Returns

none.

#### 7.963.2.2 HAL\_MspInit()

```
void HAL_MspInit (
 void)
```

Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).

##### Returns

none.

#### 7.963.2.3 Init()

```
static void Init (
 void) [static]
```

Initializes the microcontroller.

##### Returns

none.

Referenced by main().

#### 7.963.2.4 main()

```
int main (
 void)
```

This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.

##### Returns

Program return code.

### 7.963.2.5 SystemClock\_Config()

```
static void SystemClock_Config (
 void) [static]
```

System Clock Configuration. This code was created by CubeMX and configures the system clock.

#### Returns

none.

Referenced by Init().

### 7.963.2.6 VectorBase\_Config()

```
static void VectorBase_Config (
 void) [static]
```

Vector base address configuration. It should no longer be at the start of flash memory but moved forward because the first part of flash is reserved for the bootloader. Note that this is already done by the bootloader before starting this program. Unfortunately, function SystemInit() overwrites this change again.

#### Returns

none.

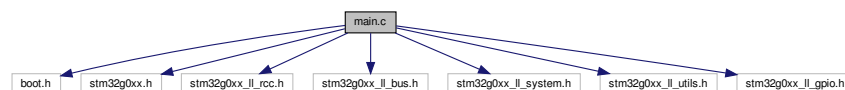
Referenced by Init().

## 7.964 main.c File Reference

Bootloader application source file.

```
#include "boot.h"
#include "stm32g0xx.h"
#include "stm32g0xx_ll_rcc.h"
#include "stm32g0xx_ll_bus.h"
#include "stm32g0xx_ll_system.h"
#include "stm32g0xx_ll_utils.h"
#include "stm32g0xx_ll_gpio.h"
```

Include dependency graph for ARMC00\_STM32G0\_Nucleo\_G071RB\_IAR/Boot/main.c:



## Functions

- static void `Init` (void)  
*Initializes the microcontroller.*
- static void `SystemClock_Config` (void)  
*System Clock Configuration. This code was created by CubeMX and configures the system clock to match the configuration in the bootloader's configuration (blt\_conf.h), specifically the macros: BOOT\_CPU\_SYSTEM\_SPEED\_KHZ and BOOT\_CPU\_XTAL\_SPEED\_KHZ. Note that the Lower Layer drivers were selected in CubeMX for the RCC subsystem.*
- void `main` (void)  
*This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.*
- void `HAL_MspInit` (void)  
*Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).*
- void `HAL_MspDeInit` (void)  
*DeInitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level de-initialization (GPIOs, clock, DMA, interrupt).*

### 7.964.1 Detailed Description

Bootloader application source file.

### 7.964.2 Function Documentation

#### 7.964.2.1 HAL\_MspDeInit()

```
void HAL_MspDeInit (
 void)
```

DeInitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level de-initialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

#### 7.964.2.2 HAL\_MspInit()

```
void HAL_MspInit (
 void)
```

Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

### 7.964.2.3 Init()

```
static void Init (
 void) [static]
```

Initializes the microcontroller.

#### Returns

none.

Referenced by main().

### 7.964.2.4 main()

```
void main (
 void)
```

This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.

#### Returns

none.

### 7.964.2.5 SystemClock\_Config()

```
static void SystemClock_Config (
 void) [static]
```

System Clock Configuration. This code was created by CubeMX and configures the system clock to match the configuration in the bootloader's configuration (blt\_conf.h), specifically the macros: BOOT\_CPU\_SYSTEM\_SPEED\_KHZ and BOOT\_CPU\_XTAL\_SPEED\_KHZ. Note that the Lower Layer drivers were selected in CubeMX for the RCC subsystem.

#### Returns

none.

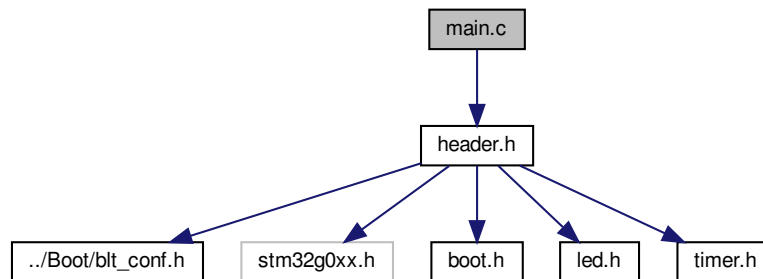
Referenced by Init().

## 7.965 main.c File Reference

Demo program application source file.

```
#include "header.h"
```

Include dependency graph for ARMC00\_STM32G0\_Nucleo\_G071RB\_IAR/Prog/main.c:



### Functions

- static void [Init](#) (void)  
*Initializes the microcontroller.*
- static void [SystemClock\\_Config](#) (void)  
*System Clock Configuration. This code was created by CubeMX and configures the system clock.*
- static void [VectorBase\\_Config](#) (void)  
*Vector base address configuration. It should no longer be at the start of flash memory but moved forward because the first part of flash is reserved for the bootloader. Note that this is already done by the bootloader before starting this program. Unfortunately, function SystemInit() overwrites this change again.*
- void [main](#) (void)  
*This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.*
- void [HAL\\_MspInit](#) (void)  
*Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).*
- void [HAL\\_MspDeInit](#) (void)  
*Deinitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level Deinitialization (GPIOs, clock, DMA, interrupt).*

### 7.965.1 Detailed Description

Demo program application source file.

### 7.965.2 Function Documentation



### 7.965.2.1 HAL\_MspDeInit()

```
void HAL_MspDeInit (
 void)
```

Deinitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level Deinitialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

### 7.965.2.2 HAL\_MspInit()

```
void HAL_MspInit (
 void)
```

Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

### 7.965.2.3 Init()

```
static void Init (
 void) [static]
```

Initializes the microcontroller.

#### Returns

none.

Referenced by main().

### 7.965.2.4 main()

```
void main (
 void)
```

This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.

#### Returns

none.

### 7.965.2.5 SystemClock\_Config()

```
static void SystemClock_Config (
 void) [static]
```

System Clock Configuration. This code was created by CubeMX and configures the system clock.

#### Returns

none.

Referenced by Init().

### 7.965.2.6 VectorBase\_Config()

```
static void VectorBase_Config (
 void) [static]
```

Vector base address configuration. It should no longer be at the start of flash memory but moved forward because the first part of flash is reserved for the bootloader. Note that this is already done by the bootloader before starting this program. Unfortunately, function SystemInit() overwrites this change again.

#### Returns

none.

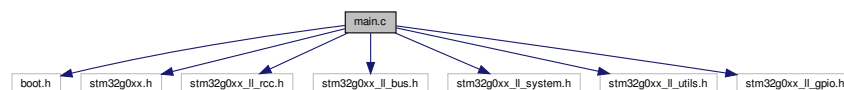
Referenced by Init().

## 7.966 main.c File Reference

Bootloader application source file.

```
#include "boot.h"
#include "stm32g0xx.h"
#include "stm32g0xx_ll_rcc.h"
#include "stm32g0xx_ll_bus.h"
#include "stm32g0xx_ll_system.h"
#include "stm32g0xx_ll_utils.h"
#include "stm32g0xx_ll_gpio.h"
```

Include dependency graph for ARMC00\_STM32G0\_Nucleo\_G071RB\_Keil/Boot/main.c:



## Functions

- static void `Init` (void)  
*Initializes the microcontroller.*
- static void `SystemClock_Config` (void)  
*System Clock Configuration. This code was created by CubeMX and configures the system clock to match the configuration in the bootloader's configuration (blt\_conf.h), specifically the macros: BOOT\_CPU\_SYSTEM\_SPEED\_KHZ and BOOT\_CPU\_XTAL\_SPEED\_KHZ. Note that the Lower Layer drivers were selected in CubeMX for the RCC subsystem.*
- int `main` (void)  
*This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.*
- void `HAL_MspInit` (void)  
*Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).*
- void `HAL_MspDeInit` (void)  
*DeInitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level de-initialization (GPIOs, clock, DMA, interrupt).*

### 7.966.1 Detailed Description

Bootloader application source file.

### 7.966.2 Function Documentation

#### 7.966.2.1 HAL\_MspDeInit()

```
void HAL_MspDeInit (
 void)
```

DeInitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level de-initialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

#### 7.966.2.2 HAL\_MspInit()

```
void HAL_MspInit (
 void)
```

Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

### 7.966.2.3 Init()

```
static void Init (
 void) [static]
```

Initializes the microcontroller.

#### Returns

none.

Referenced by main().

### 7.966.2.4 main()

```
int main (
 void)
```

This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.

#### Returns

Program return code.

### 7.966.2.5 SystemClock\_Config()

```
static void SystemClock_Config (
 void) [static]
```

System Clock Configuration. This code was created by CubeMX and configures the system clock to match the configuration in the bootloader's configuration (blt\_conf.h), specifically the macros: BOOT\_CPU\_SYSTEM\_SPEED\_KHZ and BOOT\_CPU\_XTAL\_SPEED\_KHZ. Note that the Lower Layer drivers were selected in CubeMX for the RCC subsystem.

#### Returns

none.

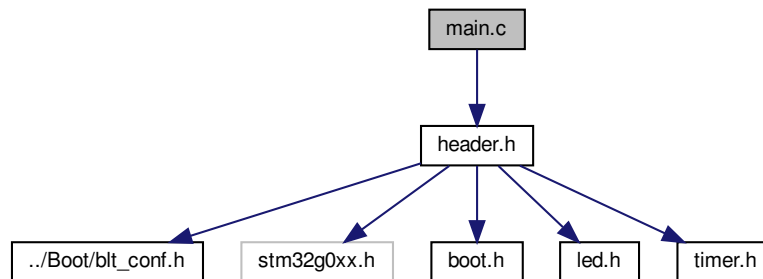
Referenced by Init().

## 7.967 main.c File Reference

Demo program application source file.

```
#include "header.h"
```

Include dependency graph for ARMCM0\_STM32G0\_Nucleo\_G071RB\_Keil/Prog/main.c:



### Functions

- static void [Init](#) (void)  
*Initializes the microcontroller.*
- static void [SystemClock\\_Config](#) (void)  
*System Clock Configuration. This code was created by CubeMX and configures the system clock.*
- static void [VectorBase\\_Config](#) (void)  
*Vector base address configuration. It should no longer be at the start of flash memory but moved forward because the first part of flash is reserved for the bootloader. Note that this is already done by the bootloader before starting this program. Unfortunately, function SystemInit() overwrites this change again.*
- int [main](#) (void)  
*This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.*
- void [HAL\\_MspInit](#) (void)  
*Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).*
- void [HAL\\_MspDeInit](#) (void)  
*Deinitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level Deinitialization (GPIOs, clock, DMA, interrupt).*

### 7.967.1 Detailed Description

Demo program application source file.

### 7.967.2 Function Documentation

#### 7.967.2.1 HAL\_MspDeInit()

```
void HAL_MspDeInit (
 void)
```

Deinitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level Deinitialization (GPIOs, clock, DMA, interrupt).

##### Returns

none.

#### 7.967.2.2 HAL\_MspInit()

```
void HAL_MspInit (
 void)
```

Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).

##### Returns

none.

#### 7.967.2.3 Init()

```
static void Init (
 void) [static]
```

Initializes the microcontroller.

##### Returns

none.

Referenced by main().

#### 7.967.2.4 main()

```
int main (
 void)
```

This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.

##### Returns

Program return code.

### 7.967.2.5 SystemClock\_Config()

```
static void SystemClock_Config (
 void) [static]
```

System Clock Configuration. This code was created by CubeMX and configures the system clock.

#### Returns

none.

Referenced by Init().

### 7.967.2.6 VectorBase\_Config()

```
static void VectorBase_Config (
 void) [static]
```

Vector base address configuration. It should no longer be at the start of flash memory but moved forward because the first part of flash is reserved for the bootloader. Note that this is already done by the bootloader before starting this program. Unfortunately, function SystemInit() overwrites this change again.

#### Returns

none.

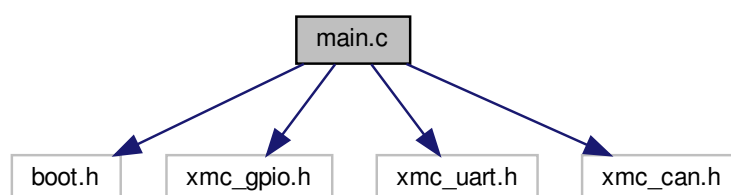
Referenced by Init().

## 7.968 main.c File Reference

Bootloader application source file.

```
#include "boot.h"
#include "xmc_gpio.h"
#include "xmc_uart.h"
#include "xmc_can.h"
```

Include dependency graph for ARMCM0\_XMC1\_XMC1400\_Boot\_Kit\_GCC/Boot/main.c:



## Functions

- static void [PostInit](#) (void)

*Post initialization of the microcontroller. Contains all initialization code that should run after the bootloader's core was initialized.*

- int [main](#) (void)

*This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.*

### 7.968.1 Detailed Description

Bootloader application source file.

### 7.968.2 Function Documentation

#### 7.968.2.1 [main\(\)](#)

```
int main (
 void)
```

This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.

#### Returns

Program exit code.

#### 7.968.2.2 [PostInit\(\)](#)

```
static void PostInit (
 void) [static]
```

Post initialization of the microcontroller. Contains all initialization code that should run after the bootloader's core was initialized.

#### Returns

none.

Referenced by [main\(\)](#).

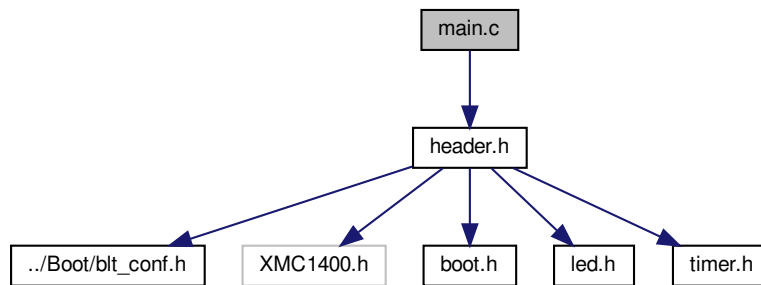


## 7.969 main.c File Reference

Demo program application source file.

```
#include "header.h"
```

Include dependency graph for ARMCM0\_XMC1\_XMC1400\_Boot\_Kit\_GCC/Prog/main.c:



### Functions

- static void `Init` (void)  
*Initializes the microcontroller.*
- int `main` (void)  
*This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.*

### 7.969.1 Detailed Description

Demo program application source file.

### 7.969.2 Function Documentation

#### 7.969.2.1 Init()

```
static void Init (
 void) [static]
```

Initializes the microcontroller.

#### Returns

none.

Referenced by `main()`.

### 7.969.2.2 main()

```
int main (
 void)
```

This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.

#### Returns

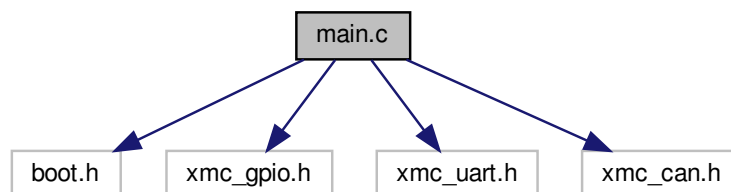
Program exit code.

## 7.970 main.c File Reference

Bootloader application source file.

```
#include "boot.h"
#include "xmc_gpio.h"
#include "xmc_uart.h"
#include "xmc_can.h"
```

Include dependency graph for ARMC0\_XMC1\_XMC1400\_Boot\_Kit\_IAR/Boot/main.c:



## Functions

- static void **Init** (void)  
*Initializes the microcontroller.*
- static void **PostInit** (void)  
*Post initialization of the microcontroller. Contains all initialization code that should run after the bootloader's core was initialized.*
- void **main** (void)  
*This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.*

### 7.970.1 Detailed Description

Bootloader application source file.

## 7.970.2 Function Documentation

### 7.970.2.1 Init()

```
static void Init (
 void) [static]
```

Initializes the microcontroller.

#### Returns

none.

Referenced by main().

### 7.970.2.2 main()

```
void main (
 void)
```

This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.

#### Returns

None.

### 7.970.2.3 PostInit()

```
static void PostInit (
 void) [static]
```

Post initialization of the microcontroller. Contains all initialization code that should run after the bootloader's core was initialized.

#### Returns

none.

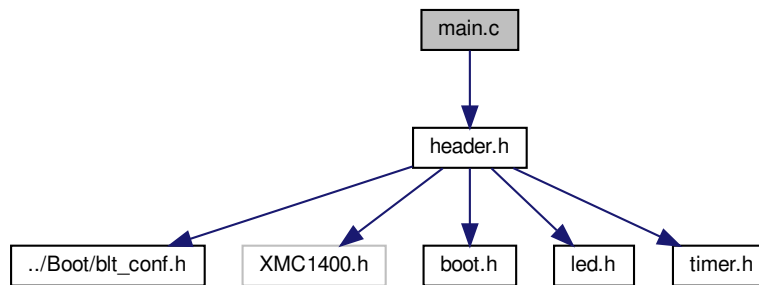
Referenced by main().

## 7.971 main.c File Reference

Demo program application source file.

```
#include "header.h"
```

Include dependency graph for ARMCM0\_XMC1\_XMC1400\_Boot\_Kit\_IAR/Prog/main.c:



### Functions

- static void `Init` (void)  
*Initializes the microcontroller.*
- void `main` (void)  
*This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.*

### 7.971.1 Detailed Description

Demo program application source file.

### 7.971.2 Function Documentation

#### 7.971.2.1 Init()

```
static void Init (
 void) [static]
```

Initializes the microcontroller.

#### Returns

none.

Referenced by `main()`.

## 7.971.2.2 main()

```
void main (
 void)
```

This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.

## Returns

none.

## 7.972 main.c File Reference

Bootloader application source file.

```
#include "boot.h"
#include "stm32l5xx.h"
#include "stm32l5xx_ll_pwr.h"
#include "stm32l5xx_ll_rcc.h"
#include "stm32l5xx_ll_bus.h"
#include "stm32l5xx_ll_system.h"
#include "stm32l5xx_ll_utils.h"
#include "stm32l5xx_ll_gpio.h"
```

Include dependency graph for ARMCM33\_STM32L5\_Nucleo\_L552ZE\_GCC/Boot/main.c:



## Functions

- static void [Init](#) (void)  
*Initializes the microcontroller.*
- static void [SystemClock\\_Config](#) (void)  
*System Clock Configuration. This code was created by CubeMX and configures the system clock to match the configuration in the bootloader's configuration (blt\_conf.h), specifically the macros: BOOT\_CPU\_SYSTEM\_SPEED\_KHZ and BOOT\_CPU\_XTAL\_SPEED\_KHZ. Note that the Lower Layer drivers were selected in CubeMX for the RCC subsystem.*
- int [main](#) (void)  
*This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.*
- void [HAL\\_MspInit](#) (void)  
*Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).*
- void [HAL\\_MspDeInit](#) (void)  
*DeInitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level de-initialization (GPIOs, clock, DMA, interrupt).*

### 7.972.1 Detailed Description

Bootloader application source file.

### 7.972.2 Function Documentation

#### 7.972.2.1 HAL\_MspDeInit()

```
void HAL_MspDeInit (
 void)
```

DeInitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level de-initialization (GPIOs, clock, DMA, interrupt).

##### Returns

none.

#### 7.972.2.2 HAL\_MspInit()

```
void HAL_MspInit (
 void)
```

Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).

##### Returns

none.

#### 7.972.2.3 Init()

```
static void Init (
 void) [static]
```

Initializes the microcontroller.

##### Returns

none.

Referenced by main().

#### 7.972.2.4 main()

```
int main (
 void)
```

This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.

##### Returns

Program return code.

#### 7.972.2.5 SystemClock\_Config()

```
static void SystemClock_Config (
 void) [static]
```

System Clock Configuration. This code was created by CubeMX and configures the system clock to match the configuration in the bootloader's configuration (blt\_conf.h), specifically the macros: BOOT\_CPU\_SYSTEM\_SPEED\_KHZ and BOOT\_CPU\_XTAL\_SPEED\_KHZ. Note that the Lower Layer drivers were selected in CubeMX for the RCC subsystem.

##### Returns

none.

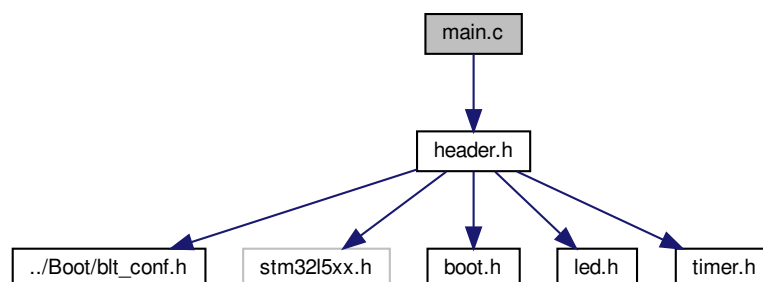
Referenced by Init().

## 7.973 main.c File Reference

Demo program application source file.

```
#include "header.h"
```

Include dependency graph for ARMCM33\_STM32L5\_Nucleo\_L552ZE\_GCC/Prog/main.c:



## Functions

- static void [Init](#) (void)  
*Initializes the microcontroller.*
- static void [SystemClock\\_Config](#) (void)  
*System Clock Configuration. This code was created by CubeMX and configures the system clock.*
- static void [VectorBase\\_Config](#) (void)  
*Vector base address configuration. It should no longer be at the start of flash memory but moved forward because the first part of flash is reserved for the bootloader. Note that this is already done by the bootloader before starting this program. Unfortunately, function SystemInit() overwrites this change again.*
- int [main](#) (void)  
*This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.*
- void [HAL\\_MspInit](#) (void)  
*Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).*
- void [HAL\\_MspDeInit](#) (void)  
*Deinitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level Deinitialization (GPIOs, clock, DMA, interrupt).*

### 7.973.1 Detailed Description

Demo program application source file.

### 7.973.2 Function Documentation

#### 7.973.2.1 HAL\_MspDeInit()

```
void HAL_MspDeInit (
 void)
```

Deinitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level Deinitialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

#### 7.973.2.2 HAL\_MspInit()

```
void HAL_MspInit (
 void)
```

Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.



### 7.973.2.3 Init()

```
static void Init (
 void) [static]
```

Initializes the microcontroller.

#### Returns

none.

Referenced by main().

### 7.973.2.4 main()

```
int main (
 void)
```

This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.

#### Returns

Program exit code.

### 7.973.2.5 SystemClock\_Config()

```
static void SystemClock_Config (
 void) [static]
```

System Clock Configuration. This code was created by CubeMX and configures the system clock.

#### Returns

none.

Referenced by Init().

### 7.973.2.6 VectorBase\_Config()

```
static void VectorBase_Config (
 void) [static]
```

Vector base address configuration. It should no longer be at the start of flash memory but moved forward because the first part of flash is reserved for the bootloader. Note that this is already done by the bootloader before starting this program. Unfortunately, function SystemInit() overwrites this change again.

#### Returns

none.

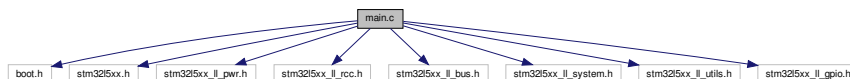
Referenced by Init().

## 7.974 main.c File Reference

Bootloader application source file.

```
#include "boot.h"
#include "stm32l5xx.h"
#include "stm32l5xx_ll_pwr.h"
#include "stm32l5xx_ll_rcc.h"
#include "stm32l5xx_ll_bus.h"
#include "stm32l5xx_ll_system.h"
#include "stm32l5xx_ll_utils.h"
#include "stm32l5xx_ll_gpio.h"
```

Include dependency graph for ARMC33\_STM32L5\_Nucleo\_L552ZE\_IAR/Boot/main.c:



## Functions

- static void [Init](#) (void)  
*Initializes the microcontroller.*
- static void [SystemClock\\_Config](#) (void)  
*System Clock Configuration. This code was created by CubeMX and configures the system clock to match the configuration in the bootloader's configuration (blt\_conf.h), specifically the macros: BOOT\_CPU\_SYSTEM\_SPEED\_KHZ and BOOT\_CPU\_XTAL\_SPEED\_KHZ. Note that the Lower Layer drivers were selected in CubeMX for the RCC subsystem.*
- void [main](#) (void)  
*This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.*
- void [HAL\\_MspInit](#) (void)  
*Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).*
- void [HAL\\_MspDeInit](#) (void)  
*DeInitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level de-initialization (GPIOs, clock, DMA, interrupt).*

## 7.974.1 Detailed Description

Bootloader application source file.

## 7.974.2 Function Documentation

### 7.974.2.1 HAL\_MspDeInit()

```
void HAL_MspDeInit (
 void)
```

DeInitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level de-initialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

### 7.974.2.2 HAL\_MspInit()

```
void HAL_MspInit (
 void)
```

Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

### 7.974.2.3 Init()

```
static void Init (
 void) [static]
```

Initializes the microcontroller.

#### Returns

none.

Referenced by main().

#### 7.974.2.4 main()

```
void main (
 void)
```

This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.

##### Returns

none.

#### 7.974.2.5 SystemClock\_Config()

```
static void SystemClock_Config (
 void) [static]
```

System Clock Configuration. This code was created by CubeMX and configures the system clock to match the configuration in the bootloader's configuration (blt\_conf.h), specifically the macros: BOOT\_CPU\_SYSTEM\_SPEED\_KHZ and BOOT\_CPU\_XTAL\_SPEED\_KHZ. Note that the Lower Layer drivers were selected in CubeMX for the RCC subsystem.

##### Returns

none.

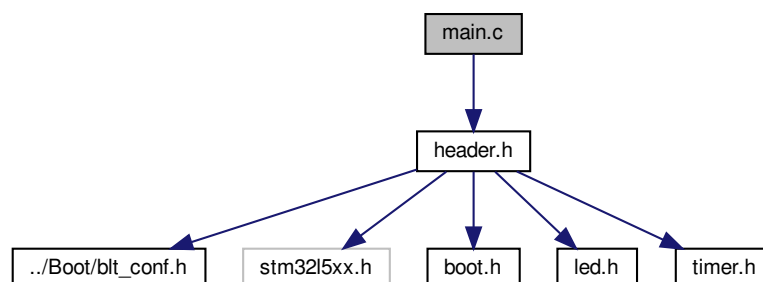
Referenced by Init().

## 7.975 main.c File Reference

Demo program application source file.

```
#include "header.h"
```

Include dependency graph for ARMCM33\_STM32L5\_Nucleo\_L552ZE\_IAR/Prog/main.c:



## Functions

- static void `Init` (void)  
*Initializes the microcontroller.*
- static void `SystemClock_Config` (void)  
*System Clock Configuration. This code was created by CubeMX and configures the system clock.*
- static void `VectorBase_Config` (void)  
*Vector base address configuration. It should no longer be at the start of flash memory but moved forward because the first part of flash is reserved for the bootloader. Note that this is already done by the bootloader before starting this program. Unfortunately, function `SystemInit()` overwrites this change again.*
- void `main` (void)  
*This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.*
- void `HAL_MspInit` (void)  
*Initializes the Global MSP. This function is called from `HAL_Init()` function to perform system level initialization (GPIOs, clock, DMA, interrupt).*
- void `HAL_MspDeInit` (void)  
*Deinitializes the Global MSP. This function is called from `HAL_DeInit()` function to perform system level Deinitialization (GPIOs, clock, DMA, interrupt).*

### 7.975.1 Detailed Description

Demo program application source file.

### 7.975.2 Function Documentation

#### 7.975.2.1 HAL\_MspDeInit()

```
void HAL_MspDeInit (
 void)
```

Deinitializes the Global MSP. This function is called from `HAL_DeInit()` function to perform system level Deinitialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

#### 7.975.2.2 HAL\_MspInit()

```
void HAL_MspInit (
 void)
```

Initializes the Global MSP. This function is called from `HAL_Init()` function to perform system level initialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

### 7.975.2.3 Init()

```
static void Init (
 void) [static]
```

Initializes the microcontroller.

#### Returns

none.

Referenced by main().

### 7.975.2.4 main()

```
void main (
 void)
```

This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.

#### Returns

none.

### 7.975.2.5 SystemClock\_Config()

```
static void SystemClock_Config (
 void) [static]
```

System Clock Configuration. This code was created by CubeMX and configures the system clock.

#### Returns

none.

Referenced by Init().

## 7.975.2.6 VectorBase\_Config()

```
static void VectorBase_Config (
 void) [static]
```

Vector base address configuration. It should no longer be at the start of flash memory but moved forward because the first part of flash is reserved for the bootloader. Note that this is already done by the bootloader before starting this program. Unfortunately, function SystemInit() overwrites this change again.

## Returns

none.

Referenced by Init().

## 7.976 main.c File Reference

Bootloader application source file.

```
#include "boot.h"
#include "stm32l5xx.h"
#include "stm32l5xx_ll_pwr.h"
#include "stm32l5xx_ll_rcc.h"
#include "stm32l5xx_ll_bus.h"
#include "stm32l5xx_ll_system.h"
#include "stm32l5xx_ll_utils.h"
#include "stm32l5xx_ll_gpio.h"
```

Include dependency graph for ARMC33\_STM32L5\_Nucleo\_L552ZE\_Keil/Boot/main.c:



## Functions

- static void [Init](#) (void)  
*Initializes the microcontroller.*
- static void [SystemClock\\_Config](#) (void)  
*System Clock Configuration. This code was created by CubeMX and configures the system clock to match the configuration in the bootloader's configuration (blt\_conf.h), specifically the macros: BOOT\_CPU\_SYSTEM\_SPEED\_KHZ and BOOT\_CPU\_XTAL\_SPEED\_KHZ. Note that the Lower Layer drivers were selected in CubeMX for the RCC subsystem.*
- int [main](#) (void)  
*This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.*
- void [HAL\\_MspInit](#) (void)  
*Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).*
- void [HAL\\_MspDeInit](#) (void)  
*DeInitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level de-initialization (GPIOs, clock, DMA, interrupt).*

## 7.976.1 Detailed Description

Bootloader application source file.

## 7.976.2 Function Documentation

### 7.976.2.1 HAL\_MspDeInit()

```
void HAL_MspDeInit (
 void)
```

DeInitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level de-initialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

### 7.976.2.2 HAL\_MspInit()

```
void HAL_MspInit (
 void)
```

Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

### 7.976.2.3 Init()

```
static void Init (
 void) [static]
```

Initializes the microcontroller.

#### Returns

none.

Referenced by main().



### 7.976.2.4 main()

```
int main (
 void)
```

This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.

#### Returns

Program return code.

### 7.976.2.5 SystemClock\_Config()

```
static void SystemClock_Config (
 void) [static]
```

System Clock Configuration. This code was created by CubeMX and configures the system clock to match the configuration in the bootloader's configuration (blt\_conf.h), specifically the macros: BOOT\_CPU\_SYSTEM\_SPEED\_KHZ and BOOT\_CPU\_XTAL\_SPEED\_KHZ. Note that the Lower Layer drivers were selected in CubeMX for the RCC subsystem.

#### Returns

none.

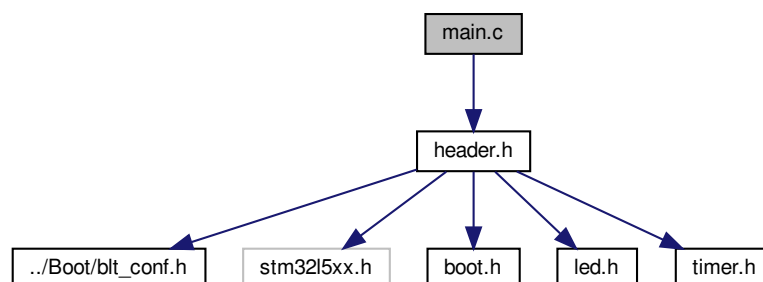
Referenced by Init().

## 7.977 main.c File Reference

Demo program application source file.

```
#include "header.h"
```

Include dependency graph for ARMCM33\_STM32L5\_Nucleo\_L552ZE\_Keil/Prog/main.c:



## Functions

- static void [Init](#) (void)  
*Initializes the microcontroller.*
- static void [SystemClock\\_Config](#) (void)  
*System Clock Configuration. This code was created by CubeMX and configures the system clock.*
- static void [VectorBase\\_Config](#) (void)  
*Vector base address configuration. It should no longer be at the start of flash memory but moved forward because the first part of flash is reserved for the bootloader. Note that this is already done by the bootloader before starting this program. Unfortunately, function SystemInit() overwrites this change again.*
- int [main](#) (void)  
*This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.*
- void [HAL\\_MspInit](#) (void)  
*Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).*
- void [HAL\\_MspDeInit](#) (void)  
*Deinitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level Deinitialization (GPIOs, clock, DMA, interrupt).*

### 7.977.1 Detailed Description

Demo program application source file.

### 7.977.2 Function Documentation

#### 7.977.2.1 HAL\_MspDeInit()

```
void HAL_MspDeInit (
 void)
```

Deinitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level Deinitialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

#### 7.977.2.2 HAL\_MspInit()

```
void HAL_MspInit (
 void)
```

Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

### 7.977.2.3 Init()

```
static void Init (
 void) [static]
```

Initializes the microcontroller.

#### Returns

none.

Referenced by main().

### 7.977.2.4 main()

```
int main (
 void)
```

This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.

#### Returns

Program exit code.

### 7.977.2.5 SystemClock\_Config()

```
static void SystemClock_Config (
 void) [static]
```

System Clock Configuration. This code was created by CubeMX and configures the system clock.

#### Returns

none.

Referenced by Init().

### 7.977.2.6 VectorBase\_Config()

```
static void VectorBase_Config (
 void) [static]
```

Vector base address configuration. It should no longer be at the start of flash memory but moved forward because the first part of flash is reserved for the bootloader. Note that this is already done by the bootloader before starting this program. Unfortunately, function SystemInit() overwrites this change again.

#### Returns

none.

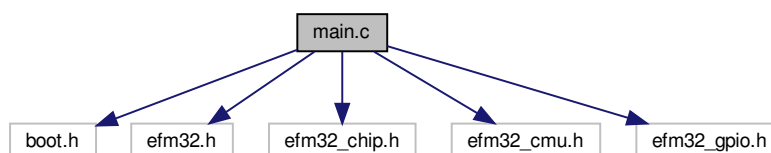
Referenced by Init().

## 7.978 main.c File Reference

Bootloader application source file.

```
#include "boot.h"
#include "efm32.h"
#include "efm32_chip.h"
#include "efm32_cmu.h"
#include "efm32_gpio.h"
```

Include dependency graph for ARMCM3\_EFM32\_Olimex\_EM32G880F128STK\_GCC/Boot/main.c:



## Functions

- static void [Init](#) (void)  
*Initializes the microcontroller.*
- int [main](#) (void)  
*This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.*

### 7.978.1 Detailed Description

Bootloader application source file.

## 7.978.2 Function Documentation

### 7.978.2.1 Init()

```
static void Init (
 void) [static]
```

Initializes the microcontroller.

#### Returns

none.

Referenced by main().

### 7.978.2.2 main()

```
int main (
 void)
```

This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.

#### Returns

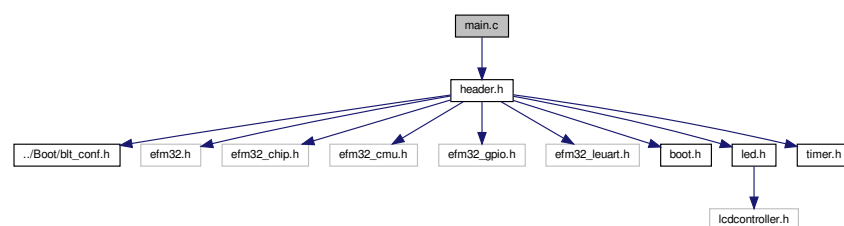
Program return code.

## 7.979 main.c File Reference

Demo program application source file.

```
#include "header.h"
```

Include dependency graph for ARMCM3\_EFM32\_Olimex\_EM32G880F128STK\_GCC/Prog/main.c:



## Functions

- static void `Init` (void)  
*Initializes the microcontroller.*
- int `main` (void)  
*This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.*

### 7.979.1 Detailed Description

Demo program application source file.

### 7.979.2 Function Documentation

#### 7.979.2.1 `Init()`

```
static void Init (
 void) [static]
```

Initializes the microcontroller.

#### Returns

none.

Referenced by `main()`.

#### 7.979.2.2 `main()`

```
int main (
 void)
```

This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.

#### Returns

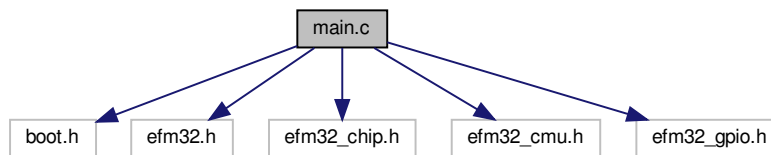
Program return code.

## 7.980 main.c File Reference

Bootloader application source file.

```
#include "boot.h"
#include "efm32.h"
#include "efm32_chip.h"
#include "efm32_cmu.h"
#include "efm32_gpio.h"
```

Include dependency graph for ARMCM3\_EFM32\_Olimex\_EM32G880F128STK\_IAR/Boot/main.c:



### Functions

- static void `Init` (void)  
*Initializes the microcontroller.*
- void `main` (void)  
*This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.*

### 7.980.1 Detailed Description

Bootloader application source file.

### 7.980.2 Function Documentation

#### 7.980.2.1 Init()

```
static void Init (
 void) [static]
```

Initializes the microcontroller.

#### Returns

none.

Referenced by `main()`.

### 7.980.2.2 main()

```
void main (
 void)
```

This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.

#### Returns

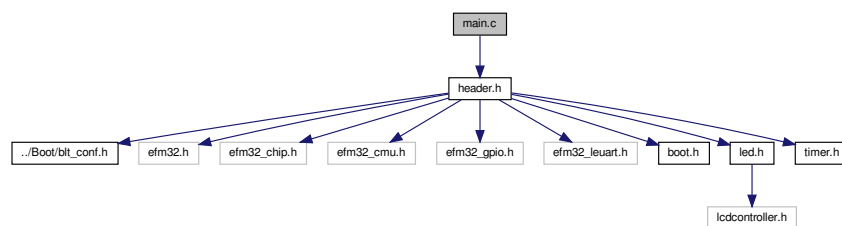
Program return code.

## 7.981 main.c File Reference

Demo program application source file.

```
#include "header.h"
```

Include dependency graph for ARMC32\_EFM32\_Olimex\_EM32G880F128STK\_IAR/Prog/main.c:



## Functions

- static void [Init](#) (void)  
*Initializes the microcontroller.*
- void [main](#) (void)  
*This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.*

### 7.981.1 Detailed Description

Demo program application source file.

### 7.981.2 Function Documentation



## 7.981.2.1 Init()

```
static void Init (
 void) [static]
```

Initializes the microcontroller.

## Returns

none.

Referenced by main().

## 7.981.2.2 main()

```
void main (
 void)
```

This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.

## Returns

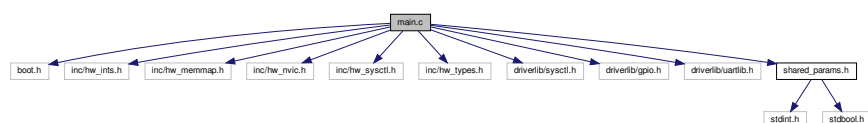
none.

## 7.982 main.c File Reference

Bootloader application source file.

```
#include "boot.h"
#include "inc/hw_ints.h"
#include "inc/hw_memmap.h"
#include "inc/hw_nvic.h"
#include "inc/hw_sysctl.h"
#include "inc/hw_types.h"
#include "driverlib/sysctl.h"
#include "driverlib/gpio.h"
#include "driverlib/uartlib.h"
#include "shared_params.h"
```

Include dependency graph for ARMC3\_LM3S\_EK\_LM3S6965\_GCC/Boot/main.c:



## Functions

- static void `Init` (void)  
*Initializes the microcontroller.*
- int `main` (void)  
*This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.*

### 7.982.1 Detailed Description

Bootloader application source file.

### 7.982.2 Function Documentation

#### 7.982.2.1 `Init()`

```
static void Init (
 void) [static]
```

Initializes the microcontroller.

#### Returns

none.

Referenced by `main()`.

#### 7.982.2.2 `main()`

```
int main (
 void)
```

This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.

#### Returns

Program return code.



### 7.983.2.2 main()

```
int main (
 void)
```

This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.

#### Returns

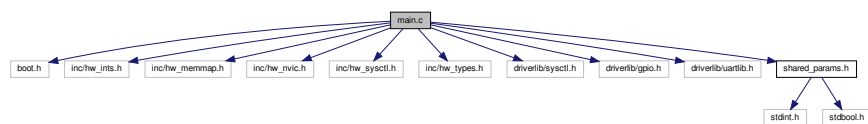
Program return code.

## 7.984 main.c File Reference

Bootloader application source file.

```
#include "boot.h"
#include "inc/hw_ints.h"
#include "inc/hw_memmap.h"
#include "inc/hw_nvic.h"
#include "inc/hw_sysctl.h"
#include "inc/hw_types.h"
#include "driverlib/sysctl.h"
#include "driverlib/gpio.h"
#include "driverlib/uartlib.h"
#include "shared_params.h"
```

Include dependency graph for ARMCM3\_LM3S\_EK\_LM3S6965\_IAR/Boot/main.c:



## Functions

- static void **Init** (void)  
*Initializes the microcontroller.*
- void **main** (void)  
*This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.*

### 7.984.1 Detailed Description

Bootloader application source file.

### 7.984.2 Function Documentation

## 7.984.2.1 Init()

```
static void Init (
 void) [static]
```

Initializes the microcontroller.

## Returns

none.

Referenced by main().

## 7.984.2.2 main()

```
void main (
 void)
```

This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.

## Returns

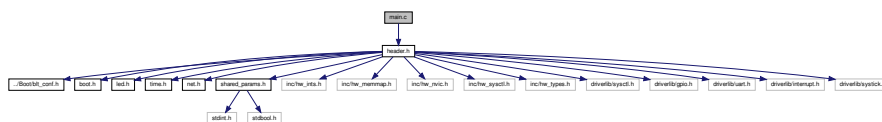
none.

## 7.985 main.c File Reference

Demo program application source file.

```
#include "header.h"
```

Include dependency graph for ARMC3M3\_LM3S\_EK\_LM3S6965\_IAR/Prog/main.c:



## Functions

- static void `Init` (void)  
*Initializes the microcontroller.*
- void `main` (void)  
*This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.*

### 7.985.1 Detailed Description

Demo program application source file.

### 7.985.2 Function Documentation

#### 7.985.2.1 Init()

```
static void Init (
 void) [static]
```

Initializes the microcontroller.

#### Returns

none.

Referenced by main().

#### 7.985.2.2 main()

```
void main (
 void)
```

This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.

#### Returns

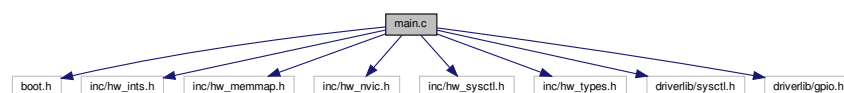
none.

## 7.986 main.c File Reference

Bootloader application source file.

```
#include "boot.h"
#include "inc/hw_ints.h"
#include "inc/hw_memmap.h"
#include "inc/hw_nvic.h"
#include "inc/hw_sysctl.h"
#include "inc/hw_types.h"
#include "driverlib/sysctl.h"
#include "driverlib/gpio.h"
```

Include dependency graph for ARMCM3\_LM3S\_EK\_LM3S8962\_GCC/Boot/main.c:



## Functions

- static void `Init` (void)  
*Initializes the microcontroller.*
- int `main` (void)  
*This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.*

### 7.986.1 Detailed Description

Bootloader application source file.

### 7.986.2 Function Documentation

#### 7.986.2.1 `Init()`

```
static void Init (
 void) [static]
```

Initializes the microcontroller.

#### Returns

none.

Referenced by `main()`.

#### 7.986.2.2 `main()`

```
int main (
 void)
```

This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.

#### Returns

Program return code.

## 7.987 main.c File Reference

Demo program application source file.

```
#include "header.h"
```

Include dependency graph for ARMC3M3\_LM3S\_EK\_LM3S8962\_GCC/Prog/main.c:



## Functions

- static void [Init](#) (void)  
*Initializes the microcontroller.*
- int [main](#) (void)  
*This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.*

### 7.987.1 Detailed Description

Demo program application source file.

### 7.987.2 Function Documentation

#### 7.987.2.1 Init()

```
static void Init (
 void) [static]
```

Initializes the microcontroller.

#### Returns

none.

Referenced by [main\(\)](#).



### 7.987.2.2 main()

```
int main (
 void)
```

This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.

#### Returns

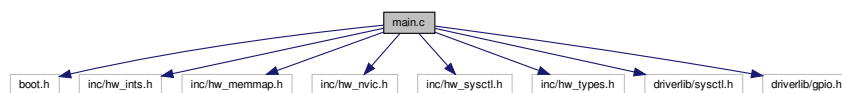
Program return code.

## 7.988 main.c File Reference

Bootloader application source file.

```
#include "boot.h"
#include "inc/hw_ints.h"
#include "inc/hw_memmap.h"
#include "inc/hw_nvic.h"
#include "inc/hw_sysctl.h"
#include "inc/hw_types.h"
#include "driverlib/sysctl.h"
#include "driverlib/gpio.h"
```

Include dependency graph for ARMCM3\_LM3S\_EK\_LM3S8962\_IAR/Boot/main.c:



## Functions

- static void [Init](#) (void)  
*Initializes the microcontroller.*
- void [main](#) (void)  
*This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.*

### 7.988.1 Detailed Description

Bootloader application source file.

### 7.988.2 Function Documentation

### 7.988.2.1 Init()

```
static void Init (
 void) [static]
```

Initializes the microcontroller.

#### Returns

none.

Referenced by main().

### 7.988.2.2 main()

```
void main (
 void)
```

This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.

#### Returns

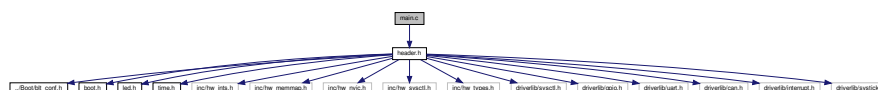
none.

## 7.989 main.c File Reference

Demo program application source file.

```
#include "header.h"
```

Include dependency graph for ARMCM3\_LM3S\_EK\_LM3S8962\_IAR/Prog/main.c:



## Functions

- static void `Init` (void)  
*Initializes the microcontroller.*
- void `main` (void)  
*This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.*

### 7.989.1 Detailed Description

Demo program application source file.

### 7.989.2 Function Documentation

#### 7.989.2.1 Init()

```
static void Init (
 void) [static]
```

Initializes the microcontroller.

##### Returns

none.

Referenced by main().

#### 7.989.2.2 main()

```
void main (
 void)
```

This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.

##### Returns

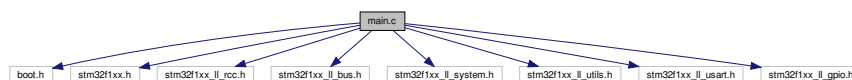
none.

## 7.990 main.c File Reference

Bootloader application source file.

```
#include "boot.h"
#include "stm32f1xx.h"
#include "stm32f1xx_ll_rcc.h"
#include "stm32f1xx_ll_bus.h"
#include "stm32f1xx_ll_system.h"
#include "stm32f1xx_ll_utils.h"
#include "stm32f1xx_ll_usart.h"
#include "stm32f1xx_ll_gpio.h"
```

Include dependency graph for ARMCM3\_STM32F1\_Nucleo\_F103RB\_GCC/Boot/main.c:



## Functions

- static void `Init` (void)  
*Initializes the microcontroller.*
- static void `SystemClock_Config` (void)  
*System Clock Configuration. This code was created by CubeMX and configures the system clock to match the configuration in the bootloader's configuration (blt\_conf.h), specifically the macros: BOOT\_CPU\_SYSTEM\_SPEED\_KHZ and BOOT\_CPU\_XTAL\_SPEED\_KHZ. Note that the Lower Layer drivers were selected in CubeMX for the RCC subsystem.*
- int `main` (void)  
*This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.*
- void `HAL_MspInit` (void)  
*Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).*
- void `HAL_MspDeInit` (void)  
*DeInitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level de-initialization (GPIOs, clock, DMA, interrupt).*

### 7.990.1 Detailed Description

Bootloader application source file.

### 7.990.2 Function Documentation

#### 7.990.2.1 HAL\_MspDeInit()

```
void HAL_MspDeInit (
 void)
```

DeInitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level de-initialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

#### 7.990.2.2 HAL\_MspInit()

```
void HAL_MspInit (
 void)
```

Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

### 7.990.2.3 Init()

```
static void Init (
 void) [static]
```

Initializes the microcontroller.

#### Returns

none.

Referenced by main().

### 7.990.2.4 main()

```
int main (
 void)
```

This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.

#### Returns

Program return code.

### 7.990.2.5 SystemClock\_Config()

```
static void SystemClock_Config (
 void) [static]
```

System Clock Configuration. This code was created by CubeMX and configures the system clock to match the configuration in the bootloader's configuration (blt\_conf.h), specifically the macros: BOOT\_CPU\_SYSTEM\_SPEED\_KHZ and BOOT\_CPU\_XTAL\_SPEED\_KHZ. Note that the Lower Layer drivers were selected in CubeMX for the RCC subsystem.

#### Returns

none.

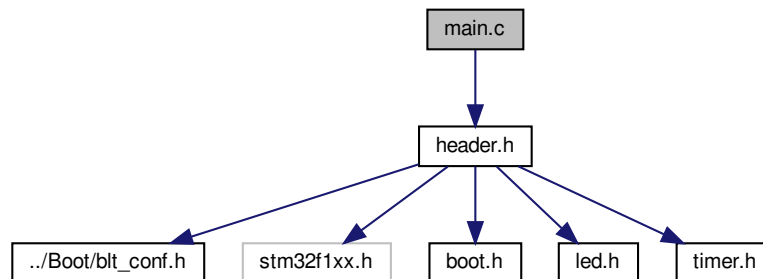
Referenced by Init().

## 7.991 main.c File Reference

Demo program application source file.

```
#include "header.h"
```

Include dependency graph for ARMC32F1\_Nucleo\_F103RB\_GCC/Prog/main.c:



### Functions

- static void `Init` (void)  
*Initializes the microcontroller.*
- static void `SystemClock_Config` (void)  
*System Clock Configuration. This code was created by CubeMX and configures the system clock.*
- static void `VectorBase_Config` (void)  
*Vector base address configuration. It should no longer be at the start of flash memory but moved forward because the first part of flash is reserved for the bootloader. Note that this is already done by the bootloader before starting this program. Unfortunately, function `SystemInit()` overwrites this change again.*
- int `main` (void)  
*This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.*
- void `HAL_MspInit` (void)  
*Initializes the Global MSP. This function is called from `HAL_Init()` function to perform system level initialization (GPIOs, clock, DMA, interrupt).*
- void `HAL_MspDeInit` (void)  
*Deinitializes the Global MSP. This function is called from `HAL_DeInit()` function to perform system level Deinitialization (GPIOs, clock, DMA, interrupt).*

### 7.991.1 Detailed Description

Demo program application source file.

### 7.991.2 Function Documentation

#### 7.991.2.1 HAL\_MspDeInit()

```
void HAL_MspDeInit (
 void)
```

Deinitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level Deinitialization (GPIOs, clock, DMA, interrupt).

##### Returns

none.

#### 7.991.2.2 HAL\_MspInit()

```
void HAL_MspInit (
 void)
```

Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).

##### Returns

none.

#### 7.991.2.3 Init()

```
static void Init (
 void) [static]
```

Initializes the microcontroller.

##### Returns

none.

Referenced by main().

#### 7.991.2.4 main()

```
int main (
 void)
```

This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.

##### Returns

Program return code.

### 7.991.2.5 SystemClock\_Config()

```
static void SystemClock_Config (
 void) [static]
```

System Clock Configuration. This code was created by CubeMX and configures the system clock.

#### Returns

none.

Referenced by Init().

### 7.991.2.6 VectorBase\_Config()

```
static void VectorBase_Config (
 void) [static]
```

Vector base address configuration. It should no longer be at the start of flash memory but moved forward because the first part of flash is reserved for the bootloader. Note that this is already done by the bootloader before starting this program. Unfortunately, function SystemInit() overwrites this change again.

#### Returns

none.

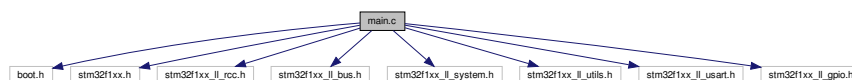
Referenced by Init().

## 7.992 main.c File Reference

Bootloader application source file.

```
#include "boot.h"
#include "stm32f1xx.h"
#include "stm32f1xx_ll_rcc.h"
#include "stm32f1xx_ll_bus.h"
#include "stm32f1xx_ll_system.h"
#include "stm32f1xx_ll_utils.h"
#include "stm32f1xx_ll_usart.h"
#include "stm32f1xx_ll_gpio.h"
```

Include dependency graph for ARMCM3\_STM32F1\_Nucleo\_F103RB\_IAR/Boot/main.c:





## Functions

- static void `Init` (void)  
*Initializes the microcontroller.*
- static void `SystemClock_Config` (void)  
*System Clock Configuration. This code was created by CubeMX and configures the system clock to match the configuration in the bootloader's configuration (blt\_conf.h), specifically the macros: BOOT\_CPU\_SYSTEM\_SPEED\_KHZ and BOOT\_CPU\_XTAL\_SPEED\_KHZ. Note that the Lower Layer drivers were selected in CubeMX for the RCC subsystem.*
- void `main` (void)  
*This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.*
- void `HAL_MspInit` (void)  
*Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).*
- void `HAL_MspDeInit` (void)  
*DeInitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level de-initialization (GPIOs, clock, DMA, interrupt).*

### 7.992.1 Detailed Description

Bootloader application source file.

### 7.992.2 Function Documentation

#### 7.992.2.1 HAL\_MspDeInit()

```
void HAL_MspDeInit (
 void)
```

DeInitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level de-initialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

#### 7.992.2.2 HAL\_MspInit()

```
void HAL_MspInit (
 void)
```

Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

### 7.992.2.3 Init()

```
static void Init (
 void) [static]
```

Initializes the microcontroller.

#### Returns

none.

Referenced by main().

### 7.992.2.4 main()

```
void main (
 void)
```

This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.

#### Returns

none.

### 7.992.2.5 SystemClock\_Config()

```
static void SystemClock_Config (
 void) [static]
```

System Clock Configuration. This code was created by CubeMX and configures the system clock to match the configuration in the bootloader's configuration (blt\_conf.h), specifically the macros: BOOT\_CPU\_SYSTEM\_SPEED\_KHZ and BOOT\_CPU\_XTAL\_SPEED\_KHZ. Note that the Lower Layer drivers were selected in CubeMX for the RCC subsystem.

#### Returns

none.

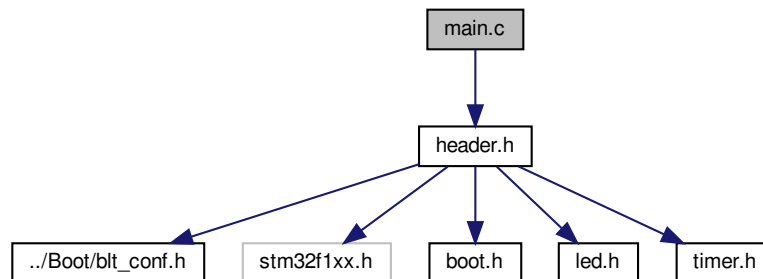
Referenced by Init().

## 7.993 main.c File Reference

Demo program application source file.

```
#include "header.h"
```

Include dependency graph for ARMCM3\_STM32F1\_Nucleo\_F103RB\_IAR/Prog/main.c:



### Functions

- static void [Init](#) (void)  
*Initializes the microcontroller.*
- static void [SystemClock\\_Config](#) (void)  
*System Clock Configuration. This code was created by CubeMX and configures the system clock.*
- static void [VectorBase\\_Config](#) (void)  
*Vector base address configuration. It should no longer be at the start of flash memory but moved forward because the first part of flash is reserved for the bootloader. Note that this is already done by the bootloader before starting this program. Unfortunately, function SystemInit() overwrites this change again.*
- void [main](#) (void)  
*This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.*
- void [HAL\\_MspInit](#) (void)  
*Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).*
- void [HAL\\_MspDeInit](#) (void)  
*Deinitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level Deinitialization (GPIOs, clock, DMA, interrupt).*

### 7.993.1 Detailed Description

Demo program application source file.

### 7.993.2 Function Documentation

#### 7.993.2.1 HAL\_MspDeInit()

```
void HAL_MspDeInit (
 void)
```

Deinitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level Deinitialization (GPIOs, clock, DMA, interrupt).

##### Returns

none.

#### 7.993.2.2 HAL\_MspInit()

```
void HAL_MspInit (
 void)
```

Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).

##### Returns

none.

#### 7.993.2.3 Init()

```
static void Init (
 void) [static]
```

Initializes the microcontroller.

##### Returns

none.

Referenced by main().

#### 7.993.2.4 main()

```
void main (
 void)
```

This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.

##### Returns

none.

## 7.993.2.5 SystemClock\_Config()

```
static void SystemClock_Config (
 void) [static]
```

System Clock Configuration. This code was created by CubeMX and configures the system clock.

## Returns

none.

Referenced by Init().

## 7.993.2.6 VectorBase\_Config()

```
static void VectorBase_Config (
 void) [static]
```

Vector base address configuration. It should no longer be at the start of flash memory but moved forward because the first part of flash is reserved for the bootloader. Note that this is already done by the bootloader before starting this program. Unfortunately, function SystemInit() overwrites this change again.

## Returns

none.

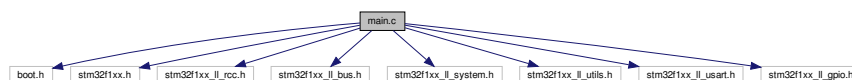
Referenced by Init().

## 7.994 main.c File Reference

Bootloader application source file.

```
#include "boot.h"
#include "stm32f1xx.h"
#include "stm32f1xx_ll_rcc.h"
#include "stm32f1xx_ll_bus.h"
#include "stm32f1xx_ll_system.h"
#include "stm32f1xx_ll_utils.h"
#include "stm32f1xx_ll_usart.h"
#include "stm32f1xx_ll_gpio.h"
```

Include dependency graph for ARMCM3\_STM32F1\_Nucleo\_F103RB\_Keil/Boot/main.c:



## Functions

- static void `Init` (void)  
*Initializes the microcontroller.*
- static void `SystemClock_Config` (void)  
*System Clock Configuration. This code was created by CubeMX and configures the system clock to match the configuration in the bootloader's configuration (blt\_conf.h), specifically the macros: BOOT\_CPU\_SYSTEM\_SPEED\_KHZ and BOOT\_CPU\_XTAL\_SPEED\_KHZ. Note that the Lower Layer drivers were selected in CubeMX for the RCC subsystem.*
- int `main` (void)  
*This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.*
- void `HAL_MspInit` (void)  
*Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).*
- void `HAL_MspDeInit` (void)  
*DeInitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level de-initialization (GPIOs, clock, DMA, interrupt).*

### 7.994.1 Detailed Description

Bootloader application source file.

### 7.994.2 Function Documentation

#### 7.994.2.1 HAL\_MspDeInit()

```
void HAL_MspDeInit (
 void)
```

DeInitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level de-initialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

#### 7.994.2.2 HAL\_MspInit()

```
void HAL_MspInit (
 void)
```

Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

### 7.994.2.3 Init()

```
static void Init (
 void) [static]
```

Initializes the microcontroller.

#### Returns

none.

Referenced by main().

### 7.994.2.4 main()

```
int main (
 void)
```

This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.

#### Returns

Program return code.

### 7.994.2.5 SystemClock\_Config()

```
static void SystemClock_Config (
 void) [static]
```

System Clock Configuration. This code was created by CubeMX and configures the system clock to match the configuration in the bootloader's configuration (blt\_conf.h), specifically the macros: BOOT\_CPU\_SYSTEM\_SPEED\_KHZ and BOOT\_CPU\_XTAL\_SPEED\_KHZ. Note that the Lower Layer drivers were selected in CubeMX for the RCC subsystem.

#### Returns

none.

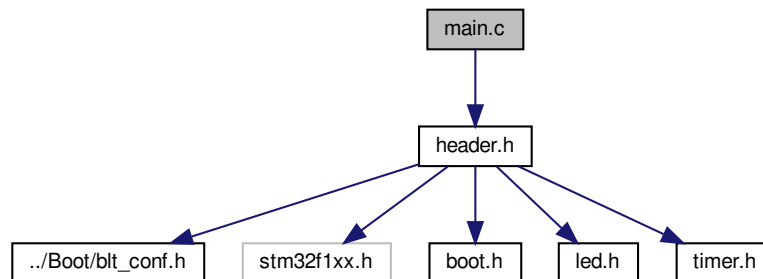
Referenced by Init().

## 7.995 main.c File Reference

Demo program application source file.

```
#include "header.h"
```

Include dependency graph for ARMCM3\_STM32F1\_Nucleo\_F103RB\_Keil/Prog/main.c:



## Functions

- static void `Init` (void)  
*Initializes the microcontroller.*
- static void `SystemClock_Config` (void)  
*System Clock Configuration. This code was created by CubeMX and configures the system clock.*
- static void `VectorBase_Config` (void)  
*Vector base address configuration. It should no longer be at the start of flash memory but moved forward because the first part of flash is reserved for the bootloader. Note that this is already done by the bootloader before starting this program. Unfortunately, function `SystemInit()` overwrites this change again.*
- int `main` (void)  
*This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.*
- void `HAL_MspInit` (void)  
*Initializes the Global MSP. This function is called from `HAL_Init()` function to perform system level initialization (GPIOs, clock, DMA, interrupt).*
- void `HAL_MspDeInit` (void)  
*Deinitializes the Global MSP. This function is called from `HAL_DeInit()` function to perform system level Deinitialization (GPIOs, clock, DMA, interrupt).*

### 7.995.1 Detailed Description

Demo program application source file.

### 7.995.2 Function Documentation



### 7.995.2.1 HAL\_MspDeInit()

```
void HAL_MspDeInit (
 void)
```

Deinitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level Deinitialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

### 7.995.2.2 HAL\_MspInit()

```
void HAL_MspInit (
 void)
```

Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

### 7.995.2.3 Init()

```
static void Init (
 void) [static]
```

Initializes the microcontroller.

#### Returns

none.

Referenced by main().

### 7.995.2.4 main()

```
int main (
 void)
```

This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.

#### Returns

Program return code.

### 7.995.2.5 SystemClock\_Config()

```
static void SystemClock_Config (
 void) [static]
```

System Clock Configuration. This code was created by CubeMX and configures the system clock.

#### Returns

none.

Referenced by Init().

### 7.995.2.6 VectorBase\_Config()

```
static void VectorBase_Config (
 void) [static]
```

Vector base address configuration. It should no longer be at the start of flash memory but moved forward because the first part of flash is reserved for the bootloader. Note that this is already done by the bootloader before starting this program. Unfortunately, function SystemInit() overwrites this change again.

#### Returns

none.

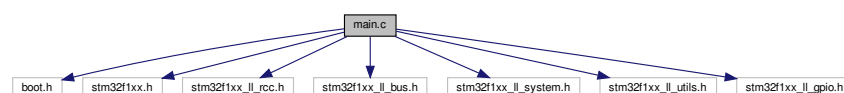
Referenced by Init().

## 7.996 main.c File Reference

Bootloader application source file.

```
#include "boot.h"
#include "stm32f1xx.h"
#include "stm32f1xx_ll_rcc.h"
#include "stm32f1xx_ll_bus.h"
#include "stm32f1xx_ll_system.h"
#include "stm32f1xx_ll_utils.h"
#include "stm32f1xx_ll_gpio.h"
```

Include dependency graph for ARMC32F1\_Olimex\_STM32H103\_GCC/Boot/main.c:



## Functions

- static void `Init` (void)  
*Initializes the microcontroller.*
- static void `SystemClock_Config` (void)  
*System Clock Configuration. This code was created by CubeMX and configures the system clock to match the configuration in the bootloader's configuration (blt\_conf.h), specifically the macros: BOOT\_CPU\_SYSTEM\_SPEED\_KHZ and BOOT\_CPU\_XTAL\_SPEED\_KHZ. Note that the Lower Layer drivers were selected in CubeMX for the RCC subsystem.*
- int `main` (void)  
*This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.*
- void `HAL_MspInit` (void)  
*Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).*
- void `HAL_MspDeInit` (void)  
*DeInitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level de-initialization (GPIOs, clock, DMA, interrupt).*

### 7.996.1 Detailed Description

Bootloader application source file.

### 7.996.2 Function Documentation

#### 7.996.2.1 HAL\_MspDeInit()

```
void HAL_MspDeInit (
 void)
```

DeInitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level de-initialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

#### 7.996.2.2 HAL\_MspInit()

```
void HAL_MspInit (
 void)
```

Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

### 7.996.2.3 Init()

```
static void Init (
 void) [static]
```

Initializes the microcontroller.

#### Returns

none.

Referenced by main().

### 7.996.2.4 main()

```
int main (
 void)
```

This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.

#### Returns

Program return code.

### 7.996.2.5 SystemClock\_Config()

```
static void SystemClock_Config (
 void) [static]
```

System Clock Configuration. This code was created by CubeMX and configures the system clock to match the configuration in the bootloader's configuration (blt\_conf.h), specifically the macros: BOOT\_CPU\_SYSTEM\_SPEED\_KHZ and BOOT\_CPU\_XTAL\_SPEED\_KHZ. Note that the Lower Layer drivers were selected in CubeMX for the RCC subsystem.

#### Returns

none.

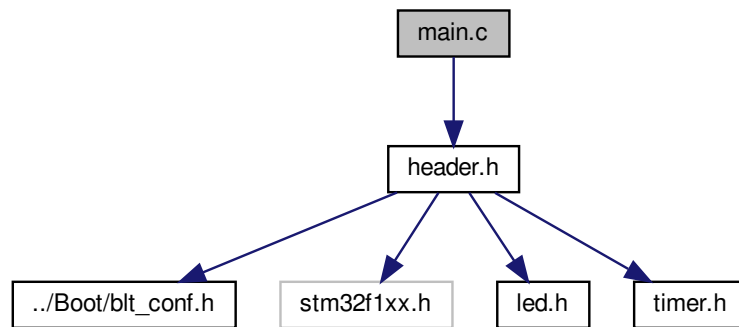
Referenced by Init().

## 7.997 main.c File Reference

Demo program application source file.

```
#include "header.h"
```

Include dependency graph for ARMCM3\_STM32F1\_Olimex\_STM32H103\_GCC/Prog/main.c:



### Functions

- static void `Init` (void)  
*Initializes the microcontroller.*
- static void `SystemClock_Config` (void)  
*System Clock Configuration. This code was created by CubeMX and configures the system clock.*
- static void `VectorBase_Config` (void)  
*Vector base address configuration. It should no longer be at the start of flash memory but moved forward because the first part of flash is reserved for the bootloader. Note that this is already done by the bootloader before starting this program. Unfortunately, function `SystemInit()` overwrites this change again.*
- int `main` (void)  
*This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.*
- void `HAL_MspInit` (void)  
*Initializes the Global MSP. This function is called from `HAL_Init()` function to perform system level initialization (GPIOs, clock, DMA, interrupt).*
- void `HAL_MspDeInit` (void)  
*Deinitializes the Global MSP. This function is called from `HAL_DeInit()` function to perform system level Deinitialization (GPIOs, clock, DMA, interrupt).*

### 7.997.1 Detailed Description

Demo program application source file.

### 7.997.2 Function Documentation

#### 7.997.2.1 HAL\_MspDeInit()

```
void HAL_MspDeInit (
 void)
```

Deinitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level Deinitialization (GPIOs, clock, DMA, interrupt).

##### Returns

none.

#### 7.997.2.2 HAL\_MspInit()

```
void HAL_MspInit (
 void)
```

Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).

##### Returns

none.

#### 7.997.2.3 Init()

```
static void Init (
 void) [static]
```

Initializes the microcontroller.

##### Returns

none.

Referenced by main().

#### 7.997.2.4 main()

```
int main (
 void)
```

This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.

##### Returns

Program return code.

### 7.997.2.5 SystemClock\_Config()

```
static void SystemClock_Config (
 void) [static]
```

System Clock Configuration. This code was created by CubeMX and configures the system clock.

#### Returns

none.

Referenced by Init().

### 7.997.2.6 VectorBase\_Config()

```
static void VectorBase_Config (
 void) [static]
```

Vector base address configuration. It should no longer be at the start of flash memory but moved forward because the first part of flash is reserved for the bootloader. Note that this is already done by the bootloader before starting this program. Unfortunately, function SystemInit() overwrites this change again.

#### Returns

none.

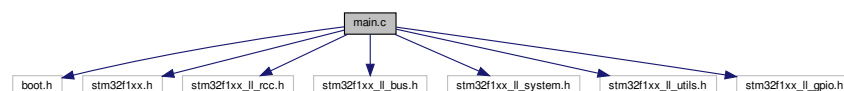
Referenced by Init().

## 7.998 main.c File Reference

Bootloader application source file.

```
#include "boot.h"
#include "stm32f1xx.h"
#include "stm32f1xx_ll_rcc.h"
#include "stm32f1xx_ll_bus.h"
#include "stm32f1xx_ll_system.h"
#include "stm32f1xx_ll_utils.h"
#include "stm32f1xx_ll_gpio.h"
```

Include dependency graph for ARMC32F1\_Olimex\_STM32H103\_IAR/Boot/main.c:



## Functions

- static void `Init` (void)  
*Initializes the microcontroller.*
- static void `SystemClock_Config` (void)  
*System Clock Configuration. This code was created by CubeMX and configures the system clock to match the configuration in the bootloader's configuration (blt\_conf.h), specifically the macros: BOOT\_CPU\_SYSTEM\_SPEED\_KHZ and BOOT\_CPU\_XTAL\_SPEED\_KHZ. Note that the Lower Layer drivers were selected in CubeMX for the RCC subsystem.*
- void `main` (void)  
*This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.*
- void `HAL_MspInit` (void)  
*Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).*
- void `HAL_MspDeInit` (void)  
*DeInitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level de-initialization (GPIOs, clock, DMA, interrupt).*

### 7.998.1 Detailed Description

Bootloader application source file.

### 7.998.2 Function Documentation

#### 7.998.2.1 HAL\_MspDeInit()

```
void HAL_MspDeInit (
 void)
```

DeInitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level de-initialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

#### 7.998.2.2 HAL\_MspInit()

```
void HAL_MspInit (
 void)
```

Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.



### 7.998.2.3 Init()

```
static void Init (
 void) [static]
```

Initializes the microcontroller.

#### Returns

none.

Referenced by main().

### 7.998.2.4 main()

```
void main (
 void)
```

This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.

#### Returns

none.

### 7.998.2.5 SystemClock\_Config()

```
static void SystemClock_Config (
 void) [static]
```

System Clock Configuration. This code was created by CubeMX and configures the system clock to match the configuration in the bootloader's configuration (blt\_conf.h), specifically the macros: BOOT\_CPU\_SYSTEM\_SPEED\_KHZ and BOOT\_CPU\_XTAL\_SPEED\_KHZ. Note that the Lower Layer drivers were selected in CubeMX for the RCC subsystem.

#### Returns

none.

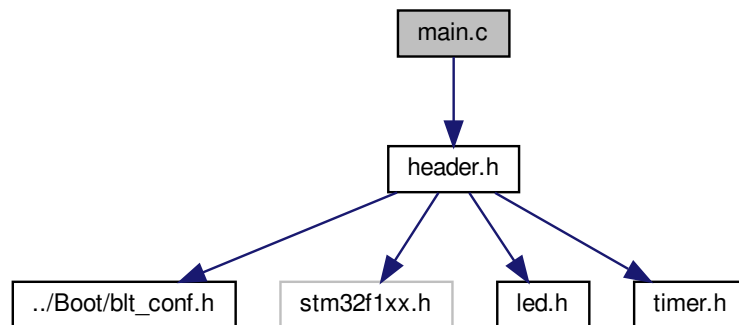
Referenced by Init().

## 7.999 main.c File Reference

Demo program application source file.

```
#include "header.h"
```

Include dependency graph for ARMCM3\_STM32F1\_Olimex\_STM32H103\_IAR/Prog/main.c:



### Functions

- static void [Init](#) (void)  
*Initializes the microcontroller.*
- static void [SystemClock\\_Config](#) (void)  
*System Clock Configuration. This code was created by CubeMX and configures the system clock.*
- static void [VectorBase\\_Config](#) (void)  
*Vector base address configuration. It should no longer be at the start of flash memory but moved forward because the first part of flash is reserved for the bootloader. Note that this is already done by the bootloader before starting this program. Unfortunately, function SystemInit() overwrites this change again.*
- void [main](#) (void)  
*This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.*
- void [HAL\\_MspInit](#) (void)  
*Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).*
- void [HAL\\_MspDeInit](#) (void)  
*Deinitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level Deinitialization (GPIOs, clock, DMA, interrupt).*

### 7.999.1 Detailed Description

Demo program application source file.

### 7.999.2 Function Documentation

### 7.999.2.1 HAL\_MspDeInit()

```
void HAL_MspDeInit (
 void)
```

Deinitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level Deinitialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

### 7.999.2.2 HAL\_MspInit()

```
void HAL_MspInit (
 void)
```

Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

### 7.999.2.3 Init()

```
static void Init (
 void) [static]
```

Initializes the microcontroller.

#### Returns

none.

Referenced by main().

### 7.999.2.4 main()

```
void main (
 void)
```

This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.

#### Returns

none.

### 7.999.2.5 SystemClock\_Config()

```
static void SystemClock_Config (
 void) [static]
```

System Clock Configuration. This code was created by CubeMX and configures the system clock.

#### Returns

none.

Referenced by Init().

### 7.999.2.6 VectorBase\_Config()

```
static void VectorBase_Config (
 void) [static]
```

Vector base address configuration. It should no longer be at the start of flash memory but moved forward because the first part of flash is reserved for the bootloader. Note that this is already done by the bootloader before starting this program. Unfortunately, function SystemInit() overwrites this change again.

#### Returns

none.

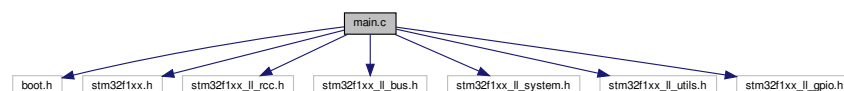
Referenced by Init().

## 7.1000 main.c File Reference

Bootloader application source file.

```
#include "boot.h"
#include "stm32f1xx.h"
#include "stm32f1xx_ll_rcc.h"
#include "stm32f1xx_ll_bus.h"
#include "stm32f1xx_ll_system.h"
#include "stm32f1xx_ll_utils.h"
#include "stm32f1xx_ll_gpio.h"
```

Include dependency graph for ARMC32F1\_Olimex\_STM32H103\_Keil/Boot/main.c:



## Functions

- static void `Init` (void)  
*Initializes the microcontroller.*
- static void `SystemClock_Config` (void)  
*System Clock Configuration. This code was created by CubeMX and configures the system clock to match the configuration in the bootloader's configuration (blt\_conf.h), specifically the macros: BOOT\_CPU\_SYSTEM\_SPEED\_KHZ and BOOT\_CPU\_XTAL\_SPEED\_KHZ. Note that the Lower Layer drivers were selected in CubeMX for the RCC subsystem.*
- int `main` (void)  
*This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.*
- void `HAL_MspInit` (void)  
*Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).*
- void `HAL_MspDeInit` (void)  
*DeInitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level de-initialization (GPIOs, clock, DMA, interrupt).*

### 7.1000.1 Detailed Description

Bootloader application source file.

### 7.1000.2 Function Documentation

#### 7.1000.2.1 HAL\_MspDeInit()

```
void HAL_MspDeInit (
 void)
```

DeInitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level de-initialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

#### 7.1000.2.2 HAL\_MspInit()

```
void HAL_MspInit (
 void)
```

Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

### 7.1000.2.3 Init()

```
static void Init (
 void) [static]
```

Initializes the microcontroller.

#### Returns

none.

Referenced by main().

### 7.1000.2.4 main()

```
int main (
 void)
```

This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.

#### Returns

Program return code.

### 7.1000.2.5 SystemClock\_Config()

```
static void SystemClock_Config (
 void) [static]
```

System Clock Configuration. This code was created by CubeMX and configures the system clock to match the configuration in the bootloader's configuration (blt\_conf.h), specifically the macros: BOOT\_CPU\_SYSTEM\_SPEED\_KHZ and BOOT\_CPU\_XTAL\_SPEED\_KHZ. Note that the Lower Layer drivers were selected in CubeMX for the RCC subsystem.

#### Returns

none.

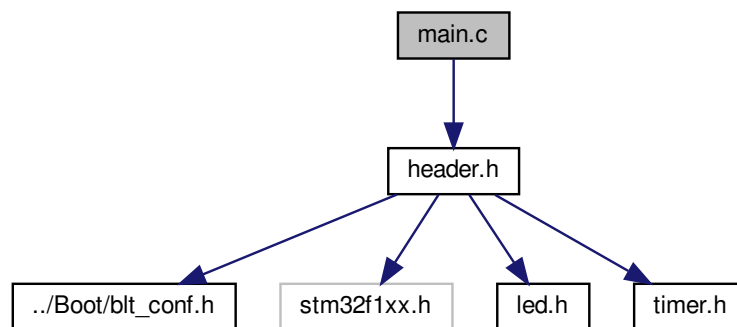
Referenced by Init().

## 7.1001 main.c File Reference

Demo program application source file.

```
#include "header.h"
```

Include dependency graph for ARMCM3\_STM32F1\_Olimex\_STM32H103\_Keil/Prog/main.c:



### Functions

- static void [Init](#) (void)  
*Initializes the microcontroller.*
- static void [SystemClock\\_Config](#) (void)  
*System Clock Configuration. This code was created by CubeMX and configures the system clock.*
- static void [VectorBase\\_Config](#) (void)  
*Vector base address configuration. It should no longer be at the start of flash memory but moved forward because the first part of flash is reserved for the bootloader. Note that this is already done by the bootloader before starting this program. Unfortunately, function SystemInit() overwrites this change again.*
- int [main](#) (void)  
*This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.*
- void [HAL\\_MspInit](#) (void)  
*Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).*
- void [HAL\\_MspDeInit](#) (void)  
*Deinitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level Deinitialization (GPIOs, clock, DMA, interrupt).*

### 7.1001.1 Detailed Description

Demo program application source file.

### 7.1001.2 Function Documentation

#### 7.1001.2.1 HAL\_MspDeInit()

```
void HAL_MspDeInit (
 void)
```

Deinitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level Deinitialization (GPIOs, clock, DMA, interrupt).

##### Returns

none.

#### 7.1001.2.2 HAL\_MspInit()

```
void HAL_MspInit (
 void)
```

Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).

##### Returns

none.

#### 7.1001.2.3 Init()

```
static void Init (
 void) [static]
```

Initializes the microcontroller.

##### Returns

none.

Referenced by main().

#### 7.1001.2.4 main()

```
int main (
 void)
```

This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.

##### Returns

Program return code.



## 7.1001.2.5 SystemClock\_Config()

```
static void SystemClock_Config (
 void) [static]
```

System Clock Configuration. This code was created by CubeMX and configures the system clock.

## Returns

none.

Referenced by Init().

## 7.1001.2.6 VectorBase\_Config()

```
static void VectorBase_Config (
 void) [static]
```

Vector base address configuration. It should no longer be at the start of flash memory but moved forward because the first part of flash is reserved for the bootloader. Note that this is already done by the bootloader before starting this program. Unfortunately, function SystemInit() overwrites this change again.

## Returns

none.

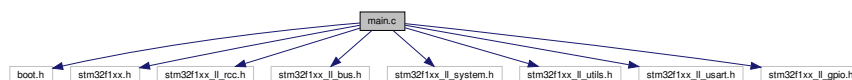
Referenced by Init().

## 7.1002 main.c File Reference

Bootloader application source file.

```
#include "boot.h"
#include "stm32f1xx.h"
#include "stm32f1xx_ll_rcc.h"
#include "stm32f1xx_ll_bus.h"
#include "stm32f1xx_ll_system.h"
#include "stm32f1xx_ll_utils.h"
#include "stm32f1xx_ll_usart.h"
#include "stm32f1xx_ll_gpio.h"
```

Include dependency graph for ARMC32F1\_Olimex\_STM32P103\_GCC/Boot/main.c:



## Functions

- static void `Init` (void)  
*Initializes the microcontroller.*
- static void `SystemClock_Config` (void)  
*System Clock Configuration. This code was created by CubeMX and configures the system clock to match the configuration in the bootloader's configuration (blt\_conf.h), specifically the macros: BOOT\_CPU\_SYSTEM\_SPEED\_KHZ and BOOT\_CPU\_XTAL\_SPEED\_KHZ. Note that the Lower Layer drivers were selected in CubeMX for the RCC subsystem.*
- int `main` (void)  
*This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.*
- void `HAL_MspInit` (void)  
*Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).*
- void `HAL_MspDeInit` (void)  
*DeInitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level de-initialization (GPIOs, clock, DMA, interrupt).*

### 7.1002.1 Detailed Description

Bootloader application source file.

### 7.1002.2 Function Documentation

#### 7.1002.2.1 HAL\_MspDeInit()

```
void HAL_MspDeInit (
 void)
```

DeInitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level de-initialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

#### 7.1002.2.2 HAL\_MspInit()

```
void HAL_MspInit (
 void)
```

Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

### 7.1002.2.3 Init()

```
static void Init (
 void) [static]
```

Initializes the microcontroller.

#### Returns

none.

Referenced by main().

### 7.1002.2.4 main()

```
int main (
 void)
```

This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.

#### Returns

Program return code.

### 7.1002.2.5 SystemClock\_Config()

```
static void SystemClock_Config (
 void) [static]
```

System Clock Configuration. This code was created by CubeMX and configures the system clock to match the configuration in the bootloader's configuration (blt\_conf.h), specifically the macros: BOOT\_CPU\_SYSTEM\_SPEED\_KHZ and BOOT\_CPU\_XTAL\_SPEED\_KHZ. Note that the Lower Layer drivers were selected in CubeMX for the RCC subsystem.

#### Returns

none.

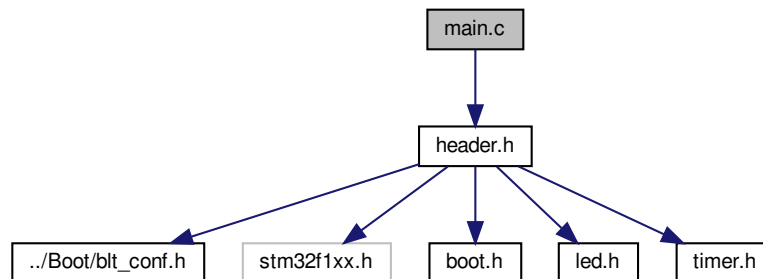
Referenced by Init().

## 7.1003 main.c File Reference

Demo program application source file.

```
#include "header.h"
```

Include dependency graph for ARMC32F1\_Olimex\_STM32P103\_GCC/Prog/main.c:



### Functions

- static void [Init](#) (void)  
*Initializes the microcontroller.*
- static void [SystemClock\\_Config](#) (void)  
*System Clock Configuration. This code was created by CubeMX and configures the system clock.*
- static void [VectorBase\\_Config](#) (void)  
*Vector base address configuration. It should no longer be at the start of flash memory but moved forward because the first part of flash is reserved for the bootloader. Note that this is already done by the bootloader before starting this program. Unfortunately, function SystemInit() overwrites this change again.*
- int [main](#) (void)  
*This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.*
- void [HAL\\_MspInit](#) (void)  
*Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).*
- void [HAL\\_MspDeInit](#) (void)  
*Deinitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level Deinitialization (GPIOs, clock, DMA, interrupt).*

### 7.1003.1 Detailed Description

Demo program application source file.

### 7.1003.2 Function Documentation

### 7.1003.2.1 HAL\_MspDeInit()

```
void HAL_MspDeInit (
 void)
```

Deinitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level Deinitialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

### 7.1003.2.2 HAL\_MspInit()

```
void HAL_MspInit (
 void)
```

Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

### 7.1003.2.3 Init()

```
static void Init (
 void) [static]
```

Initializes the microcontroller.

#### Returns

none.

Referenced by main().

### 7.1003.2.4 main()

```
int main (
 void)
```

This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.

#### Returns

Program return code.

### 7.1003.2.5 SystemClock\_Config()

```
static void SystemClock_Config (
 void) [static]
```

System Clock Configuration. This code was created by CubeMX and configures the system clock.

#### Returns

none.

Referenced by Init().

### 7.1003.2.6 VectorBase\_Config()

```
static void VectorBase_Config (
 void) [static]
```

Vector base address configuration. It should no longer be at the start of flash memory but moved forward because the first part of flash is reserved for the bootloader. Note that this is already done by the bootloader before starting this program. Unfortunately, function SystemInit() overwrites this change again.

#### Returns

none.

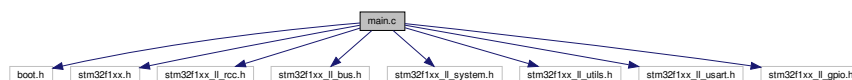
Referenced by Init().

## 7.1004 main.c File Reference

Bootloader application source file.

```
#include "boot.h"
#include "stm32f1xx.h"
#include "stm32f1xx_ll_rcc.h"
#include "stm32f1xx_ll_bus.h"
#include "stm32f1xx_ll_system.h"
#include "stm32f1xx_ll_utils.h"
#include "stm32f1xx_ll_usart.h"
#include "stm32f1xx_ll_gpio.h"
```

Include dependency graph for ARMCM3\_STM32F1\_Olimex\_STM32P103\_IAR/Boot/main.c:



## Functions

- static void `Init` (void)  
*Initializes the microcontroller.*
- static void `SystemClock_Config` (void)  
*System Clock Configuration. This code was created by CubeMX and configures the system clock to match the configuration in the bootloader's configuration (blt\_conf.h), specifically the macros: BOOT\_CPU\_SYSTEM\_SPEED\_KHZ and BOOT\_CPU\_XTAL\_SPEED\_KHZ. Note that the Lower Layer drivers were selected in CubeMX for the RCC subsystem.*
- void `main` (void)  
*This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.*
- void `HAL_MspInit` (void)  
*Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).*
- void `HAL_MspDeInit` (void)  
*DeInitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level de-initialization (GPIOs, clock, DMA, interrupt).*

### 7.1004.1 Detailed Description

Bootloader application source file.

### 7.1004.2 Function Documentation

#### 7.1004.2.1 HAL\_MspDeInit()

```
void HAL_MspDeInit (
 void)
```

DeInitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level de-initialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

#### 7.1004.2.2 HAL\_MspInit()

```
void HAL_MspInit (
 void)
```

Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

### 7.1004.2.3 Init()

```
static void Init (
 void) [static]
```

Initializes the microcontroller.

#### Returns

none.

Referenced by main().

### 7.1004.2.4 main()

```
void main (
 void)
```

This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.

#### Returns

none.

### 7.1004.2.5 SystemClock\_Config()

```
static void SystemClock_Config (
 void) [static]
```

System Clock Configuration. This code was created by CubeMX and configures the system clock to match the configuration in the bootloader's configuration (blt\_conf.h), specifically the macros: BOOT\_CPU\_SYSTEM\_SPEED\_KHZ and BOOT\_CPU\_XTAL\_SPEED\_KHZ. Note that the Lower Layer drivers were selected in CubeMX for the RCC subsystem.

#### Returns

none.

Referenced by Init().

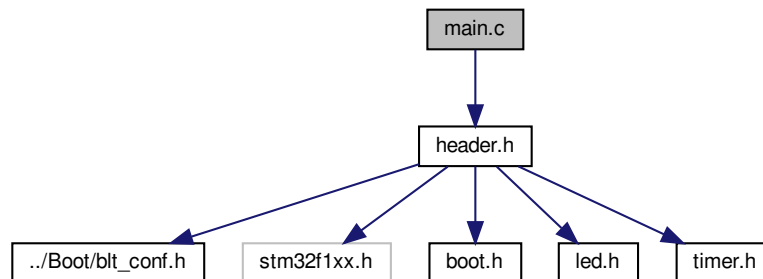


## 7.1005 main.c File Reference

Demo program application source file.

```
#include "header.h"
```

Include dependency graph for ARMCM3\_STM32F1\_Olimex\_STM32P103\_IAR/Prog/main.c:



### Functions

- static void [Init](#) (void)  
*Initializes the microcontroller.*
- static void [SystemClock\\_Config](#) (void)  
*System Clock Configuration. This code was created by CubeMX and configures the system clock.*
- static void [VectorBase\\_Config](#) (void)  
*Vector base address configuration. It should no longer be at the start of flash memory but moved forward because the first part of flash is reserved for the bootloader. Note that this is already done by the bootloader before starting this program. Unfortunately, function SystemInit() overwrites this change again.*
- void [main](#) (void)  
*This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.*
- void [HAL\\_MspInit](#) (void)  
*Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).*
- void [HAL\\_MspDeInit](#) (void)  
*Deinitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level Deinitialization (GPIOs, clock, DMA, interrupt).*

### 7.1005.1 Detailed Description

Demo program application source file.

### 7.1005.2 Function Documentation

#### 7.1005.2.1 HAL\_MspDeInit()

```
void HAL_MspDeInit (
 void)
```

Deinitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level Deinitialization (GPIOs, clock, DMA, interrupt).

##### Returns

none.

#### 7.1005.2.2 HAL\_MspInit()

```
void HAL_MspInit (
 void)
```

Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).

##### Returns

none.

#### 7.1005.2.3 Init()

```
static void Init (
 void) [static]
```

Initializes the microcontroller.

##### Returns

none.

Referenced by main().

#### 7.1005.2.4 main()

```
void main (
 void)
```

This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.

##### Returns

none.

## 7.1005.2.5 SystemClock\_Config()

```
static void SystemClock_Config (
 void) [static]
```

System Clock Configuration. This code was created by CubeMX and configures the system clock.

## Returns

none.

Referenced by Init().

## 7.1005.2.6 VectorBase\_Config()

```
static void VectorBase_Config (
 void) [static]
```

Vector base address configuration. It should no longer be at the start of flash memory but moved forward because the first part of flash is reserved for the bootloader. Note that this is already done by the bootloader before starting this program. Unfortunately, function SystemInit() overwrites this change again.

## Returns

none.

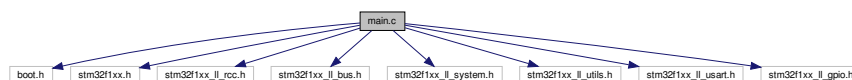
Referenced by Init().

## 7.1006 main.c File Reference

Bootloader application source file.

```
#include "boot.h"
#include "stm32f1xx.h"
#include "stm32f1xx_ll_rcc.h"
#include "stm32f1xx_ll_bus.h"
#include "stm32f1xx_ll_system.h"
#include "stm32f1xx_ll_utils.h"
#include "stm32f1xx_ll_usart.h"
#include "stm32f1xx_ll_gpio.h"
```

Include dependency graph for ARMC32F1\_Olimex\_STM32P103\_Keil/Boot/main.c:



## Functions

- static void `Init` (void)  
*Initializes the microcontroller.*
- static void `SystemClock_Config` (void)  
*System Clock Configuration. This code was created by CubeMX and configures the system clock to match the configuration in the bootloader's configuration (blt\_conf.h), specifically the macros: BOOT\_CPU\_SYSTEM\_SPEED\_KHZ and BOOT\_CPU\_XTAL\_SPEED\_KHZ. Note that the Lower Layer drivers were selected in CubeMX for the RCC subsystem.*
- int `main` (void)  
*This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.*
- void `HAL_MspInit` (void)  
*Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).*
- void `HAL_MspDeInit` (void)  
*DeInitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level de-initialization (GPIOs, clock, DMA, interrupt).*

### 7.1006.1 Detailed Description

Bootloader application source file.

### 7.1006.2 Function Documentation

#### 7.1006.2.1 HAL\_MspDeInit()

```
void HAL_MspDeInit (
 void)
```

DeInitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level de-initialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

#### 7.1006.2.2 HAL\_MspInit()

```
void HAL_MspInit (
 void)
```

Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

### 7.1006.2.3 Init()

```
static void Init (
 void) [static]
```

Initializes the microcontroller.

#### Returns

none.

Referenced by main().

### 7.1006.2.4 main()

```
int main (
 void)
```

This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.

#### Returns

Program exit code.

### 7.1006.2.5 SystemClock\_Config()

```
static void SystemClock_Config (
 void) [static]
```

System Clock Configuration. This code was created by CubeMX and configures the system clock to match the configuration in the bootloader's configuration (blt\_conf.h), specifically the macros: BOOT\_CPU\_SYSTEM\_SPEED\_KHZ and BOOT\_CPU\_XTAL\_SPEED\_KHZ. Note that the Lower Layer drivers were selected in CubeMX for the RCC subsystem.

#### Returns

none.

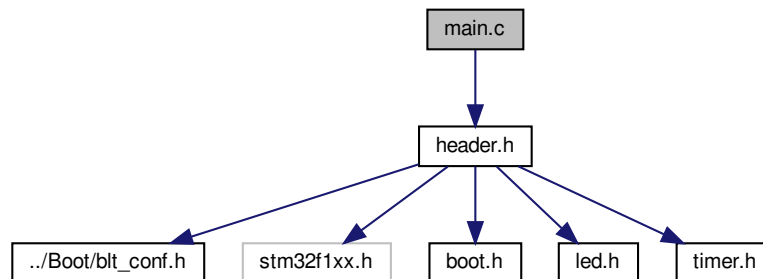
Referenced by Init().

## 7.1007 main.c File Reference

Demo program application source file.

```
#include "header.h"
```

Include dependency graph for ARMCM3\_STM32F1\_Olimex\_STM32P103\_Keil/Prog/main.c:



### Functions

- static void `Init` (void)  
*Initializes the microcontroller.*
- static void `SystemClock_Config` (void)  
*System Clock Configuration. This code was created by CubeMX and configures the system clock.*
- static void `VectorBase_Config` (void)  
*Vector base address configuration. It should no longer be at the start of flash memory but moved forward because the first part of flash is reserved for the bootloader. Note that this is already done by the bootloader before starting this program. Unfortunately, function `SystemInit()` overwrites this change again.*
- int `main` (void)  
*This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.*
- void `HAL_MspInit` (void)  
*Initializes the Global MSP. This function is called from `HAL_Init()` function to perform system level initialization (GPIOs, clock, DMA, interrupt).*
- void `HAL_MspDeInit` (void)  
*Deinitializes the Global MSP. This function is called from `HAL_DeInit()` function to perform system level Deinitialization (GPIOs, clock, DMA, interrupt).*

### 7.1007.1 Detailed Description

Demo program application source file.

### 7.1007.2 Function Documentation

### 7.1007.2.1 HAL\_MspDeInit()

```
void HAL_MspDeInit (
 void)
```

Deinitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level Deinitialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

### 7.1007.2.2 HAL\_MspInit()

```
void HAL_MspInit (
 void)
```

Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

### 7.1007.2.3 Init()

```
static void Init (
 void) [static]
```

Initializes the microcontroller.

#### Returns

none.

Referenced by main().

### 7.1007.2.4 main()

```
int main (
 void)
```

This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.

#### Returns

Program exit code.

### 7.1007.2.5 SystemClock\_Config()

```
static void SystemClock_Config (
 void) [static]
```

System Clock Configuration. This code was created by CubeMX and configures the system clock.

#### Returns

none.

Referenced by Init().

### 7.1007.2.6 VectorBase\_Config()

```
static void VectorBase_Config (
 void) [static]
```

Vector base address configuration. It should no longer be at the start of flash memory but moved forward because the first part of flash is reserved for the bootloader. Note that this is already done by the bootloader before starting this program. Unfortunately, function SystemInit() overwrites this change again.

#### Returns

none.

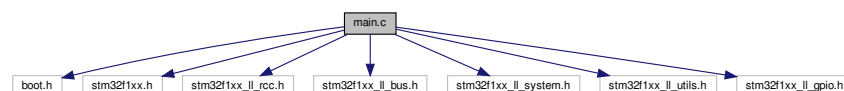
Referenced by Init().

## 7.1008 main.c File Reference

Bootloader application source file.

```
#include "boot.h"
#include "stm32f1xx.h"
#include "stm32f1xx_ll_rcc.h"
#include "stm32f1xx_ll_bus.h"
#include "stm32f1xx_ll_system.h"
#include "stm32f1xx_ll_utils.h"
#include "stm32f1xx_ll_gpio.h"
```

Include dependency graph for ARMC32F1\_Olimexino\_STM32\_GCC/Boot/main.c:





## Functions

- static void `Init` (void)  
*Initializes the microcontroller.*
- static void `SystemClock_Config` (void)  
*System Clock Configuration. This code was created by CubeMX and configures the system clock to match the configuration in the bootloader's configuration (blt\_conf.h), specifically the macros: BOOT\_CPU\_SYSTEM\_SPEED\_KHZ and BOOT\_CPU\_XTAL\_SPEED\_KHZ. Note that the Lower Layer drivers were selected in CubeMX for the RCC subsystem.*
- int `main` (void)  
*This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.*
- void `HAL_MspInit` (void)  
*Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).*
- void `HAL_MspDeInit` (void)  
*DeInitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level de-initialization (GPIOs, clock, DMA, interrupt).*

### 7.1008.1 Detailed Description

Bootloader application source file.

### 7.1008.2 Function Documentation

#### 7.1008.2.1 HAL\_MspDeInit()

```
void HAL_MspDeInit (
 void)
```

DeInitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level de-initialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

#### 7.1008.2.2 HAL\_MspInit()

```
void HAL_MspInit (
 void)
```

Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

### 7.1008.2.3 Init()

```
static void Init (
 void) [static]
```

Initializes the microcontroller.

#### Returns

none.

Referenced by main().

### 7.1008.2.4 main()

```
int main (
 void)
```

This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.

#### Returns

Program return code.

### 7.1008.2.5 SystemClock\_Config()

```
static void SystemClock_Config (
 void) [static]
```

System Clock Configuration. This code was created by CubeMX and configures the system clock to match the configuration in the bootloader's configuration (blt\_conf.h), specifically the macros: BOOT\_CPU\_SYSTEM\_SPEED\_KHZ and BOOT\_CPU\_XTAL\_SPEED\_KHZ. Note that the Lower Layer drivers were selected in CubeMX for the RCC subsystem.

#### Returns

none.

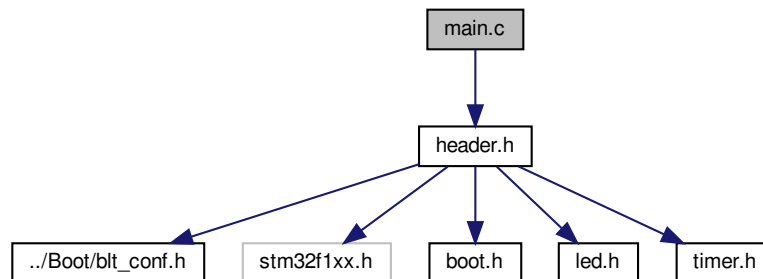
Referenced by Init().

## 7.1009 main.c File Reference

Demo program application source file.

```
#include "header.h"
```

Include dependency graph for ARMCM3\_STM32F1\_Olimexino\_STM32\_GCC/Prog/main.c:



### Functions

- static void [Init](#) (void)  
*Initializes the microcontroller.*
- static void [SystemClock\\_Config](#) (void)  
*System Clock Configuration. This code was created by CubeMX and configures the system clock.*
- static void [VectorBase\\_Config](#) (void)  
*Vector base address configuration. It should no longer be at the start of flash memory but moved forward because the first part of flash is reserved for the bootloader. Note that this is already done by the bootloader before starting this program. Unfortunately, function SystemInit() overwrites this change again.*
- int [main](#) (void)  
*This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.*
- void [HAL\\_MspInit](#) (void)  
*Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).*
- void [HAL\\_MspDeInit](#) (void)  
*Deinitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level Deinitialization (GPIOs, clock, DMA, interrupt).*

### 7.1009.1 Detailed Description

Demo program application source file.

### 7.1009.2 Function Documentation

### 7.1009.2.1 HAL\_MspDeInit()

```
void HAL_MspDeInit (
 void)
```

Deinitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level Deinitialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

### 7.1009.2.2 HAL\_MspInit()

```
void HAL_MspInit (
 void)
```

Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

### 7.1009.2.3 Init()

```
static void Init (
 void) [static]
```

Initializes the microcontroller.

#### Returns

none.

Referenced by main().

### 7.1009.2.4 main()

```
int main (
 void)
```

This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.

#### Returns

Program return code.

## 7.1009.2.5 SystemClock\_Config()

```
static void SystemClock_Config (
 void) [static]
```

System Clock Configuration. This code was created by CubeMX and configures the system clock.

## Returns

none.

Referenced by Init().

## 7.1009.2.6 VectorBase\_Config()

```
static void VectorBase_Config (
 void) [static]
```

Vector base address configuration. It should no longer be at the start of flash memory but moved forward because the first part of flash is reserved for the bootloader. Note that this is already done by the bootloader before starting this program. Unfortunately, function SystemInit() overwrites this change again.

## Returns

none.

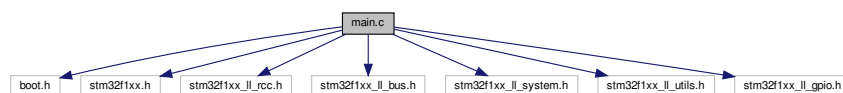
Referenced by Init().

## 7.1010 main.c File Reference

Bootloader application source file.

```
#include "boot.h"
#include "stm32f1xx.h"
#include "stm32f1xx_ll_rcc.h"
#include "stm32f1xx_ll_bus.h"
#include "stm32f1xx_ll_system.h"
#include "stm32f1xx_ll_utils.h"
#include "stm32f1xx_ll_gpio.h"
```

Include dependency graph for ARMC32F1\_Olimexino\_STM32\_IAR/Boot/main.c:



## Functions

- static void `Init` (void)  
*Initializes the microcontroller.*
- static void `SystemClock_Config` (void)  
*System Clock Configuration. This code was created by CubeMX and configures the system clock to match the configuration in the bootloader's configuration (blt\_conf.h), specifically the macros: BOOT\_CPU\_SYSTEM\_SPEED\_KHZ and BOOT\_CPU\_XTAL\_SPEED\_KHZ. Note that the Lower Layer drivers were selected in CubeMX for the RCC subsystem.*
- int `main` (void)  
*This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.*
- void `HAL_MspInit` (void)  
*Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).*
- void `HAL_MspDeInit` (void)  
*DeInitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level de-initialization (GPIOs, clock, DMA, interrupt).*

### 7.1010.1 Detailed Description

Bootloader application source file.

### 7.1010.2 Function Documentation

#### 7.1010.2.1 HAL\_MspDeInit()

```
void HAL_MspDeInit (
 void)
```

DeInitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level de-initialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

#### 7.1010.2.2 HAL\_MspInit()

```
void HAL_MspInit (
 void)
```

Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

### 7.1010.2.3 Init()

```
static void Init (
 void) [static]
```

Initializes the microcontroller.

#### Returns

none.

Referenced by main().

### 7.1010.2.4 main()

```
int main (
 void)
```

This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.

#### Returns

none.

### 7.1010.2.5 SystemClock\_Config()

```
static void SystemClock_Config (
 void) [static]
```

System Clock Configuration. This code was created by CubeMX and configures the system clock to match the configuration in the bootloader's configuration (blt\_conf.h), specifically the macros: BOOT\_CPU\_SYSTEM\_SPEED\_KHZ and BOOT\_CPU\_XTAL\_SPEED\_KHZ. Note that the Lower Layer drivers were selected in CubeMX for the RCC subsystem.

#### Returns

none.

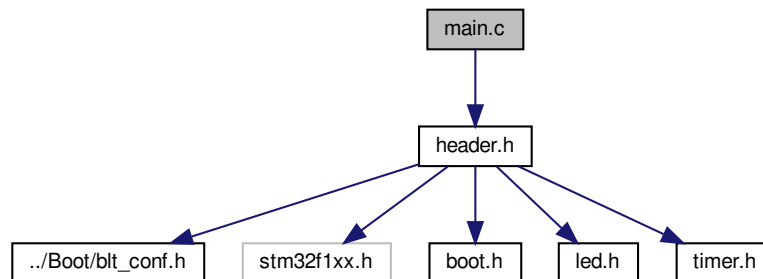
Referenced by Init().

## 7.1011 main.c File Reference

Demo program application source file.

```
#include "header.h"
```

Include dependency graph for ARMCM3\_STM32F1\_Olimexino\_STM32\_IAR/Prog/main.c:



### Functions

- static void `Init` (void)  
*Initializes the microcontroller.*
- static void `SystemClock_Config` (void)  
*System Clock Configuration. This code was created by CubeMX and configures the system clock.*
- static void `VectorBase_Config` (void)  
*Vector base address configuration. It should no longer be at the start of flash memory but moved forward because the first part of flash is reserved for the bootloader. Note that this is already done by the bootloader before starting this program. Unfortunately, function `SystemInit()` overwrites this change again.*
- void `main` (void)  
*This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.*
- void `HAL_MspInit` (void)  
*Initializes the Global MSP. This function is called from `HAL_Init()` function to perform system level initialization (GPIOs, clock, DMA, interrupt).*
- void `HAL_MspDeInit` (void)  
*Deinitializes the Global MSP. This function is called from `HAL_DeInit()` function to perform system level Deinitialization (GPIOs, clock, DMA, interrupt).*

### 7.1011.1 Detailed Description

Demo program application source file.

### 7.1011.2 Function Documentation



### 7.1011.2.1 HAL\_MspDeInit()

```
void HAL_MspDeInit (
 void)
```

Deinitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level Deinitialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

### 7.1011.2.2 HAL\_MspInit()

```
void HAL_MspInit (
 void)
```

Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

### 7.1011.2.3 Init()

```
static void Init (
 void) [static]
```

Initializes the microcontroller.

#### Returns

none.

Referenced by main().

### 7.1011.2.4 main()

```
void main (
 void)
```

This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.

#### Returns

none.

### 7.1011.2.5 SystemClock\_Config()

```
static void SystemClock_Config (
 void) [static]
```

System Clock Configuration. This code was created by CubeMX and configures the system clock.

#### Returns

none.

Referenced by Init().

### 7.1011.2.6 VectorBase\_Config()

```
static void VectorBase_Config (
 void) [static]
```

Vector base address configuration. It should no longer be at the start of flash memory but moved forward because the first part of flash is reserved for the bootloader. Note that this is already done by the bootloader before starting this program. Unfortunately, function SystemInit() overwrites this change again.

#### Returns

none.

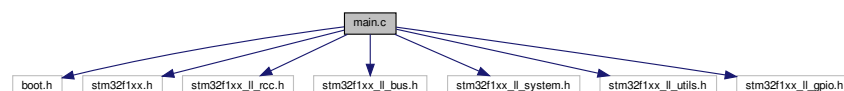
Referenced by Init().

## 7.1012 main.c File Reference

Bootloader application source file.

```
#include "boot.h"
#include "stm32f1xx.h"
#include "stm32f1xx_ll_rcc.h"
#include "stm32f1xx_ll_bus.h"
#include "stm32f1xx_ll_system.h"
#include "stm32f1xx_ll_utils.h"
#include "stm32f1xx_ll_gpio.h"
```

Include dependency graph for ARMC32F1\_Olimexino\_STM32\_Keil/Boot/main.c:



## Functions

- static void `Init` (void)  
*Initializes the microcontroller.*
- static void `SystemClock_Config` (void)  
*System Clock Configuration. This code was created by CubeMX and configures the system clock to match the configuration in the bootloader's configuration (blt\_conf.h), specifically the macros: BOOT\_CPU\_SYSTEM\_SPEED\_KHZ and BOOT\_CPU\_XTAL\_SPEED\_KHZ. Note that the Lower Layer drivers were selected in CubeMX for the RCC subsystem.*
- int `main` (void)  
*This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.*
- void `HAL_MspInit` (void)  
*Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).*
- void `HAL_MspDeInit` (void)  
*DeInitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level de-initialization (GPIOs, clock, DMA, interrupt).*

### 7.1012.1 Detailed Description

Bootloader application source file.

### 7.1012.2 Function Documentation

#### 7.1012.2.1 HAL\_MspDeInit()

```
void HAL_MspDeInit (
 void)
```

DeInitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level de-initialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

#### 7.1012.2.2 HAL\_MspInit()

```
void HAL_MspInit (
 void)
```

Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

### 7.1012.2.3 Init()

```
static void Init (
 void) [static]
```

Initializes the microcontroller.

#### Returns

none.

Referenced by main().

### 7.1012.2.4 main()

```
int main (
 void)
```

This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.

#### Returns

Program return code.

### 7.1012.2.5 SystemClock\_Config()

```
static void SystemClock_Config (
 void) [static]
```

System Clock Configuration. This code was created by CubeMX and configures the system clock to match the configuration in the bootloader's configuration (blt\_conf.h), specifically the macros: BOOT\_CPU\_SYSTEM\_SPEED\_KHZ and BOOT\_CPU\_XTAL\_SPEED\_KHZ. Note that the Lower Layer drivers were selected in CubeMX for the RCC subsystem.

#### Returns

none.

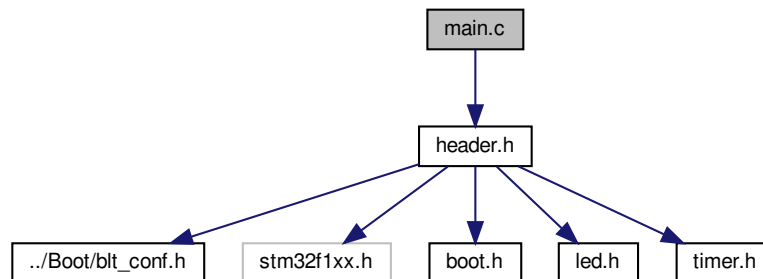
Referenced by Init().

## 7.1013 main.c File Reference

Demo program application source file.

```
#include "header.h"
```

Include dependency graph for ARMCM3\_STM32F1\_Olimexino\_STM32\_Keil/Prog/main.c:



### Functions

- static void [Init](#) (void)  
*Initializes the microcontroller.*
- static void [SystemClock\\_Config](#) (void)  
*System Clock Configuration. This code was created by CubeMX and configures the system clock.*
- static void [VectorBase\\_Config](#) (void)  
*Vector base address configuration. It should no longer be at the start of flash memory but moved forward because the first part of flash is reserved for the bootloader. Note that this is already done by the bootloader before starting this program. Unfortunately, function SystemInit() overwrites this change again.*
- int [main](#) (void)  
*This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.*
- void [HAL\\_MspInit](#) (void)  
*Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).*
- void [HAL\\_MspDeInit](#) (void)  
*Deinitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level Deinitialization (GPIOs, clock, DMA, interrupt).*

### 7.1013.1 Detailed Description

Demo program application source file.

### 7.1013.2 Function Documentation

#### 7.1013.2.1 HAL\_MspDeInit()

```
void HAL_MspDeInit (
 void)
```

Deinitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level Deinitialization (GPIOs, clock, DMA, interrupt).

##### Returns

none.

#### 7.1013.2.2 HAL\_MspInit()

```
void HAL_MspInit (
 void)
```

Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).

##### Returns

none.

#### 7.1013.2.3 Init()

```
static void Init (
 void) [static]
```

Initializes the microcontroller.

##### Returns

none.

Referenced by main().

#### 7.1013.2.4 main()

```
int main (
 void)
```

This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.

##### Returns

Program return code.

## 7.1013.2.5 SystemClock\_Config()

```
static void SystemClock_Config (
 void) [static]
```

System Clock Configuration. This code was created by CubeMX and configures the system clock.

## Returns

none.

Referenced by Init().

## 7.1013.2.6 VectorBase\_Config()

```
static void VectorBase_Config (
 void) [static]
```

Vector base address configuration. It should no longer be at the start of flash memory but moved forward because the first part of flash is reserved for the bootloader. Note that this is already done by the bootloader before starting this program. Unfortunately, function SystemInit() overwrites this change again.

## Returns

none.

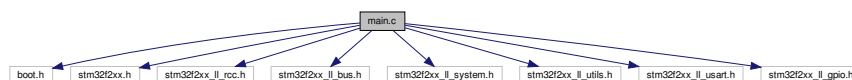
Referenced by Init().

## 7.1014 main.c File Reference

Bootloader application source file.

```
#include "boot.h"
#include "stm32f2xx.h"
#include "stm32f2xx_ll_rcc.h"
#include "stm32f2xx_ll_bus.h"
#include "stm32f2xx_ll_system.h"
#include "stm32f2xx_ll_utils.h"
#include "stm32f2xx_ll_usart.h"
#include "stm32f2xx_ll_gpio.h"
```

Include dependency graph for ARMC32F2\_Olimex\_STM32P207\_GCC/Boot/main.c:



## Functions

- static void `Init` (void)  
*Initializes the microcontroller.*
- static void `SystemClock_Config` (void)  
*System Clock Configuration. This code was created by CubeMX and configures the system clock to match the configuration in the bootloader's configuration (blt\_conf.h), specifically the macros: BOOT\_CPU\_SYSTEM\_SPEED\_KHZ and BOOT\_CPU\_XTAL\_SPEED\_KHZ. Note that the Lower Layer drivers were selected in CubeMX for the RCC subsystem.*
- int `main` (void)  
*This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.*
- void `HAL_MspInit` (void)  
*Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).*
- void `HAL_MspDeInit` (void)  
*DeInitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level de-initialization (GPIOs, clock, DMA, interrupt).*

### 7.1014.1 Detailed Description

Bootloader application source file.

### 7.1014.2 Function Documentation

#### 7.1014.2.1 HAL\_MspDeInit()

```
void HAL_MspDeInit (
 void)
```

DeInitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level de-initialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

#### 7.1014.2.2 HAL\_MspInit()

```
void HAL_MspInit (
 void)
```

Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.



### 7.1014.2.3 Init()

```
static void Init (
 void) [static]
```

Initializes the microcontroller.

#### Returns

none.

Referenced by main().

### 7.1014.2.4 main()

```
int main (
 void)
```

This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.

#### Returns

Program return code.

### 7.1014.2.5 SystemClock\_Config()

```
static void SystemClock_Config (
 void) [static]
```

System Clock Configuration. This code was created by CubeMX and configures the system clock to match the configuration in the bootloader's configuration (blt\_conf.h), specifically the macros: BOOT\_CPU\_SYSTEM\_SPEED\_KHZ and BOOT\_CPU\_XTAL\_SPEED\_KHZ. Note that the Lower Layer drivers were selected in CubeMX for the RCC subsystem.

#### Returns

none.

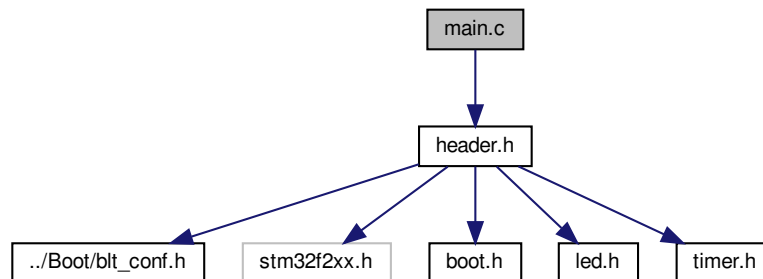
Referenced by Init().

## 7.1015 main.c File Reference

Demo program application source file.

```
#include "header.h"
```

Include dependency graph for ARMCM3\_STM32F2\_Olimex\_STM32P207\_GCC/Prog/main.c:



### Functions

- static void `Init` (void)  
*Initializes the microcontroller.*
- static void `SystemClock_Config` (void)  
*System Clock Configuration. This code was created by CubeMX and configures the system clock.*
- static void `VectorBase_Config` (void)  
*Vector base address configuration. It should no longer be at the start of flash memory but moved forward because the first part of flash is reserved for the bootloader. Note that this is already done by the bootloader before starting this program. Unfortunately, function `SystemInit()` overwrites this change again.*
- int `main` (void)  
*This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.*
- void `HAL_MspInit` (void)  
*Initializes the Global MSP. This function is called from `HAL_Init()` function to perform system level initialization (GPIOs, clock, DMA, interrupt).*
- void `HAL_MspDeInit` (void)  
*Deinitializes the Global MSP. This function is called from `HAL_DeInit()` function to perform system level Deinitialization (GPIOs, clock, DMA, interrupt).*

### 7.1015.1 Detailed Description

Demo program application source file.

### 7.1015.2 Function Documentation

### 7.1015.2.1 HAL\_MspDeInit()

```
void HAL_MspDeInit (
 void)
```

Deinitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level Deinitialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

### 7.1015.2.2 HAL\_MspInit()

```
void HAL_MspInit (
 void)
```

Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

### 7.1015.2.3 Init()

```
static void Init (
 void) [static]
```

Initializes the microcontroller.

#### Returns

none.

Referenced by main().

### 7.1015.2.4 main()

```
int main (
 void)
```

This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.

#### Returns

Program exit code.

### 7.1015.2.5 SystemClock\_Config()

```
static void SystemClock_Config (
 void) [static]
```

System Clock Configuration. This code was created by CubeMX and configures the system clock.

#### Returns

none.

Referenced by Init().

### 7.1015.2.6 VectorBase\_Config()

```
static void VectorBase_Config (
 void) [static]
```

Vector base address configuration. It should no longer be at the start of flash memory but moved forward because the first part of flash is reserved for the bootloader. Note that this is already done by the bootloader before starting this program. Unfortunately, function SystemInit() overwrites this change again.

#### Returns

none.

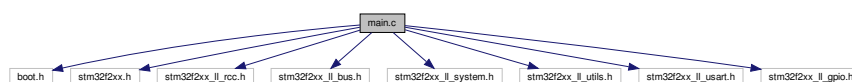
Referenced by Init().

## 7.1016 main.c File Reference

Bootloader application source file.

```
#include "boot.h"
#include "stm32f2xx.h"
#include "stm32f2xx_ll_rcc.h"
#include "stm32f2xx_ll_bus.h"
#include "stm32f2xx_ll_system.h"
#include "stm32f2xx_ll_utils.h"
#include "stm32f2xx_ll_usart.h"
#include "stm32f2xx_ll_gpio.h"
```

Include dependency graph for ARMC32F2\_Olimex\_STM32P207\_IAR/Boot/main.c:



## Functions

- static void `Init` (void)  
*Initializes the microcontroller.*
- static void `SystemClock_Config` (void)  
*System Clock Configuration. This code was created by CubeMX and configures the system clock to match the configuration in the bootloader's configuration (blt\_conf.h), specifically the macros: BOOT\_CPU\_SYSTEM\_SPEED\_KHZ and BOOT\_CPU\_XTAL\_SPEED\_KHZ. Note that the Lower Layer drivers were selected in CubeMX for the RCC subsystem.*
- int `main` (void)  
*This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.*
- void `HAL_MspInit` (void)  
*Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).*
- void `HAL_MspDeInit` (void)  
*DeInitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level de-initialization (GPIOs, clock, DMA, interrupt).*

### 7.1016.1 Detailed Description

Bootloader application source file.

### 7.1016.2 Function Documentation

#### 7.1016.2.1 HAL\_MspDeInit()

```
void HAL_MspDeInit (
 void)
```

DeInitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level de-initialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

#### 7.1016.2.2 HAL\_MspInit()

```
void HAL_MspInit (
 void)
```

Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

### 7.1016.2.3 Init()

```
static void Init (
 void) [static]
```

Initializes the microcontroller.

#### Returns

none.

Referenced by main().

### 7.1016.2.4 main()

```
int main (
 void)
```

This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.

#### Returns

none.

### 7.1016.2.5 SystemClock\_Config()

```
static void SystemClock_Config (
 void) [static]
```

System Clock Configuration. This code was created by CubeMX and configures the system clock to match the configuration in the bootloader's configuration (blt\_conf.h), specifically the macros: BOOT\_CPU\_SYSTEM\_SPEED\_KHZ and BOOT\_CPU\_XTAL\_SPEED\_KHZ. Note that the Lower Layer drivers were selected in CubeMX for the RCC subsystem.

#### Returns

none.

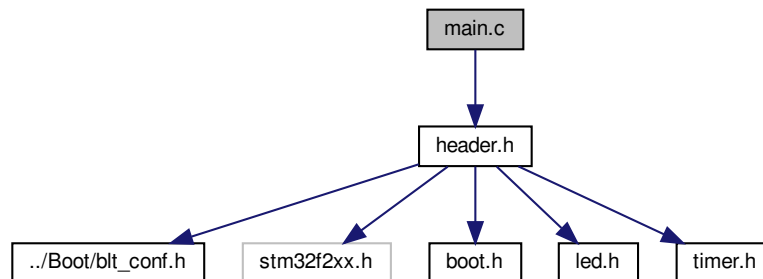
Referenced by Init().

## 7.1017 main.c File Reference

Demo program application source file.

```
#include "header.h"
```

Include dependency graph for ARMCM3\_STM32F2\_Olimex\_STM32P207\_IAR/Prog/main.c:



### Functions

- static void [Init](#) (void)  
*Initializes the microcontroller.*
- static void [SystemClock\\_Config](#) (void)  
*System Clock Configuration. This code was created by CubeMX and configures the system clock.*
- static void [VectorBase\\_Config](#) (void)  
*Vector base address configuration. It should no longer be at the start of flash memory but moved forward because the first part of flash is reserved for the bootloader. Note that this is already done by the bootloader before starting this program. Unfortunately, function SystemInit() overwrites this change again.*
- int [main](#) (void)  
*This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.*
- void [HAL\\_MspInit](#) (void)  
*Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).*
- void [HAL\\_MspDeInit](#) (void)  
*Deinitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level Deinitialization (GPIOs, clock, DMA, interrupt).*

### 7.1017.1 Detailed Description

Demo program application source file.

### 7.1017.2 Function Documentation

#### 7.1017.2.1 HAL\_MspDeInit()

```
void HAL_MspDeInit (
 void)
```

Deinitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level Deinitialization (GPIOs, clock, DMA, interrupt).

##### Returns

none.

#### 7.1017.2.2 HAL\_MspInit()

```
void HAL_MspInit (
 void)
```

Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).

##### Returns

none.

#### 7.1017.2.3 Init()

```
static void Init (
 void) [static]
```

Initializes the microcontroller.

##### Returns

none.

Referenced by main().

#### 7.1017.2.4 main()

```
int main (
 void)
```

This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.

##### Returns

none.



## 7.1017.2.5 SystemClock\_Config()

```
static void SystemClock_Config (
 void) [static]
```

System Clock Configuration. This code was created by CubeMX and configures the system clock.

## Returns

none.

Referenced by Init().

## 7.1017.2.6 VectorBase\_Config()

```
static void VectorBase_Config (
 void) [static]
```

Vector base address configuration. It should no longer be at the start of flash memory but moved forward because the first part of flash is reserved for the bootloader. Note that this is already done by the bootloader before starting this program. Unfortunately, function SystemInit() overwrites this change again.

## Returns

none.

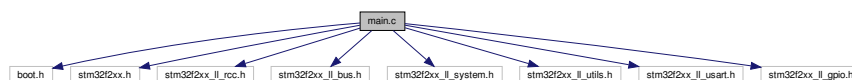
Referenced by Init().

## 7.1018 main.c File Reference

Bootloader application source file.

```
#include "boot.h"
#include "stm32f2xx.h"
#include "stm32f2xx_ll_rcc.h"
#include "stm32f2xx_ll_bus.h"
#include "stm32f2xx_ll_system.h"
#include "stm32f2xx_ll_utils.h"
#include "stm32f2xx_ll_usart.h"
#include "stm32f2xx_ll_gpio.h"
```

Include dependency graph for ARMCM3\_STM32F2\_Olimex\_STM32P207\_Keil/Boot/main.c:



## Functions

- static void `Init` (void)  
*Initializes the microcontroller.*
- static void `SystemClock_Config` (void)  
*System Clock Configuration. This code was created by CubeMX and configures the system clock to match the configuration in the bootloader's configuration (blt\_conf.h), specifically the macros: BOOT\_CPU\_SYSTEM\_SPEED\_KHZ and BOOT\_CPU\_XTAL\_SPEED\_KHZ. Note that the Lower Layer drivers were selected in CubeMX for the RCC subsystem.*
- int `main` (void)  
*This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.*
- void `HAL_MspInit` (void)  
*Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).*
- void `HAL_MspDeInit` (void)  
*DeInitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level de-initialization (GPIOs, clock, DMA, interrupt).*

### 7.1018.1 Detailed Description

Bootloader application source file.

### 7.1018.2 Function Documentation

#### 7.1018.2.1 HAL\_MspDeInit()

```
void HAL_MspDeInit (
 void)
```

DeInitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level de-initialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

#### 7.1018.2.2 HAL\_MspInit()

```
void HAL_MspInit (
 void)
```

Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

### 7.1018.2.3 Init()

```
static void Init (
 void) [static]
```

Initializes the microcontroller.

#### Returns

none.

Referenced by main().

### 7.1018.2.4 main()

```
int main (
 void)
```

This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.

#### Returns

Program return code.

### 7.1018.2.5 SystemClock\_Config()

```
static void SystemClock_Config (
 void) [static]
```

System Clock Configuration. This code was created by CubeMX and configures the system clock to match the configuration in the bootloader's configuration (blt\_conf.h), specifically the macros: BOOT\_CPU\_SYSTEM\_SPEED\_KHZ and BOOT\_CPU\_XTAL\_SPEED\_KHZ. Note that the Lower Layer drivers were selected in CubeMX for the RCC subsystem.

#### Returns

none.

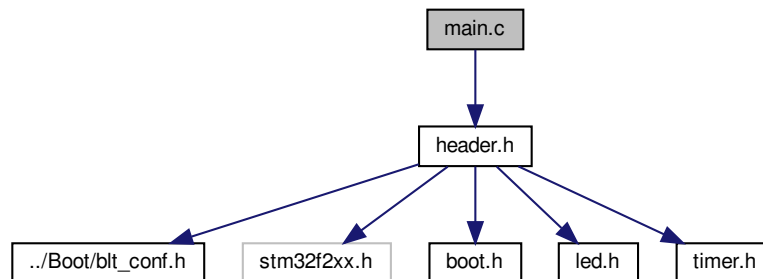
Referenced by Init().

## 7.1019 main.c File Reference

Demo program application source file.

```
#include "header.h"
```

Include dependency graph for ARMCM3\_STM32F2\_Olimex\_STM32P207\_Keil/Prog/main.c:



### Functions

- static void [Init](#) (void)  
*Initializes the microcontroller.*
- static void [SystemClock\\_Config](#) (void)  
*System Clock Configuration. This code was created by CubeMX and configures the system clock.*
- static void [VectorBase\\_Config](#) (void)  
*Vector base address configuration. It should no longer be at the start of flash memory but moved forward because the first part of flash is reserved for the bootloader. Note that this is already done by the bootloader before starting this program. Unfortunately, function SystemInit() overwrites this change again.*
- int [main](#) (void)  
*This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.*
- void [HAL\\_MspInit](#) (void)  
*Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).*
- void [HAL\\_MspDeInit](#) (void)  
*Deinitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level Deinitialization (GPIOs, clock, DMA, interrupt).*

### 7.1019.1 Detailed Description

Demo program application source file.

### 7.1019.2 Function Documentation

### 7.1019.2.1 HAL\_MspDeInit()

```
void HAL_MspDeInit (
 void)
```

Deinitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level Deinitialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

### 7.1019.2.2 HAL\_MspInit()

```
void HAL_MspInit (
 void)
```

Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

### 7.1019.2.3 Init()

```
static void Init (
 void) [static]
```

Initializes the microcontroller.

#### Returns

none.

Referenced by main().

### 7.1019.2.4 main()

```
int main (
 void)
```

This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.

#### Returns

Program exit code.

### 7.1019.2.5 SystemClock\_Config()

```
static void SystemClock_Config (
 void) [static]
```

System Clock Configuration. This code was created by CubeMX and configures the system clock.

#### Returns

none.

Referenced by Init().

### 7.1019.2.6 VectorBase\_Config()

```
static void VectorBase_Config (
 void) [static]
```

Vector base address configuration. It should no longer be at the start of flash memory but moved forward because the first part of flash is reserved for the bootloader. Note that this is already done by the bootloader before starting this program. Unfortunately, function SystemInit() overwrites this change again.

#### Returns

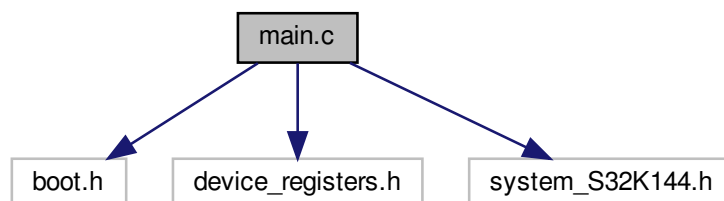
none.

Referenced by Init().

## 7.1020 main.c File Reference

Bootloader application source file.

```
#include "boot.h"
#include "device_registers.h"
#include "system_S32K144.h"
Include dependency graph for ARMCM4_S32K14_S32K144EVB_GCC/Boot/main.c:
```



## Functions

- static void `Init` (void)  
*Initializes the microcontroller.*
- static void `SystemClockConfig` (void)  
*System Clock Configuration. This code was derived from a S32 Design Studio example program. It uses the 8 MHz external crystal as a source for the PLL and configures the normal RUN mode for the following clock settings:*
- int `main` (void)  
*This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.*

### 7.1020.1 Detailed Description

Bootloader application source file.

### 7.1020.2 Function Documentation

#### 7.1020.2.1 Init()

```
static void Init (
 void) [static]
```

Initializes the microcontroller.

#### Returns

none.

Referenced by `main()`.

#### 7.1020.2.2 main()

```
int main (
 void)
```

This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.

#### Returns

Program return code.

### 7.1020.2.3 SystemClockConfig()

```
static void SystemClockConfig (
 void) [static]
```

System Clock Configuration. This code was derived from a S32 Design Studio example program. It uses the 8 MHz external crystal as a source for the PLL and configures the normal RUN mode for the following clock settings:

- SPLL\_CLK = 160 MHz
- CORE\_CLK = 80 MHz
- SYS\_CLK = 80 MHz
- BUS\_CLK = 40 MHz
- FLASH\_CLK = 26.67 MHz
- SIRCDIV1\_CLK = 8 MHz
- SIRCDIV2\_CLK = 8 MHz

#### Returns

none.

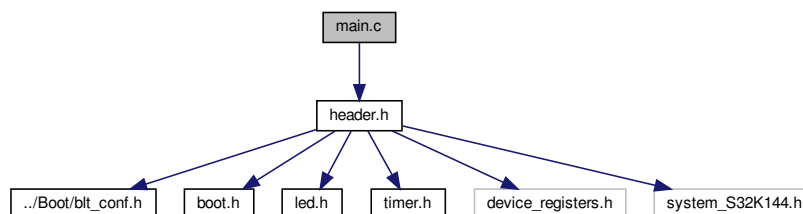
Referenced by Init().

## 7.1021 main.c File Reference

Demo program application source file.

```
#include "header.h"
```

Include dependency graph for ARMCM4\_S32K14\_S32K144EVB\_GCC/Prog/main.c:



## Functions

- static void [Init](#) (void)  
*Initializes the microcontroller.*
- static void [SystemClockConfig](#) (void)  
*System Clock Configuration. This code was derived from a S32 Design Studio example program. It uses the 8 MHz external crystal as a source for the PLL and configures the normal RUN mode for the following clock settings:*
- int [main](#) (void)  
*This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.*



### 7.1021.1 Detailed Description

Demo program application source file.

### 7.1021.2 Function Documentation

#### 7.1021.2.1 Init()

```
static void Init (
 void) [static]
```

Initializes the microcontroller.

##### Returns

none.

Referenced by main().

#### 7.1021.2.2 main()

```
int main (
 void)
```

This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.

##### Returns

Program return code.

#### 7.1021.2.3 SystemClockConfig()

```
static void SystemClockConfig (
 void) [static]
```

System Clock Configuration. This code was derived from a S32 Design Studio example program. It uses the 8 MHz external crystal as a source for the PLL and configures the normal RUN mode for the following clock settings:

- SPLL\_CLK = 160 MHz
- CORE\_CLK = 80 MHz
- SYS\_CLK = 80 MHz
- BUS\_CLK = 40 MHz
- FLASH\_CLK = 26.67 MHz
- SIRCDIV1\_CLK = 8 MHz
- SIRCDIV2\_CLK = 8 MHz

##### Returns

none.

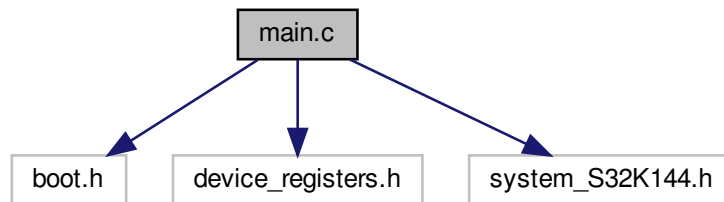
Referenced by Init().

## 7.1022 main.c File Reference

Bootloader application source file.

```
#include "boot.h"
#include "device_registers.h"
#include "system_S32K144.h"
```

Include dependency graph for ARMCM4\_S32K14\_S32K144EVB\_IAR/Boot/main.c:



### Functions

- static void [Init](#) (void)  
*Initializes the microcontroller.*
- static void [SystemClockConfig](#) (void)  
*System Clock Configuration. This code was derived from a S32 Design Studio example program. It uses the 8 MHz external crystal as a source for the PLL and configures the normal RUN mode for the following clock settings:*
- void [main](#) (void)  
*This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.*

### 7.1022.1 Detailed Description

Bootloader application source file.

### 7.1022.2 Function Documentation

#### 7.1022.2.1 Init()

```
static void Init (
 void) [static]
```

Initializes the microcontroller.

#### Returns

none.

Referenced by [main\(\)](#).

## 7.1022.2.2 main()

```
void main (
 void)
```

This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.

**Returns**

none.

## 7.1022.2.3 SystemClockConfig()

```
static void SystemClockConfig (
 void) [static]
```

System Clock Configuration. This code was derived from a S32 Design Studio example program. It uses the 8 MHz external crystal as a source for the PLL and configures the normal RUN mode for the following clock settings:

- SPLL\_CLK = 160 MHz
- CORE\_CLK = 80 MHz
- SYS\_CLK = 80 MHz
- BUS\_CLK = 40 MHz
- FLASH\_CLK = 26.67 MHz
- SIRCDIV1\_CLK = 8 MHz
- SIRCDIV2\_CLK = 8 MHz

**Returns**

none.

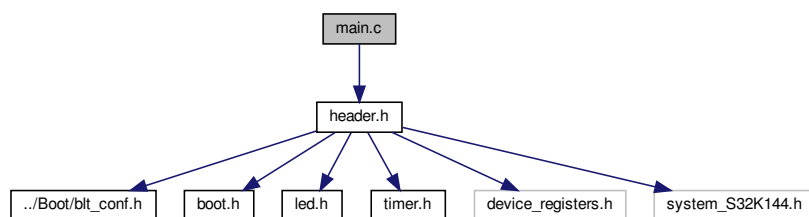
Referenced by Init().

## 7.1023 main.c File Reference

Demo program application source file.

```
#include "header.h"
```

Include dependency graph for ARMC4\_S32K14\_S32K144EVB\_IAR/Prog/main.c:



## Functions

- static void `Init` (void)  
*Initializes the microcontroller.*
- static void `SystemClockConfig` (void)  
*System Clock Configuration. This code was derived from a S32 Design Studio example program. It uses the 8 MHz external crystal as a source for the PLL and configures the normal RUN mode for the following clock settings:*
- void `main` (void)  
*This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.*

### 7.1023.1 Detailed Description

Demo program application source file.

### 7.1023.2 Function Documentation

#### 7.1023.2.1 Init()

```
static void Init (
 void) [static]
```

Initializes the microcontroller.

#### Returns

none.

Referenced by `main()`.

#### 7.1023.2.2 main()

```
void main (
 void)
```

This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.

#### Returns

none.

## 7.1023.2.3 SystemClockConfig()

```
static void SystemClockConfig (
 void) [static]
```

System Clock Configuration. This code was derived from a S32 Design Studio example program. It uses the 8 MHz external crystal as a source for the PLL and configures the normal RUN mode for the following clock settings:

- SPLL\_CLK = 160 MHz
- CORE\_CLK = 80 MHz
- SYS\_CLK = 80 MHz
- BUS\_CLK = 40 MHz
- FLASH\_CLK = 26.67 MHz
- SIRCDIV1\_CLK = 8 MHz
- SIRCDIV2\_CLK = 8 MHz

**Returns**

none.

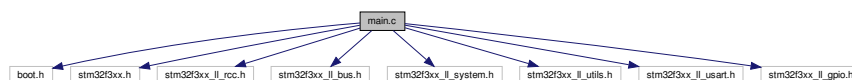
Referenced by Init().

## 7.1024 main.c File Reference

Bootloader application source file.

```
#include "boot.h"
#include "stm32f3xx.h"
#include "stm32f3xx_ll_rcc.h"
#include "stm32f3xx_ll_bus.h"
#include "stm32f3xx_ll_system.h"
#include "stm32f3xx_ll_utils.h"
#include "stm32f3xx_ll_usart.h"
#include "stm32f3xx_ll_gpio.h"
```

Include dependency graph for ARMCM4\_STM32F3\_Discovery\_F303VC\_GCC/Boot/main.c:



## Functions

- static void `Init` (void)  
*Initializes the microcontroller.*
- static void `SystemClock_Config` (void)  
*System Clock Configuration. This code was created by CubeMX and configures the system clock to match the configuration in the bootloader's configuration (blt\_conf.h), specifically the macros: BOOT\_CPU\_SYSTEM\_SPEED\_KHZ and BOOT\_CPU\_XTAL\_SPEED\_KHZ. Note that the Lower Layer drivers were selected in CubeMX for the RCC subsystem.*
- int `main` (void)  
*This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.*
- void `HAL_MspInit` (void)  
*Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).*
- void `HAL_MspDeInit` (void)  
*DeInitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level de-initialization (GPIOs, clock, DMA, interrupt).*

### 7.1024.1 Detailed Description

Bootloader application source file.

### 7.1024.2 Function Documentation

#### 7.1024.2.1 HAL\_MspDeInit()

```
void HAL_MspDeInit (
 void)
```

DeInitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level de-initialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

#### 7.1024.2.2 HAL\_MspInit()

```
void HAL_MspInit (
 void)
```

Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

### 7.1024.2.3 Init()

```
static void Init (
 void) [static]
```

Initializes the microcontroller.

#### Returns

none.

Referenced by main().

### 7.1024.2.4 main()

```
int main (
 void)
```

This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.

#### Returns

Program return code.

### 7.1024.2.5 SystemClock\_Config()

```
static void SystemClock_Config (
 void) [static]
```

System Clock Configuration. This code was created by CubeMX and configures the system clock to match the configuration in the bootloader's configuration (blt\_conf.h), specifically the macros: BOOT\_CPU\_SYSTEM\_SPEED\_KHZ and BOOT\_CPU\_XTAL\_SPEED\_KHZ. Note that the Lower Layer drivers were selected in CubeMX for the RCC subsystem.

#### Returns

none.

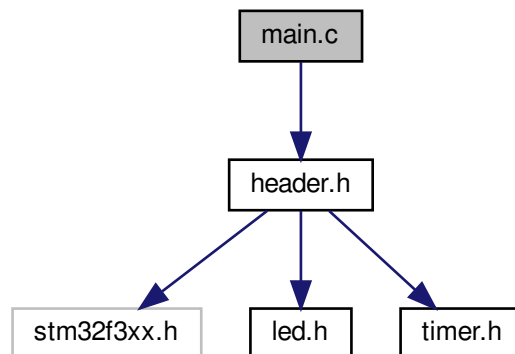
Referenced by Init().

## 7.1025 main.c File Reference

Demo program application source file.

```
#include "header.h"
```

Include dependency graph for ARMCM4\_STM32F3\_Discovery\_F303VC\_GCC/Prog/main.c:



### Functions

- static void `Init` (void)  
*Initializes the microcontroller.*
- static void `SystemClock_Config` (void)  
*System Clock Configuration. This code was created by CubeMX and configures the system clock.*
- static void `VectorBase_Config` (void)  
*Vector base address configuration. It should no longer be at the start of flash memory but moved forward because the first part of flash is reserved for the bootloader. Note that this is already done by the bootloader before starting this program. Unfortunately, function `SystemInit()` overwrites this change again.*
- int `main` (void)  
*This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.*
- void `HAL_MspInit` (void)  
*Initializes the Global MSP. This function is called from `HAL_Init()` function to perform system level initialization (GPIOs, clock, DMA, interrupt).*
- void `HAL_MspDeInit` (void)  
*Deinitializes the Global MSP. This function is called from `HAL_DeInit()` function to perform system level Deinitialization (GPIOs, clock, DMA, interrupt).*

### 7.1025.1 Detailed Description

Demo program application source file.



## 7.1025.2 Function Documentation

### 7.1025.2.1 HAL\_MspDeInit()

```
void HAL_MspDeInit (
 void)
```

Deinitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level Deinitialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

### 7.1025.2.2 HAL\_MspInit()

```
void HAL_MspInit (
 void)
```

Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

### 7.1025.2.3 Init()

```
static void Init (
 void) [static]
```

Initializes the microcontroller.

#### Returns

none.

Referenced by main().

#### 7.1025.2.4 main()

```
int main (
 void)
```

This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.

##### Returns

Program exit code.

#### 7.1025.2.5 SystemClock\_Config()

```
static void SystemClock_Config (
 void) [static]
```

System Clock Configuration. This code was created by CubeMX and configures the system clock.

##### Returns

none.

Referenced by Init().

#### 7.1025.2.6 VectorBase\_Config()

```
static void VectorBase_Config (
 void) [static]
```

Vector base address configuration. It should no longer be at the start of flash memory but moved forward because the first part of flash is reserved for the bootloader. Note that this is already done by the bootloader before starting this program. Unfortunately, function SystemInit() overwrites this change again.

##### Returns

none.

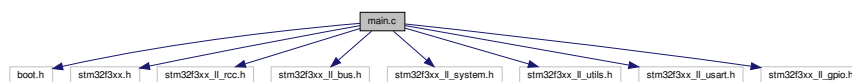
Referenced by Init().

## 7.1026 main.c File Reference

Bootloader application source file.

```
#include "boot.h"
#include "stm32f3xx.h"
#include "stm32f3xx_ll_rcc.h"
#include "stm32f3xx_ll_bus.h"
#include "stm32f3xx_ll_system.h"
#include "stm32f3xx_ll_utils.h"
#include "stm32f3xx_ll_usart.h"
#include "stm32f3xx_ll_gpio.h"
```

Include dependency graph for ARMCM4\_STM32F3\_Discovery\_F303VC\_IAR/Boot/main.c:



## Functions

- static void [Init](#) (void)  
*Initializes the microcontroller.*
- static void [SystemClock\\_Config](#) (void)  
*System Clock Configuration. This code was created by CubeMX and configures the system clock to match the configuration in the bootloader's configuration (blt\_conf.h), specifically the macros: BOOT\_CPU\_SYSTEM\_SPEED\_KHZ and BOOT\_CPU\_XTAL\_SPEED\_KHZ. Note that the Lower Layer drivers were selected in CubeMX for the RCC subsystem.*
- void [main](#) (void)  
*This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.*
- void [HAL\\_MspInit](#) (void)  
*Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).*
- void [HAL\\_MspDeInit](#) (void)  
*DeInitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level de-initialization (GPIOs, clock, DMA, interrupt).*

### 7.1026.1 Detailed Description

Bootloader application source file.

### 7.1026.2 Function Documentation

### 7.1026.2.1 HAL\_MspDeInit()

```
void HAL_MspDeInit (
 void)
```

DeInitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level de-initialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

### 7.1026.2.2 HAL\_MspInit()

```
void HAL_MspInit (
 void)
```

Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

### 7.1026.2.3 Init()

```
static void Init (
 void) [static]
```

Initializes the microcontroller.

#### Returns

none.

Referenced by main().

### 7.1026.2.4 main()

```
void main (
 void)
```

This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.

#### Returns

none.

### 7.1026.2.5 SystemClock\_Config()

```
static void SystemClock_Config (
 void) [static]
```

System Clock Configuration. This code was created by CubeMX and configures the system clock to match the configuration in the bootloader's configuration (blt\_conf.h), specifically the macros: BOOT\_CPU\_SYSTEM\_SPEED\_KHZ and BOOT\_CPU\_XTAL\_SPEED\_KHZ. Note that the Lower Layer drivers were selected in CubeMX for the RCC subsystem.

#### Returns

none.

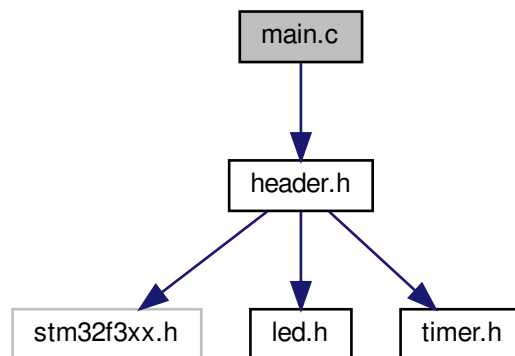
Referenced by Init().

## 7.1027 main.c File Reference

Demo program application source file.

```
#include "header.h"
```

Include dependency graph for ARMCM4\_STM32F3\_Discovery\_F303VC\_IAR/Prog/main.c:



## Functions

- static void [Init](#) (void)  
*Initializes the microcontroller.*
- static void [SystemClock\\_Config](#) (void)  
*System Clock Configuration. This code was created by CubeMX and configures the system clock.*
- static void [VectorBase\\_Config](#) (void)  
*Vector base address configuration. It should no longer be at the start of flash memory but moved forward because the first part of flash is reserved for the bootloader. Note that this is already done by the bootloader before starting this program. Unfortunately, function SystemInit() overwrites this change again.*

- void `main` (void)  
*This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.*
- void `HAL_MspInit` (void)  
*Initializes the Global MSP. This function is called from `HAL_Init()` function to perform system level initialization (GPIOs, clock, DMA, interrupt).*
- void `HAL_MspDeInit` (void)  
*Deinitializes the Global MSP. This function is called from `HAL_DeInit()` function to perform system level Deinitialization (GPIOs, clock, DMA, interrupt).*

### 7.1027.1 Detailed Description

Demo program application source file.

### 7.1027.2 Function Documentation

#### 7.1027.2.1 `HAL_MspDeInit()`

```
void HAL_MspDeInit (
 void)
```

Deinitializes the Global MSP. This function is called from `HAL_DeInit()` function to perform system level Deinitialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

#### 7.1027.2.2 `HAL_MspInit()`

```
void HAL_MspInit (
 void)
```

Initializes the Global MSP. This function is called from `HAL_Init()` function to perform system level initialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

### 7.1027.2.3 Init()

```
static void Init (
 void) [static]
```

Initializes the microcontroller.

#### Returns

none.

Referenced by main().

### 7.1027.2.4 main()

```
void main (
 void)
```

This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.

#### Returns

none.

### 7.1027.2.5 SystemClock\_Config()

```
static void SystemClock_Config (
 void) [static]
```

System Clock Configuration. This code was created by CubeMX and configures the system clock.

#### Returns

none.

Referenced by Init().

### 7.1027.2.6 VectorBase\_Config()

```
static void VectorBase_Config (
 void) [static]
```

Vector base address configuration. It should no longer be at the start of flash memory but moved forward because the first part of flash is reserved for the bootloader. Note that this is already done by the bootloader before starting this program. Unfortunately, function SystemInit() overwrites this change again.

#### Returns

none.

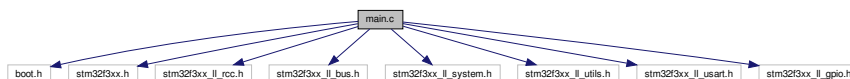
Referenced by Init().

## 7.1028 main.c File Reference

Bootloader application source file.

```
#include "boot.h"
#include "stm32f3xx.h"
#include "stm32f3xx_ll_rcc.h"
#include "stm32f3xx_ll_bus.h"
#include "stm32f3xx_ll_system.h"
#include "stm32f3xx_ll_utils.h"
#include "stm32f3xx_ll_usart.h"
#include "stm32f3xx_ll_gpio.h"
```

Include dependency graph for ARMC4\_STM32F3\_Discovery\_F303VC\_Keil/Boot/main.c:



## Functions

- static void [Init](#) (void)  
*Initializes the microcontroller.*
- static void [SystemClock\\_Config](#) (void)  
*System Clock Configuration. This code was created by CubeMX and configures the system clock to match the configuration in the bootloader's configuration (blt\_conf.h), specifically the macros: BOOT\_CPU\_SYSTEM\_SPEED\_KHZ and BOOT\_CPU\_XTAL\_SPEED\_KHZ. Note that the Lower Layer drivers were selected in CubeMX for the RCC subsystem.*
- int [main](#) (void)  
*This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.*
- void [HAL\\_MspInit](#) (void)  
*Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).*
- void [HAL\\_MspDeInit](#) (void)  
*DeInitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level de-initialization (GPIOs, clock, DMA, interrupt).*



## 7.1028.1 Detailed Description

Bootloader application source file.

## 7.1028.2 Function Documentation

### 7.1028.2.1 HAL\_MspDeInit()

```
void HAL_MspDeInit (
 void)
```

DeInitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level de-initialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

### 7.1028.2.2 HAL\_MspInit()

```
void HAL_MspInit (
 void)
```

Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

### 7.1028.2.3 Init()

```
static void Init (
 void) [static]
```

Initializes the microcontroller.

#### Returns

none.

Referenced by main().

#### 7.1028.2.4 main()

```
int main (
 void)
```

This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.

##### Returns

Program return code.

#### 7.1028.2.5 SystemClock\_Config()

```
static void SystemClock_Config (
 void) [static]
```

System Clock Configuration. This code was created by CubeMX and configures the system clock to match the configuration in the bootloader's configuration (blt\_conf.h), specifically the macros: BOOT\_CPU\_SYSTEM\_SPEED\_KHZ and BOOT\_CPU\_XTAL\_SPEED\_KHZ. Note that the Lower Layer drivers were selected in CubeMX for the RCC subsystem.

##### Returns

none.

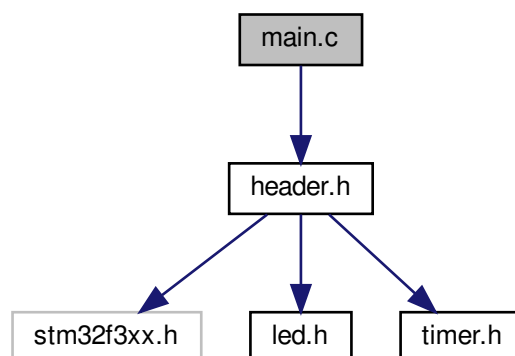
Referenced by Init().

### 7.1029 main.c File Reference

Demo program application source file.

```
#include "header.h"
```

Include dependency graph for ARMC4\_STM32F3\_Discovery\_F303VC\_Keil/Prog/main.c:



## Functions

- static void [Init](#) (void)  
*Initializes the microcontroller.*
- static void [SystemClock\\_Config](#) (void)  
*System Clock Configuration. This code was created by CubeMX and configures the system clock.*
- static void [VectorBase\\_Config](#) (void)  
*Vector base address configuration. It should no longer be at the start of flash memory but moved forward because the first part of flash is reserved for the bootloader. Note that this is already done by the bootloader before starting this program. Unfortunately, function SystemInit() overwrites this change again.*
- int [main](#) (void)  
*This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.*
- void [HAL\\_MspInit](#) (void)  
*Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).*
- void [HAL\\_MspDeInit](#) (void)  
*Deinitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level Deinitialization (GPIOs, clock, DMA, interrupt).*

### 7.1029.1 Detailed Description

Demo program application source file.

### 7.1029.2 Function Documentation

#### 7.1029.2.1 HAL\_MspDeInit()

```
void HAL_MspDeInit (
 void)
```

Deinitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level Deinitialization (GPIOs, clock, DMA, interrupt).

##### Returns

none.

#### 7.1029.2.2 HAL\_MspInit()

```
void HAL_MspInit (
 void)
```

Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).

##### Returns

none.

### 7.1029.2.3 Init()

```
static void Init (
 void) [static]
```

Initializes the microcontroller.

#### Returns

none.

Referenced by main().

### 7.1029.2.4 main()

```
int main (
 void)
```

This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.

#### Returns

Program exit code.

### 7.1029.2.5 SystemClock\_Config()

```
static void SystemClock_Config (
 void) [static]
```

System Clock Configuration. This code was created by CubeMX and configures the system clock.

#### Returns

none.

Referenced by Init().

## 7.1029.2.6 VectorBase\_Config()

```
static void VectorBase_Config (
 void) [static]
```

Vector base address configuration. It should no longer be at the start of flash memory but moved forward because the first part of flash is reserved for the bootloader. Note that this is already done by the bootloader before starting this program. Unfortunately, function SystemInit() overwrites this change again.

## Returns

none.

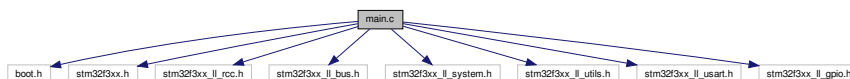
Referenced by Init().

## 7.1030 main.c File Reference

Bootloader application source file.

```
#include "boot.h"
#include "stm32f3xx.h"
#include "stm32f3xx_ll_rcc.h"
#include "stm32f3xx_ll_bus.h"
#include "stm32f3xx_ll_system.h"
#include "stm32f3xx_ll_utils.h"
#include "stm32f3xx_ll_usart.h"
#include "stm32f3xx_ll_gpio.h"
```

Include dependency graph for ARMC4\_STM32F3\_Nucleo\_F303K8\_GCC/Boot/main.c:



## Functions

- static void [Init](#) (void)  
*Initializes the microcontroller.*
- static void [SystemClock\\_Config](#) (void)  
*System Clock Configuration. This code was created by CubeMX and configures the system clock to match the configuration in the bootloader's configuration (blt\_conf.h), specifically the macros: BOOT\_CPU\_SYSTEM\_SPEED\_KHZ and BOOT\_CPU\_XTAL\_SPEED\_KHZ. Note that the Lower Layer drivers were selected in CubeMX for the RCC subsystem.*
- int [main](#) (void)  
*This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.*
- void [HAL\\_MspInit](#) (void)  
*Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).*
- void [HAL\\_MspDeInit](#) (void)  
*DeInitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level de-initialization (GPIOs, clock, DMA, interrupt).*

### 7.1030.1 Detailed Description

Bootloader application source file.

### 7.1030.2 Function Documentation

#### 7.1030.2.1 HAL\_MspDeInit()

```
void HAL_MspDeInit (
 void)
```

DeInitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level de-initialization (GPIOs, clock, DMA, interrupt).

##### Returns

none.

#### 7.1030.2.2 HAL\_MspInit()

```
void HAL_MspInit (
 void)
```

Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).

##### Returns

none.

#### 7.1030.2.3 Init()

```
static void Init (
 void) [static]
```

Initializes the microcontroller.

##### Returns

none.

Referenced by main().

#### 7.1030.2.4 main()

```
int main (
 void)
```

This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.

##### Returns

Program return code.

#### 7.1030.2.5 SystemClock\_Config()

```
static void SystemClock_Config (
 void) [static]
```

System Clock Configuration. This code was created by CubeMX and configures the system clock to match the configuration in the bootloader's configuration (blt\_conf.h), specifically the macros: BOOT\_CPU\_SYSTEM\_SPEED\_KHZ and BOOT\_CPU\_XTAL\_SPEED\_KHZ. Note that the Lower Layer drivers were selected in CubeMX for the RCC subsystem.

##### Returns

none.

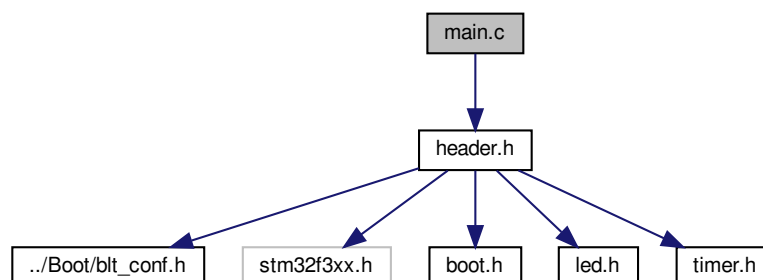
Referenced by Init().

## 7.1031 main.c File Reference

Demo program application source file.

```
#include "header.h"
```

Include dependency graph for ARMCM4\_STM32F3\_Nucleo\_F303K8\_GCC/Prog/main.c:



## Functions

- static void [Init](#) (void)  
*Initializes the microcontroller.*
- static void [SystemClock\\_Config](#) (void)  
*System Clock Configuration. This code was created by CubeMX and configures the system clock.*
- static void [VectorBase\\_Config](#) (void)  
*Vector base address configuration. It should no longer be at the start of flash memory but moved forward because the first part of flash is reserved for the bootloader. Note that this is already done by the bootloader before starting this program. Unfortunately, function SystemInit() overwrites this change again.*
- int [main](#) (void)  
*This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.*
- void [HAL\\_MspInit](#) (void)  
*Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).*
- void [HAL\\_MspDeInit](#) (void)  
*Deinitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level Deinitialization (GPIOs, clock, DMA, interrupt).*

### 7.1031.1 Detailed Description

Demo program application source file.

### 7.1031.2 Function Documentation

#### 7.1031.2.1 HAL\_MspDeInit()

```
void HAL_MspDeInit (
 void)
```

Deinitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level Deinitialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

#### 7.1031.2.2 HAL\_MspInit()

```
void HAL_MspInit (
 void)
```

Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.



### 7.1031.2.3 Init()

```
static void Init (
 void) [static]
```

Initializes the microcontroller.

#### Returns

none.

Referenced by main().

### 7.1031.2.4 main()

```
int main (
 void)
```

This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.

#### Returns

Program exit code.

### 7.1031.2.5 SystemClock\_Config()

```
static void SystemClock_Config (
 void) [static]
```

System Clock Configuration. This code was created by CubeMX and configures the system clock.

#### Returns

none.

Referenced by Init().

### 7.1031.2.6 VectorBase\_Config()

```
static void VectorBase_Config (
 void) [static]
```

Vector base address configuration. It should no longer be at the start of flash memory but moved forward because the first part of flash is reserved for the bootloader. Note that this is already done by the bootloader before starting this program. Unfortunately, function SystemInit() overwrites this change again.

#### Returns

none.

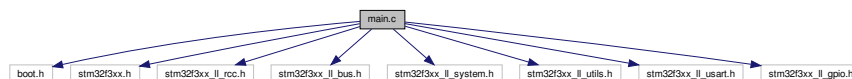
Referenced by Init().

## 7.1032 main.c File Reference

Bootloader application source file.

```
#include "boot.h"
#include "stm32f3xx.h"
#include "stm32f3xx_ll_rcc.h"
#include "stm32f3xx_ll_bus.h"
#include "stm32f3xx_ll_system.h"
#include "stm32f3xx_ll_utils.h"
#include "stm32f3xx_ll_usart.h"
#include "stm32f3xx_ll_gpio.h"
```

Include dependency graph for ARMC4\_STM32F3\_Nucleo\_F303K8\_IAR/Boot/main.c:



## Functions

- static void [Init](#) (void)  
*Initializes the microcontroller.*
- static void [SystemClock\\_Config](#) (void)  
*System Clock Configuration. This code was created by CubeMX and configures the system clock to match the configuration in the bootloader's configuration (blt\_conf.h), specifically the macros: BOOT\_CPU\_SYSTEM\_SPEED\_KHZ and BOOT\_CPU\_XTAL\_SPEED\_KHZ. Note that the Lower Layer drivers were selected in CubeMX for the RCC subsystem.*
- void [main](#) (void)  
*This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.*
- void [HAL\\_MspInit](#) (void)  
*Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).*
- void [HAL\\_MspDeInit](#) (void)  
*DeInitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level de-initialization (GPIOs, clock, DMA, interrupt).*

## 7.1032.1 Detailed Description

Bootloader application source file.

## 7.1032.2 Function Documentation

### 7.1032.2.1 HAL\_MspDeInit()

```
void HAL_MspDeInit (
 void)
```

DeInitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level de-initialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

### 7.1032.2.2 HAL\_MspInit()

```
void HAL_MspInit (
 void)
```

Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

### 7.1032.2.3 Init()

```
static void Init (
 void) [static]
```

Initializes the microcontroller.

#### Returns

none.

Referenced by main().

#### 7.1032.2.4 main()

```
void main (
 void)
```

This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.

##### Returns

None.

#### 7.1032.2.5 SystemClock\_Config()

```
static void SystemClock_Config (
 void) [static]
```

System Clock Configuration. This code was created by CubeMX and configures the system clock to match the configuration in the bootloader's configuration (blt\_conf.h), specifically the macros: BOOT\_CPU\_SYSTEM\_SPEED\_KHZ and BOOT\_CPU\_XTAL\_SPEED\_KHZ. Note that the Lower Layer drivers were selected in CubeMX for the RCC subsystem.

##### Returns

none.

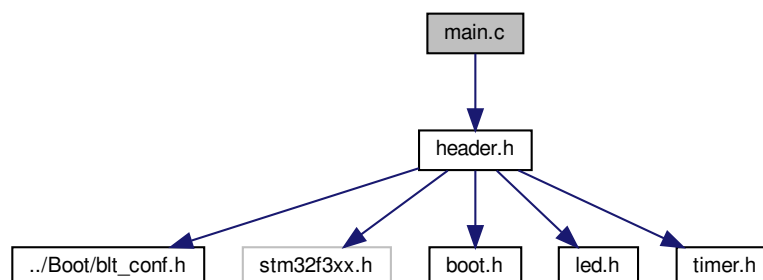
Referenced by Init().

## 7.1033 main.c File Reference

Demo program application source file.

```
#include "header.h"
```

Include dependency graph for ARMCM4\_STM32F3\_Nucleo\_F303K8\_IAR/Prog/main.c:



## Functions

- static void `Init` (void)  
*Initializes the microcontroller.*
- static void `SystemClock_Config` (void)  
*System Clock Configuration. This code was created by CubeMX and configures the system clock.*
- static void `VectorBase_Config` (void)  
*Vector base address configuration. It should no longer be at the start of flash memory but moved forward because the first part of flash is reserved for the bootloader. Note that this is already done by the bootloader before starting this program. Unfortunately, function `SystemInit()` overwrites this change again.*
- void `main` (void)  
*This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.*
- void `HAL_MspInit` (void)  
*Initializes the Global MSP. This function is called from `HAL_Init()` function to perform system level initialization (GPIOs, clock, DMA, interrupt).*
- void `HAL_MspDeInit` (void)  
*Deinitializes the Global MSP. This function is called from `HAL_DeInit()` function to perform system level Deinitialization (GPIOs, clock, DMA, interrupt).*

### 7.1033.1 Detailed Description

Demo program application source file.

### 7.1033.2 Function Documentation

#### 7.1033.2.1 HAL\_MspDeInit()

```
void HAL_MspDeInit (
 void)
```

Deinitializes the Global MSP. This function is called from `HAL_DeInit()` function to perform system level Deinitialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

#### 7.1033.2.2 HAL\_MspInit()

```
void HAL_MspInit (
 void)
```

Initializes the Global MSP. This function is called from `HAL_Init()` function to perform system level initialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

### 7.1033.2.3 Init()

```
static void Init (
 void) [static]
```

Initializes the microcontroller.

#### Returns

none.

Referenced by main().

### 7.1033.2.4 main()

```
void main (
 void)
```

This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.

#### Returns

None.

### 7.1033.2.5 SystemClock\_Config()

```
static void SystemClock_Config (
 void) [static]
```

System Clock Configuration. This code was created by CubeMX and configures the system clock.

#### Returns

none.

Referenced by Init().

## 7.1033.2.6 VectorBase\_Config()

```
static void VectorBase_Config (
 void) [static]
```

Vector base address configuration. It should no longer be at the start of flash memory but moved forward because the first part of flash is reserved for the bootloader. Note that this is already done by the bootloader before starting this program. Unfortunately, function SystemInit() overwrites this change again.

## Returns

none.

Referenced by Init().

## 7.1034 main.c File Reference

Bootloader application source file.

```
#include "boot.h"
#include "stm32f3xx.h"
#include "stm32f3xx_ll_rcc.h"
#include "stm32f3xx_ll_bus.h"
#include "stm32f3xx_ll_system.h"
#include "stm32f3xx_ll_utils.h"
#include "stm32f3xx_ll_usart.h"
#include "stm32f3xx_ll_gpio.h"
```

Include dependency graph for ARMC4\_STM32F3\_Nucleo\_F303K8\_Keil/Boot/main.c:



## Functions

- static void [Init](#) (void)  
*Initializes the microcontroller.*
- static void [SystemClock\\_Config](#) (void)  
*System Clock Configuration. This code was created by CubeMX and configures the system clock to match the configuration in the bootloader's configuration (blt\_conf.h), specifically the macros: BOOT\_CPU\_SYSTEM\_SPEED\_KHZ and BOOT\_CPU\_XTAL\_SPEED\_KHZ. Note that the Lower Layer drivers were selected in CubeMX for the RCC subsystem.*
- int [main](#) (void)  
*This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.*
- void [HAL\\_MspInit](#) (void)  
*Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).*
- void [HAL\\_MspDeInit](#) (void)  
*DeInitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level de-initialization (GPIOs, clock, DMA, interrupt).*

### 7.1034.1 Detailed Description

Bootloader application source file.

### 7.1034.2 Function Documentation

#### 7.1034.2.1 HAL\_MspDeInit()

```
void HAL_MspDeInit (
 void)
```

DeInitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level de-initialization (GPIOs, clock, DMA, interrupt).

##### Returns

none.

#### 7.1034.2.2 HAL\_MspInit()

```
void HAL_MspInit (
 void)
```

Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).

##### Returns

none.

#### 7.1034.2.3 Init()

```
static void Init (
 void) [static]
```

Initializes the microcontroller.

##### Returns

none.

Referenced by main().



#### 7.1034.2.4 main()

```
int main (
 void)
```

This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.

##### Returns

Program return code.

#### 7.1034.2.5 SystemClock\_Config()

```
static void SystemClock_Config (
 void) [static]
```

System Clock Configuration. This code was created by CubeMX and configures the system clock to match the configuration in the bootloader's configuration (blt\_conf.h), specifically the macros: BOOT\_CPU\_SYSTEM\_SPEED\_KHZ and BOOT\_CPU\_XTAL\_SPEED\_KHZ. Note that the Lower Layer drivers were selected in CubeMX for the RCC subsystem.

##### Returns

none.

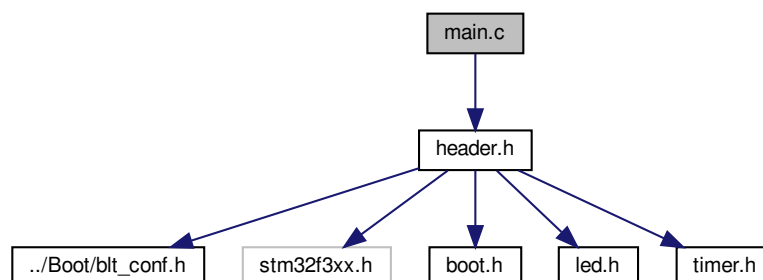
Referenced by Init().

## 7.1035 main.c File Reference

Demo program application source file.

```
#include "header.h"
```

Include dependency graph for ARMCM4\_STM32F3\_Nucleo\_F303K8\_Keil/Prog/main.c:



## Functions

- static void `Init` (void)  
*Initializes the microcontroller.*
- static void `SystemClock_Config` (void)  
*System Clock Configuration. This code was created by CubeMX and configures the system clock.*
- static void `VectorBase_Config` (void)  
*Vector base address configuration. It should no longer be at the start of flash memory but moved forward because the first part of flash is reserved for the bootloader. Note that this is already done by the bootloader before starting this program. Unfortunately, function `SystemInit()` overwrites this change again.*
- int `main` (void)  
*This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.*
- void `HAL_MspInit` (void)  
*Initializes the Global MSP. This function is called from `HAL_Init()` function to perform system level initialization (GPIOs, clock, DMA, interrupt).*
- void `HAL_MspDeInit` (void)  
*Deinitializes the Global MSP. This function is called from `HAL_DeInit()` function to perform system level Deinitialization (GPIOs, clock, DMA, interrupt).*

### 7.1035.1 Detailed Description

Demo program application source file.

### 7.1035.2 Function Documentation

#### 7.1035.2.1 `HAL_MspDeInit()`

```
void HAL_MspDeInit (
 void)
```

Deinitializes the Global MSP. This function is called from `HAL_DeInit()` function to perform system level Deinitialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

#### 7.1035.2.2 `HAL_MspInit()`

```
void HAL_MspInit (
 void)
```

Initializes the Global MSP. This function is called from `HAL_Init()` function to perform system level initialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

### 7.1035.2.3 Init()

```
static void Init (
 void) [static]
```

Initializes the microcontroller.

#### Returns

none.

Referenced by main().

### 7.1035.2.4 main()

```
int main (
 void)
```

This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.

#### Returns

Program exit code.

### 7.1035.2.5 SystemClock\_Config()

```
static void SystemClock_Config (
 void) [static]
```

System Clock Configuration. This code was created by CubeMX and configures the system clock.

#### Returns

none.

Referenced by Init().

### 7.1035.2.6 VectorBase\_Config()

```
static void VectorBase_Config (
 void) [static]
```

Vector base address configuration. It should no longer be at the start of flash memory but moved forward because the first part of flash is reserved for the bootloader. Note that this is already done by the bootloader before starting this program. Unfortunately, function SystemInit() overwrites this change again.

#### Returns

none.

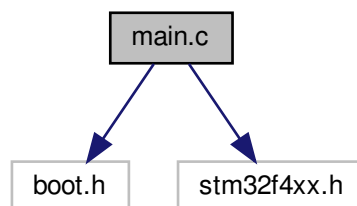
Referenced by Init().

## 7.1036 main.c File Reference

Bootloader application source file.

```
#include "boot.h"
#include "stm32f4xx.h"
#include "shared_params.h"
```

Include dependency graph for ARMCM4\_STM32F4\_Nucleo\_F429ZI\_GCC/Boot/main.c:



## Functions

- static void [Init](#) (void)  
*Initializes the microcontroller.*
- static void [SystemClock\\_Config](#) (void)  
*System Clock Configuration. This code was created by CubeMX and configures the system clock to match the configuration in the bootloader's configuration (blt\_conf.h), specifically the macros: BOOT\_CPU\_SYSTEM\_SPEED\_KHZ and BOOT\_CPU\_XTAL\_SPEED\_KHZ. Note that the Lower Layer drivers were selected in CubeMX for the RCC subsystem.*
- int [main](#) (void)  
*This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.*
- void [HAL\\_MspInit](#) (void)  
*Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).*
- void [HAL\\_MspDeInit](#) (void)  
*DeInitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level de-initialization (GPIOs, clock, DMA, interrupt).*

## 7.1036.1 Detailed Description

Bootloader application source file.

## 7.1036.2 Function Documentation

### 7.1036.2.1 HAL\_MspDeInit()

```
void HAL_MspDeInit (
 void)
```

DeInitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level de-initialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

### 7.1036.2.2 HAL\_MspInit()

```
void HAL_MspInit (
 void)
```

Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

### 7.1036.2.3 Init()

```
static void Init (
 void) [static]
```

Initializes the microcontroller.

#### Returns

none.

Referenced by main().

### 7.1036.2.4 main()

```
int main (
 void)
```

This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.

#### Returns

Program return code.

### 7.1036.2.5 SystemClock\_Config()

```
static void SystemClock_Config (
 void) [static]
```

System Clock Configuration. This code was created by CubeMX and configures the system clock to match the configuration in the bootloader's configuration (blt\_conf.h), specifically the macros: BOOT\_CPU\_SYSTEM\_SPEED\_KHZ and BOOT\_CPU\_XTAL\_SPEED\_KHZ. Note that the Lower Layer drivers were selected in CubeMX for the RCC subsystem.

#### Returns

none.

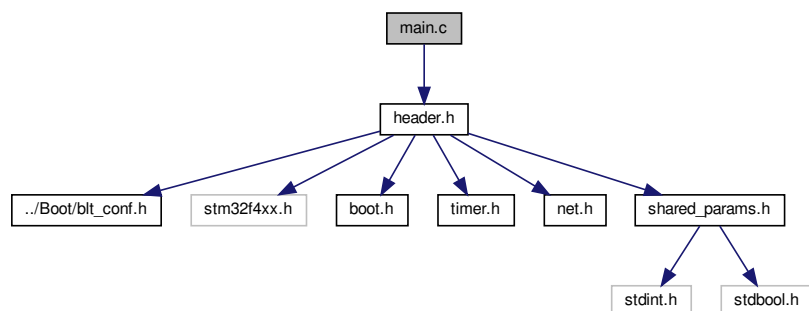
Referenced by Init().

## 7.1037 main.c File Reference

Demo program application source file.

```
#include "header.h"
```

Include dependency graph for ARMC4\_STM32F4\_Nucleo\_F429ZI\_GCC/Prog/main.c:



## Functions

- static void [Init](#) (void)  
*Initializes the microcontroller.*
- static void [SystemClock\\_Config](#) (void)  
*System Clock Configuration. This code was created by CubeMX and configures the system clock.*
- static void [VectorBase\\_Config](#) (void)  
*Vector base address configuration. It should no longer be at the start of flash memory but moved forward because the first part of flash is reserved for the bootloader. Note that this is already done by the bootloader before starting this program. Unfortunately, function SystemInit() overwrites this change again.*
- int [main](#) (void)  
*This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.*
- void [HAL\\_MspInit](#) (void)  
*Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).*
- void [HAL\\_MspDeInit](#) (void)  
*Deinitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level Deinitialization (GPIOs, clock, DMA, interrupt).*

### 7.1037.1 Detailed Description

Demo program application source file.

### 7.1037.2 Function Documentation

#### 7.1037.2.1 HAL\_MspDeInit()

```
void HAL_MspDeInit (
 void)
```

Deinitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level Deinitialization (GPIOs, clock, DMA, interrupt).

##### Returns

none.

#### 7.1037.2.2 HAL\_MspInit()

```
void HAL_MspInit (
 void)
```

Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).

##### Returns

none.

### 7.1037.2.3 Init()

```
static void Init (
 void) [static]
```

Initializes the microcontroller.

#### Returns

none.

Referenced by main().

### 7.1037.2.4 main()

```
int main (
 void)
```

This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.

#### Returns

Program exit code.

### 7.1037.2.5 SystemClock\_Config()

```
static void SystemClock_Config (
 void) [static]
```

System Clock Configuration. This code was created by CubeMX and configures the system clock.

#### Returns

none.

Referenced by Init().



## 7.1037.2.6 VectorBase\_Config()

```
static void VectorBase_Config (
 void) [static]
```

Vector base address configuration. It should no longer be at the start of flash memory but moved forward because the first part of flash is reserved for the bootloader. Note that this is already done by the bootloader before starting this program. Unfortunately, function SystemInit() overwrites this change again.

## Returns

none.

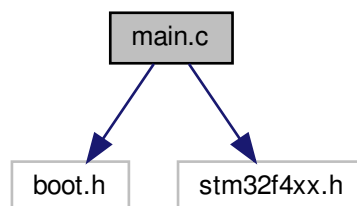
Referenced by Init().

## 7.1038 main.c File Reference

Bootloader application source file.

```
#include "boot.h"
#include "stm32f4xx.h"
#include "shared_params.h"
```

Include dependency graph for ARMCM4\_STM32F4\_Nucleo\_F429ZI\_IAR/Boot/main.c:



## Functions

- static void [Init](#) (void)  
*Initializes the microcontroller.*
- static void [SystemClock\\_Config](#) (void)  
*System Clock Configuration. This code was created by CubeMX and configures the system clock to match the configuration in the bootloader's configuration (blt\_conf.h), specifically the macros: BOOT\_CPU\_SYSTEM\_SPEED\_KHZ and BOOT\_CPU\_XTAL\_SPEED\_KHZ. Note that the Lower Layer drivers were selected in CubeMX for the RCC subsystem.*
- void [main](#) (void)  
*This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.*
- void [HAL\\_MspInit](#) (void)  
*Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).*
- void [HAL\\_MspDeInit](#) (void)  
*DeInitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level de-initialization (GPIOs, clock, DMA, interrupt).*

### 7.1038.1 Detailed Description

Bootloader application source file.

### 7.1038.2 Function Documentation

#### 7.1038.2.1 HAL\_MspDeInit()

```
void HAL_MspDeInit (
 void)
```

DeInitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level de-initialization (GPIOs, clock, DMA, interrupt).

##### Returns

none.

#### 7.1038.2.2 HAL\_MspInit()

```
void HAL_MspInit (
 void)
```

Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).

##### Returns

none.

#### 7.1038.2.3 Init()

```
static void Init (
 void) [static]
```

Initializes the microcontroller.

##### Returns

none.

Referenced by main().

#### 7.1038.2.4 main()

```
void main (
 void)
```

This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.

##### Returns

none.

#### 7.1038.2.5 SystemClock\_Config()

```
static void SystemClock_Config (
 void) [static]
```

System Clock Configuration. This code was created by CubeMX and configures the system clock to match the configuration in the bootloader's configuration (blt\_conf.h), specifically the macros: BOOT\_CPU\_SYSTEM\_SPEED\_KHZ and BOOT\_CPU\_XTAL\_SPEED\_KHZ. Note that the Lower Layer drivers were selected in CubeMX for the RCC subsystem.

##### Returns

none.

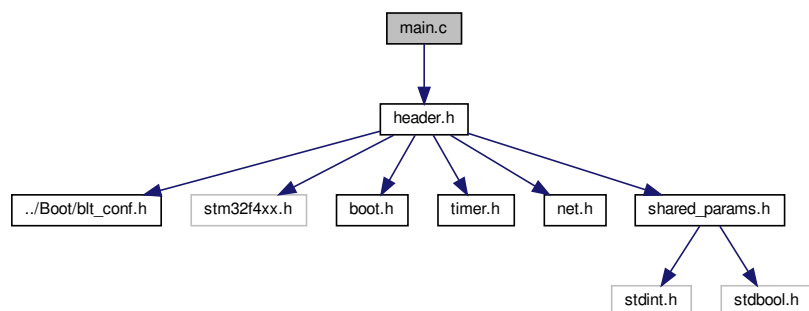
Referenced by Init().

## 7.1039 main.c File Reference

Demo program application source file.

```
#include "header.h"
```

Include dependency graph for ARMC4\_STM32F4\_Nucleo\_F429ZI\_IAR/Prog/main.c:



## Functions

- static void `Init` (void)  
*Initializes the microcontroller.*
- static void `SystemClock_Config` (void)  
*System Clock Configuration. This code was created by CubeMX and configures the system clock.*
- static void `VectorBase_Config` (void)  
*Vector base address configuration. It should no longer be at the start of flash memory but moved forward because the first part of flash is reserved for the bootloader. Note that this is already done by the bootloader before starting this program. Unfortunately, function `SystemInit()` overwrites this change again.*
- void `main` (void)  
*This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.*
- void `HAL_MspInit` (void)  
*Initializes the Global MSP. This function is called from `HAL_Init()` function to perform system level initialization (GPIOs, clock, DMA, interrupt).*
- void `HAL_MspDeInit` (void)  
*Deinitializes the Global MSP. This function is called from `HAL_DeInit()` function to perform system level Deinitialization (GPIOs, clock, DMA, interrupt).*

### 7.1039.1 Detailed Description

Demo program application source file.

### 7.1039.2 Function Documentation

#### 7.1039.2.1 HAL\_MspDeInit()

```
void HAL_MspDeInit (
 void)
```

Deinitializes the Global MSP. This function is called from `HAL_DeInit()` function to perform system level Deinitialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

#### 7.1039.2.2 HAL\_MspInit()

```
void HAL_MspInit (
 void)
```

Initializes the Global MSP. This function is called from `HAL_Init()` function to perform system level initialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

### 7.1039.2.3 Init()

```
static void Init (
 void) [static]
```

Initializes the microcontroller.

#### Returns

none.

Referenced by main().

### 7.1039.2.4 main()

```
void main (
 void)
```

This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.

#### Returns

None.

### 7.1039.2.5 SystemClock\_Config()

```
static void SystemClock_Config (
 void) [static]
```

System Clock Configuration. This code was created by CubeMX and configures the system clock.

#### Returns

none.

Referenced by Init().

### 7.1039.2.6 VectorBase\_Config()

```
static void VectorBase_Config (
 void) [static]
```

Vector base address configuration. It should no longer be at the start of flash memory but moved forward because the first part of flash is reserved for the bootloader. Note that this is already done by the bootloader before starting this program. Unfortunately, function SystemInit() overwrites this change again.

#### Returns

none.

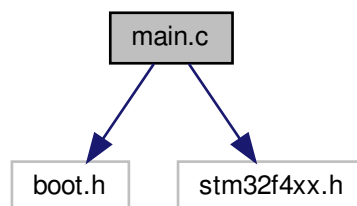
Referenced by Init().

## 7.1040 main.c File Reference

Bootloader application source file.

```
#include "boot.h"
#include "stm32f4xx.h"
#include "shared_params.h"
```

Include dependency graph for ARMCM4\_STM32F4\_Nucleo\_F429ZI\_Keil/Boot/main.c:



## Functions

- static void [Init](#) (void)  
*Initializes the microcontroller.*
- static void [SystemClock\\_Config](#) (void)  
*System Clock Configuration. This code was created by CubeMX and configures the system clock to match the configuration in the bootloader's configuration (blt\_conf.h), specifically the macros: BOOT\_CPU\_SYSTEM\_SPEED\_KHZ and BOOT\_CPU\_XTAL\_SPEED\_KHZ. Note that the Lower Layer drivers were selected in CubeMX for the RCC subsystem.*
- int [main](#) (void)  
*This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.*
- void [HAL\\_MspInit](#) (void)  
*Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).*
- void [HAL\\_MspDeInit](#) (void)  
*DeInitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level de-initialization (GPIOs, clock, DMA, interrupt).*

## 7.1040.1 Detailed Description

Bootloader application source file.

## 7.1040.2 Function Documentation

### 7.1040.2.1 HAL\_MspDeInit()

```
void HAL_MspDeInit (
 void)
```

DeInitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level de-initialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

### 7.1040.2.2 HAL\_MspInit()

```
void HAL_MspInit (
 void)
```

Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

### 7.1040.2.3 Init()

```
static void Init (
 void) [static]
```

Initializes the microcontroller.

#### Returns

none.

Referenced by main().

#### 7.1040.2.4 main()

```
int main (
 void)
```

This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.

##### Returns

Program return code.

#### 7.1040.2.5 SystemClock\_Config()

```
static void SystemClock_Config (
 void) [static]
```

System Clock Configuration. This code was created by CubeMX and configures the system clock to match the configuration in the bootloader's configuration (blt\_conf.h), specifically the macros: BOOT\_CPU\_SYSTEM\_SPEED\_KHZ and BOOT\_CPU\_XTAL\_SPEED\_KHZ. Note that the Lower Layer drivers were selected in CubeMX for the RCC subsystem.

##### Returns

none.

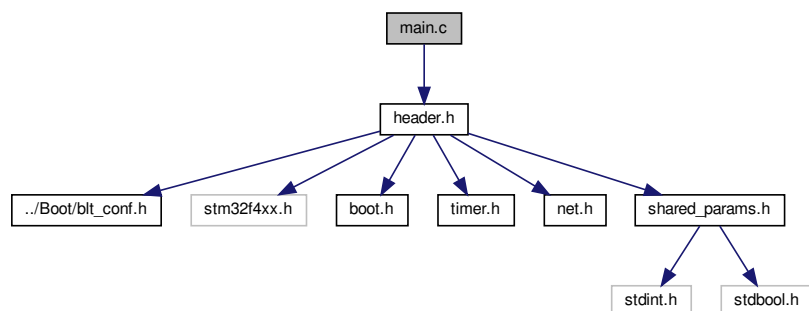
Referenced by Init().

## 7.1041 main.c File Reference

Demo program application source file.

```
#include "header.h"
```

Include dependency graph for ARMC4\_STM32F4\_Nucleo\_F429ZI\_Keil/Prog/main.c:





## Functions

- static void [Init](#) (void)  
*Initializes the microcontroller.*
- static void [SystemClock\\_Config](#) (void)  
*System Clock Configuration. This code was created by CubeMX and configures the system clock.*
- static void [VectorBase\\_Config](#) (void)  
*Vector base address configuration. It should no longer be at the start of flash memory but moved forward because the first part of flash is reserved for the bootloader. Note that this is already done by the bootloader before starting this program. Unfortunately, function SystemInit() overwrites this change again.*
- int [main](#) (void)  
*This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.*
- void [HAL\\_MspInit](#) (void)  
*Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).*
- void [HAL\\_MspDeInit](#) (void)  
*Deinitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level Deinitialization (GPIOs, clock, DMA, interrupt).*

### 7.1041.1 Detailed Description

Demo program application source file.

### 7.1041.2 Function Documentation

#### 7.1041.2.1 HAL\_MspDeInit()

```
void HAL_MspDeInit (
 void)
```

Deinitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level Deinitialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

#### 7.1041.2.2 HAL\_MspInit()

```
void HAL_MspInit (
 void)
```

Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

### 7.1041.2.3 Init()

```
static void Init (
 void) [static]
```

Initializes the microcontroller.

#### Returns

none.

Referenced by main().

### 7.1041.2.4 main()

```
int main (
 void)
```

This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.

#### Returns

Program exit code.

### 7.1041.2.5 SystemClock\_Config()

```
static void SystemClock_Config (
 void) [static]
```

System Clock Configuration. This code was created by CubeMX and configures the system clock.

#### Returns

none.

Referenced by Init().

## 7.1041.2.6 VectorBase\_Config()

```
static void VectorBase_Config (
 void) [static]
```

Vector base address configuration. It should no longer be at the start of flash memory but moved forward because the first part of flash is reserved for the bootloader. Note that this is already done by the bootloader before starting this program. Unfortunately, function SystemInit() overwrites this change again.

## Returns

none.

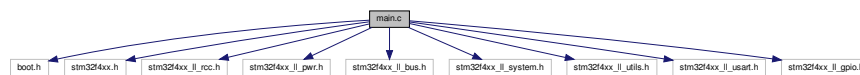
Referenced by Init().

## 7.1042 main.c File Reference

Bootloader application source file.

```
#include "boot.h"
#include "stm32f4xx.h"
#include "stm32f4xx_ll_rcc.h"
#include "stm32f4xx_ll_pwr.h"
#include "stm32f4xx_ll_bus.h"
#include "stm32f4xx_ll_system.h"
#include "stm32f4xx_ll_utils.h"
#include "stm32f4xx_ll_usart.h"
#include "stm32f4xx_ll_gpio.h"
```

Include dependency graph for ARMCM4\_STM32F4\_Olimex\_STM32P405\_GCC/Boot/main.c:



## Functions

- static void [Init](#) (void)  
*Initializes the microcontroller.*
- static void [SystemClock\\_Config](#) (void)  
*System Clock Configuration. This code was created by CubeMX and configures the system clock to match the configuration in the bootloader's configuration (blt\_conf.h), specifically the macros: BOOT\_CPU\_SYSTEM\_SPEED\_KHZ and BOOT\_CPU\_XTAL\_SPEED\_KHZ. Note that the Lower Layer drivers were selected in CubeMX for the RCC subsystem.*
- int [main](#) (void)  
*This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.*
- void [HAL\\_MspInit](#) (void)  
*Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).*
- void [HAL\\_MspDeInit](#) (void)  
*DeInitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level de-initialization (GPIOs, clock, DMA, interrupt).*

### 7.1042.1 Detailed Description

Bootloader application source file.

### 7.1042.2 Function Documentation

#### 7.1042.2.1 HAL\_MspDeInit()

```
void HAL_MspDeInit (
 void)
```

DeInitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level de-initialization (GPIOs, clock, DMA, interrupt).

##### Returns

none.

#### 7.1042.2.2 HAL\_MspInit()

```
void HAL_MspInit (
 void)
```

Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).

##### Returns

none.

#### 7.1042.2.3 Init()

```
static void Init (
 void) [static]
```

Initializes the microcontroller.

##### Returns

none.

Referenced by main().

#### 7.1042.2.4 main()

```
int main (
 void)
```

This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.

##### Returns

Program exit code.

#### 7.1042.2.5 SystemClock\_Config()

```
static void SystemClock_Config (
 void) [static]
```

System Clock Configuration. This code was created by CubeMX and configures the system clock to match the configuration in the bootloader's configuration (blt\_conf.h), specifically the macros: BOOT\_CPU\_SYSTEM\_SPEED\_KHZ and BOOT\_CPU\_XTAL\_SPEED\_KHZ. Note that the Lower Layer drivers were selected in CubeMX for the RCC subsystem.

##### Returns

none.

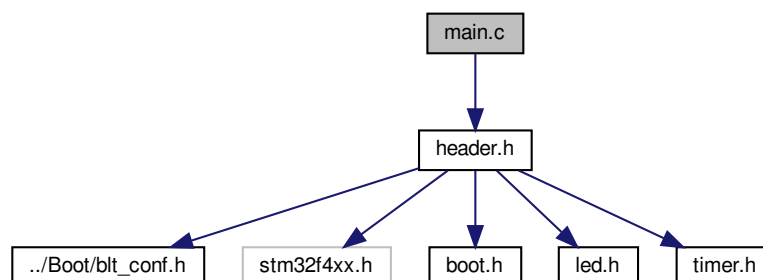
Referenced by Init().

## 7.1043 main.c File Reference

Demo program application source file.

```
#include "header.h"
```

Include dependency graph for ARMCM4\_STM32F4\_Olimex\_STM32P405\_GCC/Prog/main.c:



## Functions

- static void [Init](#) (void)  
*Initializes the microcontroller.*
- static void [SystemClock\\_Config](#) (void)  
*System Clock Configuration. This code was created by CubeMX and configures the system clock.*
- static void [VectorBase\\_Config](#) (void)  
*Vector base address configuration. It should no longer be at the start of flash memory but moved forward because the first part of flash is reserved for the bootloader. Note that this is already done by the bootloader before starting this program. Unfortunately, function SystemInit() overwrites this change again.*
- int [main](#) (void)  
*This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.*
- void [HAL\\_MspInit](#) (void)  
*Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).*
- void [HAL\\_MspDeInit](#) (void)  
*Deinitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level Deinitialization (GPIOs, clock, DMA, interrupt).*

### 7.1043.1 Detailed Description

Demo program application source file.

### 7.1043.2 Function Documentation

#### 7.1043.2.1 HAL\_MspDeInit()

```
void HAL_MspDeInit (
 void)
```

Deinitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level Deinitialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

#### 7.1043.2.2 HAL\_MspInit()

```
void HAL_MspInit (
 void)
```

Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

### 7.1043.2.3 Init()

```
static void Init (
 void) [static]
```

Initializes the microcontroller.

#### Returns

none.

Referenced by main().

### 7.1043.2.4 main()

```
int main (
 void)
```

This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.

#### Returns

Program exit code.

### 7.1043.2.5 SystemClock\_Config()

```
static void SystemClock_Config (
 void) [static]
```

System Clock Configuration. This code was created by CubeMX and configures the system clock.

#### Returns

none.

Referenced by Init().

### 7.1043.2.6 VectorBase\_Config()

```
static void VectorBase_Config (
 void) [static]
```

Vector base address configuration. It should no longer be at the start of flash memory but moved forward because the first part of flash is reserved for the bootloader. Note that this is already done by the bootloader before starting this program. Unfortunately, function SystemInit() overwrites this change again.

#### Returns

none.

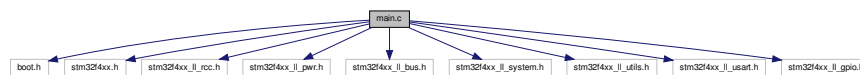
Referenced by Init().

## 7.1044 main.c File Reference

Bootloader application source file.

```
#include "boot.h"
#include "stm32f4xx.h"
#include "stm32f4xx_ll_rcc.h"
#include "stm32f4xx_ll_pwr.h"
#include "stm32f4xx_ll_bus.h"
#include "stm32f4xx_ll_system.h"
#include "stm32f4xx_ll_utils.h"
#include "stm32f4xx_ll_usart.h"
#include "stm32f4xx_ll_gpio.h"
```

Include dependency graph for ARMC4\_STM32F4\_Olimex\_STM32P405\_IAR/Boot/main.c:



## Functions

- static void [Init](#) (void)  
*Initializes the microcontroller.*
- static void [SystemClock\\_Config](#) (void)  
*System Clock Configuration. This code was created by CubeMX and configures the system clock to match the configuration in the bootloader's configuration (blt\_conf.h), specifically the macros: BOOT\_CPU\_SYSTEM\_SPEED\_KHZ and BOOT\_CPU\_XTAL\_SPEED\_KHZ. Note that the Lower Layer drivers were selected in CubeMX for the RCC subsystem.*
- void [main](#) (void)  
*This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.*
- void [HAL\\_MspInit](#) (void)  
*Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).*
- void [HAL\\_MspDeInit](#) (void)  
*DeInitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level de-initialization (GPIOs, clock, DMA, interrupt).*



### 7.1044.1 Detailed Description

Bootloader application source file.

### 7.1044.2 Function Documentation

#### 7.1044.2.1 HAL\_MspDeInit()

```
void HAL_MspDeInit (
 void)
```

DeInitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level de-initialization (GPIOs, clock, DMA, interrupt).

##### Returns

none.

#### 7.1044.2.2 HAL\_MspInit()

```
void HAL_MspInit (
 void)
```

Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).

##### Returns

none.

#### 7.1044.2.3 Init()

```
static void Init (
 void) [static]
```

Initializes the microcontroller.

##### Returns

none.

Referenced by main().

#### 7.1044.2.4 main()

```
void main (
 void)
```

This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.

#### Returns

None.

#### 7.1044.2.5 SystemClock\_Config()

```
static void SystemClock_Config (
 void) [static]
```

System Clock Configuration. This code was created by CubeMX and configures the system clock to match the configuration in the bootloader's configuration (blt\_conf.h), specifically the macros: BOOT\_CPU\_SYSTEM\_SPEED\_KHZ and BOOT\_CPU\_XTAL\_SPEED\_KHZ. Note that the Lower Layer drivers were selected in CubeMX for the RCC subsystem.

#### Returns

none.

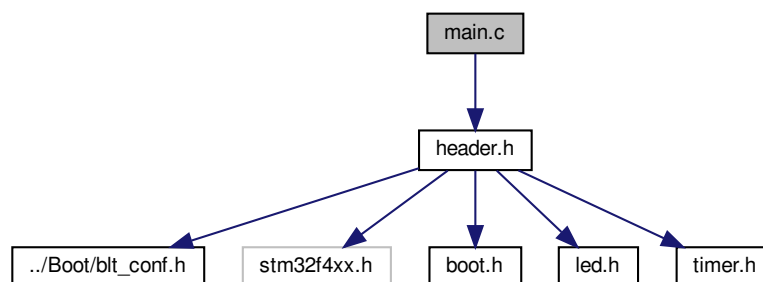
Referenced by Init().

## 7.1045 main.c File Reference

Demo program application source file.

```
#include "header.h"
```

Include dependency graph for ARMCM4\_STM32F4\_Olimex\_STM32P405\_IAR/Prog/main.c:



## Functions

- static void `Init` (void)  
*Initializes the microcontroller.*
- static void `SystemClock_Config` (void)  
*System Clock Configuration. This code was created by CubeMX and configures the system clock.*
- static void `VectorBase_Config` (void)  
*Vector base address configuration. It should no longer be at the start of flash memory but moved forward because the first part of flash is reserved for the bootloader. Note that this is already done by the bootloader before starting this program. Unfortunately, function `SystemInit()` overwrites this change again.*
- void `main` (void)  
*This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.*
- void `HAL_MspInit` (void)  
*Initializes the Global MSP. This function is called from `HAL_Init()` function to perform system level initialization (GPIOs, clock, DMA, interrupt).*
- void `HAL_MspDeInit` (void)  
*Deinitializes the Global MSP. This function is called from `HAL_DeInit()` function to perform system level Deinitialization (GPIOs, clock, DMA, interrupt).*

### 7.1045.1 Detailed Description

Demo program application source file.

### 7.1045.2 Function Documentation

#### 7.1045.2.1 `HAL_MspDeInit()`

```
void HAL_MspDeInit (
 void)
```

Deinitializes the Global MSP. This function is called from `HAL_DeInit()` function to perform system level Deinitialization (GPIOs, clock, DMA, interrupt).

##### Returns

none.

#### 7.1045.2.2 `HAL_MspInit()`

```
void HAL_MspInit (
 void)
```

Initializes the Global MSP. This function is called from `HAL_Init()` function to perform system level initialization (GPIOs, clock, DMA, interrupt).

##### Returns

none.

### 7.1045.2.3 Init()

```
static void Init (
 void) [static]
```

Initializes the microcontroller.

#### Returns

none.

Referenced by main().

### 7.1045.2.4 main()

```
void main (
 void)
```

This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.

#### Returns

None.

### 7.1045.2.5 SystemClock\_Config()

```
static void SystemClock_Config (
 void) [static]
```

System Clock Configuration. This code was created by CubeMX and configures the system clock.

#### Returns

none.

Referenced by Init().

## 7.1045.2.6 VectorBase\_Config()

```
static void VectorBase_Config (
 void) [static]
```

Vector base address configuration. It should no longer be at the start of flash memory but moved forward because the first part of flash is reserved for the bootloader. Note that this is already done by the bootloader before starting this program. Unfortunately, function SystemInit() overwrites this change again.

## Returns

none.

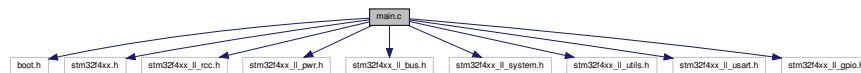
Referenced by Init().

## 7.1046 main.c File Reference

Bootloader application source file.

```
#include "boot.h"
#include "stm32f4xx.h"
#include "stm32f4xx_ll_rcc.h"
#include "stm32f4xx_ll_pwr.h"
#include "stm32f4xx_ll_bus.h"
#include "stm32f4xx_ll_system.h"
#include "stm32f4xx_ll_utils.h"
#include "stm32f4xx_ll_usart.h"
#include "stm32f4xx_ll_gpio.h"
```

Include dependency graph for ARMC44\_STM32F4\_Olimex\_STM32P405\_Keil/Boot/main.c:



## Functions

- static void [Init](#) (void)  
*Initializes the microcontroller.*
- static void [SystemClock\\_Config](#) (void)  
*System Clock Configuration. This code was created by CubeMX and configures the system clock to match the configuration in the bootloader's configuration (blt\_conf.h), specifically the macros: BOOT\_CPU\_SYSTEM\_SPEED\_KHZ and BOOT\_CPU\_XTAL\_SPEED\_KHZ. Note that the Lower Layer drivers were selected in CubeMX for the RCC subsystem.*
- int [main](#) (void)  
*This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.*
- void [HAL\\_MspInit](#) (void)  
*Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).*
- void [HAL\\_MspDeInit](#) (void)  
*DeInitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level de-initialization (GPIOs, clock, DMA, interrupt).*

### 7.1046.1 Detailed Description

Bootloader application source file.

### 7.1046.2 Function Documentation

#### 7.1046.2.1 HAL\_MspDeInit()

```
void HAL_MspDeInit (
 void)
```

DeInitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level de-initialization (GPIOs, clock, DMA, interrupt).

##### Returns

none.

#### 7.1046.2.2 HAL\_MspInit()

```
void HAL_MspInit (
 void)
```

Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).

##### Returns

none.

#### 7.1046.2.3 Init()

```
static void Init (
 void) [static]
```

Initializes the microcontroller.

##### Returns

none.

Referenced by main().

#### 7.1046.2.4 main()

```
int main (
 void)
```

This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.

##### Returns

Program exit code.

#### 7.1046.2.5 SystemClock\_Config()

```
static void SystemClock_Config (
 void) [static]
```

System Clock Configuration. This code was created by CubeMX and configures the system clock to match the configuration in the bootloader's configuration (blt\_conf.h), specifically the macros: BOOT\_CPU\_SYSTEM\_SPEED\_KHZ and BOOT\_CPU\_XTAL\_SPEED\_KHZ. Note that the Lower Layer drivers were selected in CubeMX for the RCC subsystem.

##### Returns

none.

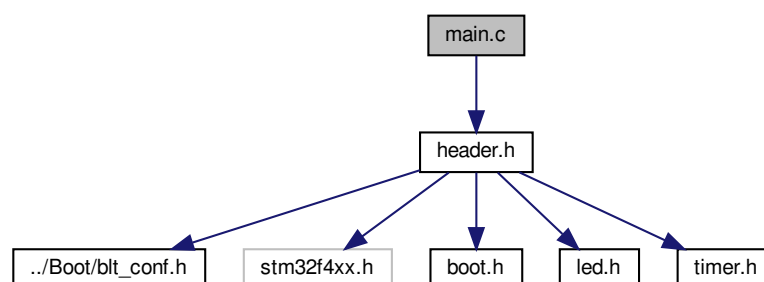
Referenced by Init().

## 7.1047 main.c File Reference

Demo program application source file.

```
#include "header.h"
```

Include dependency graph for ARMCM4\_STM32F4\_Olimex\_STM32P405\_Keil/Prog/main.c:



## Functions

- static void [Init](#) (void)  
*Initializes the microcontroller.*
- static void [SystemClock\\_Config](#) (void)  
*System Clock Configuration. This code was created by CubeMX and configures the system clock.*
- static void [VectorBase\\_Config](#) (void)  
*Vector base address configuration. It should no longer be at the start of flash memory but moved forward because the first part of flash is reserved for the bootloader. Note that this is already done by the bootloader before starting this program. Unfortunately, function SystemInit() overwrites this change again.*
- int [main](#) (void)  
*This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.*
- void [HAL\\_MspInit](#) (void)  
*Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).*
- void [HAL\\_MspDeInit](#) (void)  
*Deinitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level Deinitialization (GPIOs, clock, DMA, interrupt).*

### 7.1047.1 Detailed Description

Demo program application source file.

### 7.1047.2 Function Documentation

#### 7.1047.2.1 HAL\_MspDeInit()

```
void HAL_MspDeInit (
 void)
```

Deinitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level Deinitialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

#### 7.1047.2.2 HAL\_MspInit()

```
void HAL_MspInit (
 void)
```

Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.



### 7.1047.2.3 Init()

```
static void Init (
 void) [static]
```

Initializes the microcontroller.

#### Returns

none.

Referenced by main().

### 7.1047.2.4 main()

```
int main (
 void)
```

This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.

#### Returns

Program exit code.

### 7.1047.2.5 SystemClock\_Config()

```
static void SystemClock_Config (
 void) [static]
```

System Clock Configuration. This code was created by CubeMX and configures the system clock.

#### Returns

none.

Referenced by Init().

### 7.1047.2.6 VectorBase\_Config()

```
static void VectorBase_Config (
 void) [static]
```

Vector base address configuration. It should no longer be at the start of flash memory but moved forward because the first part of flash is reserved for the bootloader. Note that this is already done by the bootloader before starting this program. Unfortunately, function SystemInit() overwrites this change again.

#### Returns

none.

Referenced by Init().

## 7.1048 main.c File Reference

Bootloader application source file.

```
#include "boot.h"
#include "stm32l4xx.h"
#include "stm32l4xx_ll_rcc.h"
#include "stm32l4xx_ll_bus.h"
#include "stm32l4xx_ll_pwr.h"
#include "stm32l4xx_ll_system.h"
#include "stm32l4xx_ll_utils.h"
#include "stm32l4xx_ll_gpio.h"
```

Include dependency graph for ARMC4\_STM32L4\_Nucleo\_L476RG\_GCC/Boot/main.c:



## Functions

- static void [Init](#) (void)  
*Initializes the microcontroller.*
- static void [SystemClock\\_Config](#) (void)  
*System Clock Configuration. This code was created by CubeMX and configures the system clock to match the configuration in the bootloader's configuration (blt\_conf.h), specifically the macros: BOOT\_CPU\_SYSTEM\_SPEED\_KHZ and BOOT\_CPU\_XTAL\_SPEED\_KHZ. Note that the Lower Layer drivers were selected in CubeMX for the RCC subsystem.*
- int [main](#) (void)  
*This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.*
- void [HAL\\_MspInit](#) (void)  
*Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).*
- void [HAL\\_MspDeInit](#) (void)  
*DeInitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level de-initialization (GPIOs, clock, DMA, interrupt).*

## 7.1048.1 Detailed Description

Bootloader application source file.

## 7.1048.2 Function Documentation

### 7.1048.2.1 HAL\_MspDeInit()

```
void HAL_MspDeInit (
 void)
```

DeInitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level de-initialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

### 7.1048.2.2 HAL\_MspInit()

```
void HAL_MspInit (
 void)
```

Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

### 7.1048.2.3 Init()

```
static void Init (
 void) [static]
```

Initializes the microcontroller.

#### Returns

none.

Referenced by main().

#### 7.1048.2.4 main()

```
int main (
 void)
```

This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.

##### Returns

Program return code.

#### 7.1048.2.5 SystemClock\_Config()

```
static void SystemClock_Config (
 void) [static]
```

System Clock Configuration. This code was created by CubeMX and configures the system clock to match the configuration in the bootloader's configuration (blt\_conf.h), specifically the macros: BOOT\_CPU\_SYSTEM\_SPEED\_KHZ and BOOT\_CPU\_XTAL\_SPEED\_KHZ. Note that the Lower Layer drivers were selected in CubeMX for the RCC subsystem.

##### Returns

none.

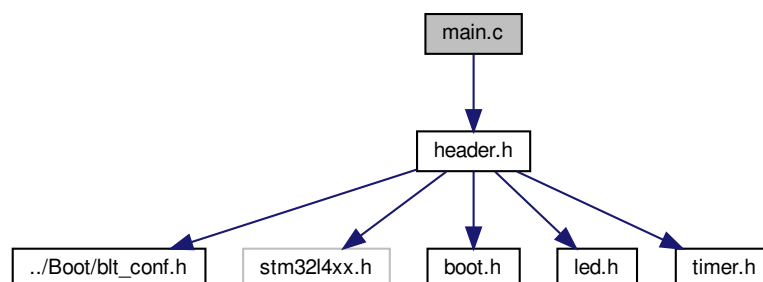
Referenced by Init().

## 7.1049 main.c File Reference

Demo program application source file.

```
#include "header.h"
```

Include dependency graph for ARMCM4\_STM32L4\_Nucleo\_L476RG\_GCC/Prog/main.c:



## Functions

- static void [Init](#) (void)  
*Initializes the microcontroller.*
- static void [SystemClock\\_Config](#) (void)  
*System Clock Configuration. This code was created by CubeMX and configures the system clock.*
- static void [VectorBase\\_Config](#) (void)  
*Vector base address configuration. It should no longer be at the start of flash memory but moved forward because the first part of flash is reserved for the bootloader. Note that this is already done by the bootloader before starting this program. Unfortunately, function SystemInit() overwrites this change again.*
- int [main](#) (void)  
*This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.*
- void [HAL\\_MspInit](#) (void)  
*Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).*
- void [HAL\\_MspDeInit](#) (void)  
*Deinitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level Deinitialization (GPIOs, clock, DMA, interrupt).*

### 7.1049.1 Detailed Description

Demo program application source file.

### 7.1049.2 Function Documentation

#### 7.1049.2.1 HAL\_MspDeInit()

```
void HAL_MspDeInit (
 void)
```

Deinitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level Deinitialization (GPIOs, clock, DMA, interrupt).

##### Returns

none.

#### 7.1049.2.2 HAL\_MspInit()

```
void HAL_MspInit (
 void)
```

Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).

##### Returns

none.

### 7.1049.2.3 Init()

```
static void Init (
 void) [static]
```

Initializes the microcontroller.

#### Returns

none.

Referenced by main().

### 7.1049.2.4 main()

```
int main (
 void)
```

This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.

#### Returns

Program exit code.

### 7.1049.2.5 SystemClock\_Config()

```
static void SystemClock_Config (
 void) [static]
```

System Clock Configuration. This code was created by CubeMX and configures the system clock.

#### Returns

none.

Referenced by Init().

## 7.1049.2.6 VectorBase\_Config()

```
static void VectorBase_Config (
 void) [static]
```

Vector base address configuration. It should no longer be at the start of flash memory but moved forward because the first part of flash is reserved for the bootloader. Note that this is already done by the bootloader before starting this program. Unfortunately, function SystemInit() overwrites this change again.

## Returns

none.

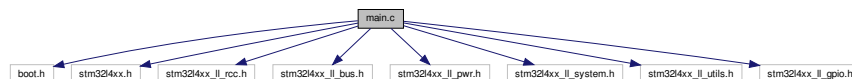
Referenced by Init().

## 7.1050 main.c File Reference

Bootloader application source file.

```
#include "boot.h"
#include "stm32l4xx.h"
#include "stm32l4xx_ll_rcc.h"
#include "stm32l4xx_ll_bus.h"
#include "stm32l4xx_ll_pwr.h"
#include "stm32l4xx_ll_system.h"
#include "stm32l4xx_ll_utils.h"
#include "stm32l4xx_ll_gpio.h"
```

Include dependency graph for ARMC4\_STM32L4\_Nucleo\_L476RG\_IAR/Boot/main.c:



## Functions

- static void [Init](#) (void)  
*Initializes the microcontroller.*
- static void [SystemClock\\_Config](#) (void)  
*System Clock Configuration. This code was created by CubeMX and configures the system clock to match the configuration in the bootloader's configuration (blt\_conf.h), specifically the macros: BOOT\_CPU\_SYSTEM\_SPEED\_KHZ and BOOT\_CPU\_XTAL\_SPEED\_KHZ. Note that the Lower Layer drivers were selected in CubeMX for the RCC subsystem.*
- int [main](#) (void)  
*This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.*
- void [HAL\\_MspInit](#) (void)  
*Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).*
- void [HAL\\_MspDeInit](#) (void)  
*DeInitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level de-initialization (GPIOs, clock, DMA, interrupt).*

### 7.1050.1 Detailed Description

Bootloader application source file.

### 7.1050.2 Function Documentation

#### 7.1050.2.1 HAL\_MspDeInit()

```
void HAL_MspDeInit (
 void)
```

DeInitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level de-initialization (GPIOs, clock, DMA, interrupt).

##### Returns

none.

#### 7.1050.2.2 HAL\_MspInit()

```
void HAL_MspInit (
 void)
```

Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).

##### Returns

none.

#### 7.1050.2.3 Init()

```
static void Init (
 void) [static]
```

Initializes the microcontroller.

##### Returns

none.

Referenced by main().



#### 7.1050.2.4 main()

```
int main (
 void)
```

This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.

##### Returns

none.

#### 7.1050.2.5 SystemClock\_Config()

```
static void SystemClock_Config (
 void) [static]
```

System Clock Configuration. This code was created by CubeMX and configures the system clock to match the configuration in the bootloader's configuration (blt\_conf.h), specifically the macros: BOOT\_CPU\_SYSTEM\_SPEED\_KHZ and BOOT\_CPU\_XTAL\_SPEED\_KHZ. Note that the Lower Layer drivers were selected in CubeMX for the RCC subsystem.

##### Returns

none.

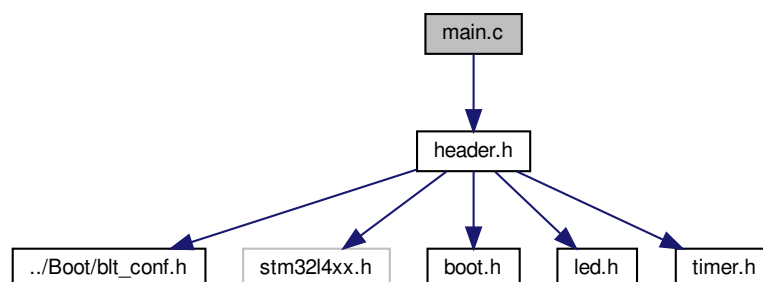
Referenced by Init().

## 7.1051 main.c File Reference

Demo program application source file.

```
#include "header.h"
```

Include dependency graph for ARMCM4\_STM32L4\_Nucleo\_L476RG\_IAR/Prog/main.c:



## Functions

- static void [Init](#) (void)  
*Initializes the microcontroller.*
- static void [SystemClock\\_Config](#) (void)  
*System Clock Configuration. This code was created by CubeMX and configures the system clock.*
- static void [VectorBase\\_Config](#) (void)  
*Vector base address configuration. It should no longer be at the start of flash memory but moved forward because the first part of flash is reserved for the bootloader. Note that this is already done by the bootloader before starting this program. Unfortunately, function SystemInit() overwrites this change again.*
- int [main](#) (void)  
*This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.*
- void [HAL\\_MspInit](#) (void)  
*Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).*
- void [HAL\\_MspDeInit](#) (void)  
*Deinitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level Deinitialization (GPIOs, clock, DMA, interrupt).*

### 7.1051.1 Detailed Description

Demo program application source file.

### 7.1051.2 Function Documentation

#### 7.1051.2.1 HAL\_MspDeInit()

```
void HAL_MspDeInit (
 void)
```

Deinitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level Deinitialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

#### 7.1051.2.2 HAL\_MspInit()

```
void HAL_MspInit (
 void)
```

Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

### 7.1051.2.3 Init()

```
static void Init (
 void) [static]
```

Initializes the microcontroller.

#### Returns

none.

Referenced by main().

### 7.1051.2.4 main()

```
int main (
 void)
```

This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.

#### Returns

none.

### 7.1051.2.5 SystemClock\_Config()

```
static void SystemClock_Config (
 void) [static]
```

System Clock Configuration. This code was created by CubeMX and configures the system clock.

#### Returns

none.

Referenced by Init().

### 7.1051.2.6 VectorBase\_Config()

```
static void VectorBase_Config (
 void) [static]
```

Vector base address configuration. It should no longer be at the start of flash memory but moved forward because the first part of flash is reserved for the bootloader. Note that this is already done by the bootloader before starting this program. Unfortunately, function SystemInit() overwrites this change again.

#### Returns

none.

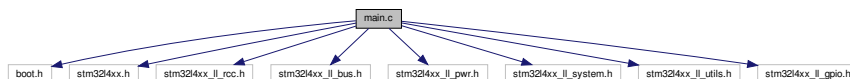
Referenced by Init().

## 7.1052 main.c File Reference

Bootloader application source file.

```
#include "boot.h"
#include "stm32l4xx.h"
#include "stm32l4xx_ll_rcc.h"
#include "stm32l4xx_ll_bus.h"
#include "stm32l4xx_ll_pwr.h"
#include "stm32l4xx_ll_system.h"
#include "stm32l4xx_ll_utils.h"
#include "stm32l4xx_ll_gpio.h"
```

Include dependency graph for ARMC4\_STM32L4\_Nucleo\_L476RG\_Keil/Boot/main.c:



## Functions

- static void [Init](#) (void)  
*Initializes the microcontroller.*
- static void [SystemClock\\_Config](#) (void)  
*System Clock Configuration. This code was created by CubeMX and configures the system clock to match the configuration in the bootloader's configuration (blt\_conf.h), specifically the macros: BOOT\_CPU\_SYSTEM\_SPEED\_KHZ and BOOT\_CPU\_XTAL\_SPEED\_KHZ. Note that the Lower Layer drivers were selected in CubeMX for the RCC subsystem.*
- int [main](#) (void)  
*This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.*
- void [HAL\\_MspInit](#) (void)  
*Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).*
- void [HAL\\_MspDeInit](#) (void)  
*DeInitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level de-initialization (GPIOs, clock, DMA, interrupt).*

## 7.1052.1 Detailed Description

Bootloader application source file.

## 7.1052.2 Function Documentation

### 7.1052.2.1 HAL\_MspDeInit()

```
void HAL_MspDeInit (
 void)
```

DeInitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level de-initialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

### 7.1052.2.2 HAL\_MspInit()

```
void HAL_MspInit (
 void)
```

Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

### 7.1052.2.3 Init()

```
static void Init (
 void) [static]
```

Initializes the microcontroller.

#### Returns

none.

Referenced by main().

#### 7.1052.2.4 main()

```
int main (
 void)
```

This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.

##### Returns

Program return code.

#### 7.1052.2.5 SystemClock\_Config()

```
static void SystemClock_Config (
 void) [static]
```

System Clock Configuration. This code was created by CubeMX and configures the system clock to match the configuration in the bootloader's configuration (blt\_conf.h), specifically the macros: BOOT\_CPU\_SYSTEM\_SPEED\_KHZ and BOOT\_CPU\_XTAL\_SPEED\_KHZ. Note that the Lower Layer drivers were selected in CubeMX for the RCC subsystem.

##### Returns

none.

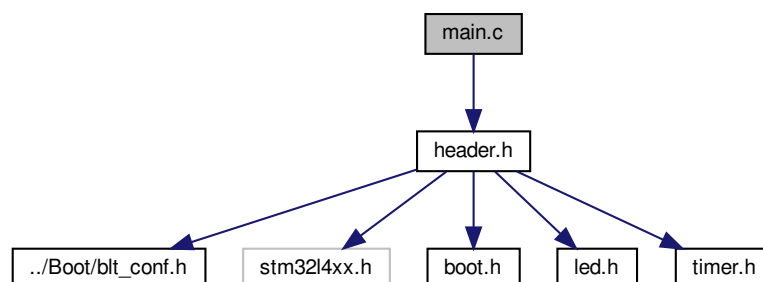
Referenced by Init().

## 7.1053 main.c File Reference

Demo program application source file.

```
#include "header.h"
```

Include dependency graph for ARMCM4\_STM32L4\_Nucleo\_L476RG\_Keil/Prog/main.c:



## Functions

- static void [Init](#) (void)  
*Initializes the microcontroller.*
- static void [SystemClock\\_Config](#) (void)  
*System Clock Configuration. This code was created by CubeMX and configures the system clock.*
- static void [VectorBase\\_Config](#) (void)  
*Vector base address configuration. It should no longer be at the start of flash memory but moved forward because the first part of flash is reserved for the bootloader. Note that this is already done by the bootloader before starting this program. Unfortunately, function SystemInit() overwrites this change again.*
- int [main](#) (void)  
*This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.*
- void [HAL\\_MspInit](#) (void)  
*Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).*
- void [HAL\\_MspDeInit](#) (void)  
*Deinitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level Deinitialization (GPIOs, clock, DMA, interrupt).*

### 7.1053.1 Detailed Description

Demo program application source file.

### 7.1053.2 Function Documentation

#### 7.1053.2.1 HAL\_MspDeInit()

```
void HAL_MspDeInit (
 void)
```

Deinitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level Deinitialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

#### 7.1053.2.2 HAL\_MspInit()

```
void HAL_MspInit (
 void)
```

Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

### 7.1053.2.3 Init()

```
static void Init (
 void) [static]
```

Initializes the microcontroller.

#### Returns

none.

Referenced by main().

### 7.1053.2.4 main()

```
int main (
 void)
```

This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.

#### Returns

Program exit code.

### 7.1053.2.5 SystemClock\_Config()

```
static void SystemClock_Config (
 void) [static]
```

System Clock Configuration. This code was created by CubeMX and configures the system clock.

#### Returns

none.

Referenced by Init().



## 7.1053.2.6 VectorBase\_Config()

```
static void VectorBase_Config (
 void) [static]
```

Vector base address configuration. It should no longer be at the start of flash memory but moved forward because the first part of flash is reserved for the bootloader. Note that this is already done by the bootloader before starting this program. Unfortunately, function SystemInit() overwrites this change again.

## Returns

none.

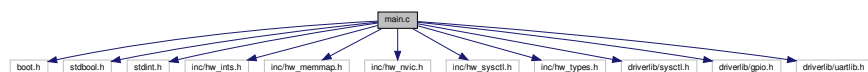
Referenced by Init().

## 7.1054 main.c File Reference

Bootloader application source file.

```
#include "boot.h"
#include <stdbool.h>
#include <stdint.h>
#include "inc/hw_ints.h"
#include "inc/hw_memmap.h"
#include "inc/hw_nvic.h"
#include "inc/hw_sysctl.h"
#include "inc/hw_types.h"
#include "driverlib/sysctl.h"
#include "driverlib/gpio.h"
#include "driverlib/uartlib.h"
```

Include dependency graph for ARMCM4\_TM4C\_DK\_TM4C123G\_IAR/Boot/main.c:



## Functions

- static void **Init** (void)  
*Initializes the microcontroller.*
- void **main** (void)  
*This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.*

## 7.1054.1 Detailed Description

Bootloader application source file.

## 7.1054.2 Function Documentation

### 7.1054.2.1 Init()

```
static void Init (
 void) [static]
```

Initializes the microcontroller.

#### Returns

none.

Referenced by main().

### 7.1054.2.2 main()

```
void main (
 void)
```

This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.

#### Returns

none.

## 7.1055 main.c File Reference

Demo program application source file.

```
#include "header.h"
```

Include dependency graph for ARMC4\_TM4C\_DK\_TM4C123G\_IAR/Prog/main.c:



## Functions

- static void [Init](#) (void)  
*Initializes the microcontroller.*
- void [main](#) (void)  
*This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.*

### 7.1055.1 Detailed Description

Demo program application source file.

### 7.1055.2 Function Documentation

#### 7.1055.2.1 Init()

```
static void Init (
 void) [static]
```

Initializes the microcontroller.

##### Returns

none.

Referenced by main().

#### 7.1055.2.2 main()

```
void main (
 void)
```

This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.

##### Returns

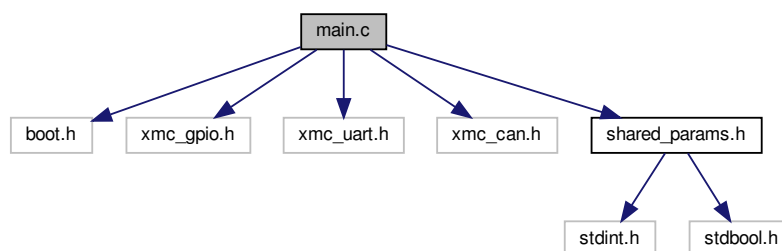
none.

## 7.1056 main.c File Reference

Bootloader application source file.

```
#include "boot.h"
#include "xmc_gpio.h"
#include "xmc_uart.h"
#include "xmc_can.h"
#include "shared_params.h"
```

Include dependency graph for ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_GCC/Boot/main.c:



## Functions

- static void `Init` (void)  
*Initializes the microcontroller.*
- static void `PostInit` (void)  
*Post initialization of the microcontroller. Contains all initialization code that should run after the bootloader's core was initialized.*
- int `main` (void)  
*This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.*

### 7.1056.1 Detailed Description

Bootloader application source file.

### 7.1056.2 Function Documentation

#### 7.1056.2.1 `Init()`

```
static void Init (
 void) [static]
```

Initializes the microcontroller.

#### Returns

none.

Referenced by `main()`.

#### 7.1056.2.2 `main()`

```
int main (
 void)
```

This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.

#### Returns

Program return code.

## 7.1056.2.3 PostInit()

```
static void PostInit (
 void) [static]
```

Post initialization of the microcontroller. Contains all initialization code that should run after the bootloader's core was initialized.

## Returns

none.

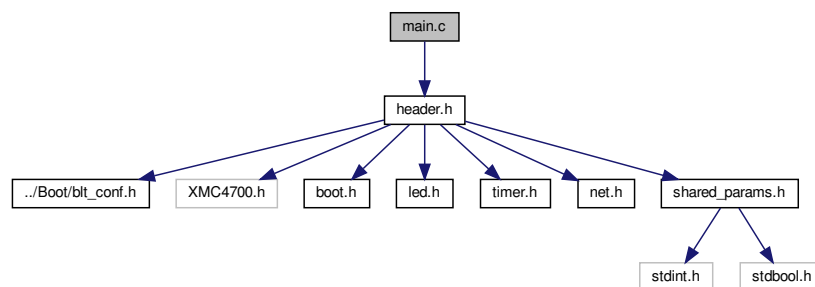
Referenced by main().

## 7.1057 main.c File Reference

Demo program application source file.

```
#include "header.h"
```

Include dependency graph for ARMC4\_XMC4700\_Relax\_Kit\_GCC/Prog/main.c:



## Functions

- static void [Init](#) (void)  
*Initializes the microcontroller.*
- int [main](#) (void)  
*This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.*

## 7.1057.1 Detailed Description

Demo program application source file.

## 7.1057.2 Function Documentation

### 7.1057.2.1 Init()

```
static void Init (
 void) [static]
```

Initializes the microcontroller.

#### Returns

none.

Referenced by main().

### 7.1057.2.2 main()

```
int main (
 void)
```

This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.

#### Returns

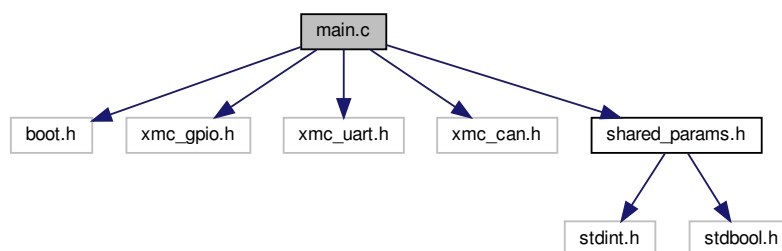
none.

## 7.1058 main.c File Reference

Bootloader application source file.

```
#include "boot.h"
#include "xmc_gpio.h"
#include "xmc_uart.h"
#include "xmc_can.h"
#include "shared_params.h"
```

Include dependency graph for ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_IAR/Boot/main.c:



## Functions

- static void `Init` (void)  
*Initializes the microcontroller.*
- static void `PostInit` (void)  
*Post initialization of the microcontroller. Contains all initialization code that should run after the bootloader's core was initialized.*
- void `main` (void)  
*This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.*

### 7.1058.1 Detailed Description

Bootloader application source file.

### 7.1058.2 Function Documentation

#### 7.1058.2.1 `Init()`

```
static void Init (
 void) [static]
```

Initializes the microcontroller.

#### Returns

none.

Referenced by `main()`.

#### 7.1058.2.2 `main()`

```
void main (
 void)
```

This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.

#### Returns

None.

### 7.1058.2.3 PostInit()

```
static void PostInit (
 void) [static]
```

Post initialization of the microcontroller. Contains all initialization code that should run after the bootloader's core was initialized.

#### Returns

none.

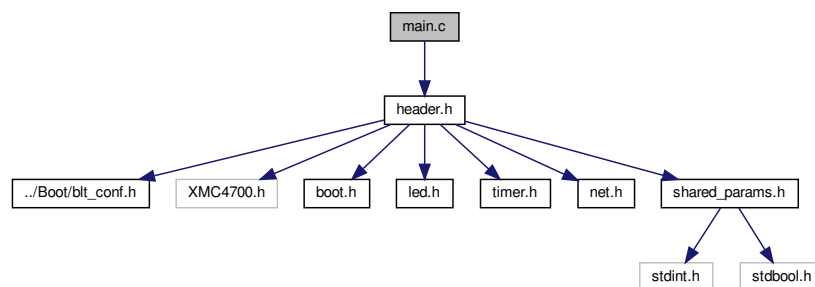
Referenced by main().

## 7.1059 main.c File Reference

Demo program application source file.

```
#include "header.h"
```

Include dependency graph for ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_IAR/Prog/main.c:



## Functions

- static void `Init` (void)  
*Initializes the microcontroller.*
- void `main` (void)  
*This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.*

### 7.1059.1 Detailed Description

Demo program application source file.

### 7.1059.2 Function Documentation



### 7.1059.2.1 Init()

```
static void Init (
 void) [static]
```

Initializes the microcontroller.

#### Returns

none.

Referenced by main().

### 7.1059.2.2 main()

```
void main (
 void)
```

This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.

#### Returns

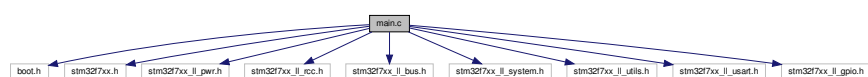
none.

## 7.1060 main.c File Reference

Bootloader application source file.

```
#include "boot.h"
#include "stm32f7xx.h"
#include "stm32f7xx_ll_pwr.h"
#include "stm32f7xx_ll_rcc.h"
#include "stm32f7xx_ll_bus.h"
#include "stm32f7xx_ll_system.h"
#include "stm32f7xx_ll_utils.h"
#include "stm32f7xx_ll_usart.h"
#include "stm32f7xx_ll_gpio.h"
```

Include dependency graph for ARMCM7\_STM32F7\_Nucleo\_F746ZG\_GCC/Boot/main.c:



## Functions

- static void `Init` (void)  
*Initializes the microcontroller.*
- static void `SystemClock_Config` (void)  
*System Clock Configuration. This code was created by CubeMX and configures the system clock to match the configuration in the bootloader's configuration (blt\_conf.h), specifically the macros: BOOT\_CPU\_SYSTEM\_SPEED\_KHZ and BOOT\_CPU\_XTAL\_SPEED\_KHZ. Note that the Lower Layer drivers were selected in CubeMX for the RCC subsystem.*
- int `main` (void)  
*This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.*
- void `HAL_MspInit` (void)  
*Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).*
- void `HAL_MspDeInit` (void)  
*DeInitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level de-initialization (GPIOs, clock, DMA, interrupt).*

### 7.1060.1 Detailed Description

Bootloader application source file.

### 7.1060.2 Function Documentation

#### 7.1060.2.1 HAL\_MspDeInit()

```
void HAL_MspDeInit (
 void)
```

DeInitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level de-initialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

#### 7.1060.2.2 HAL\_MspInit()

```
void HAL_MspInit (
 void)
```

Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

### 7.1060.2.3 Init()

```
static void Init (
 void) [static]
```

Initializes the microcontroller.

#### Returns

none.

Referenced by main().

### 7.1060.2.4 main()

```
int main (
 void)
```

This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.

#### Returns

Program return code.

### 7.1060.2.5 SystemClock\_Config()

```
static void SystemClock_Config (
 void) [static]
```

System Clock Configuration. This code was created by CubeMX and configures the system clock to match the configuration in the bootloader's configuration (blt\_conf.h), specifically the macros: BOOT\_CPU\_SYSTEM\_SPEED\_KHZ and BOOT\_CPU\_XTAL\_SPEED\_KHZ. Note that the Lower Layer drivers were selected in CubeMX for the RCC subsystem.

#### Returns

none.

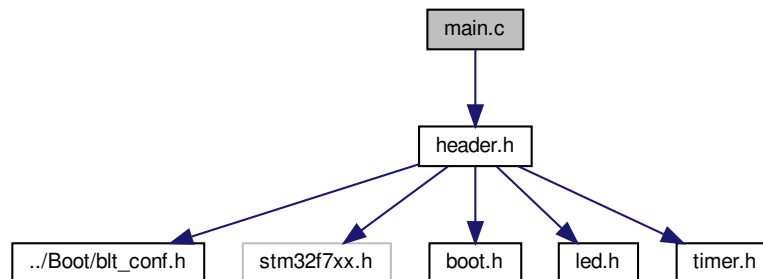
Referenced by Init().

## 7.1061 main.c File Reference

Demo program application source file.

```
#include "header.h"
```

Include dependency graph for ARMC7\_STM32F7\_Nucleo\_F746ZG\_GCC/Prog/main.c:



### Functions

- static void [Init](#) (void)  
*Initializes the microcontroller.*
- static void [SystemClock\\_Config](#) (void)  
*System Clock Configuration. This code was created by CubeMX and configures the system clock.*
- static void [VectorBase\\_Config](#) (void)  
*Vector base address configuration. It should no longer be at the start of flash memory but moved forward because the first part of flash is reserved for the bootloader. Note that this is already done by the bootloader before starting this program. Unfortunately, function SystemInit() overwrites this change again.*
- int [main](#) (void)  
*This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.*
- void [HAL\\_MspInit](#) (void)  
*Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).*
- void [HAL\\_MspDeInit](#) (void)  
*Deinitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level Deinitialization (GPIOs, clock, DMA, interrupt).*

### 7.1061.1 Detailed Description

Demo program application source file.

### 7.1061.2 Function Documentation

### 7.1061.2.1 HAL\_MspDeInit()

```
void HAL_MspDeInit (
 void)
```

Deinitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level Deinitialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

### 7.1061.2.2 HAL\_MspInit()

```
void HAL_MspInit (
 void)
```

Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

### 7.1061.2.3 Init()

```
static void Init (
 void) [static]
```

Initializes the microcontroller.

#### Returns

none.

Referenced by main().

### 7.1061.2.4 main()

```
int main (
 void)
```

This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.

#### Returns

Program exit code.

### 7.1061.2.5 SystemClock\_Config()

```
static void SystemClock_Config (
 void) [static]
```

System Clock Configuration. This code was created by CubeMX and configures the system clock.

#### Returns

none.

Referenced by Init().

### 7.1061.2.6 VectorBase\_Config()

```
static void VectorBase_Config (
 void) [static]
```

Vector base address configuration. It should no longer be at the start of flash memory but moved forward because the first part of flash is reserved for the bootloader. Note that this is already done by the bootloader before starting this program. Unfortunately, function SystemInit() overwrites this change again.

#### Returns

none.

Referenced by Init().

## 7.1062 main.c File Reference

Bootloader application source file.

```
#include "boot.h"
#include "stm32f7xx.h"
#include "stm32f7xx_ll_pwr.h"
#include "stm32f7xx_ll_rcc.h"
#include "stm32f7xx_ll_bus.h"
#include "stm32f7xx_ll_system.h"
#include "stm32f7xx_ll_utils.h"
#include "stm32f7xx_ll_usart.h"
#include "stm32f7xx_ll_gpio.h"
```

Include dependency graph for ARMCM7\_STM32F7\_Nucleo\_F746ZG\_IAR/Boot/main.c:



## Functions

- static void `Init` (void)  
*Initializes the microcontroller.*
- static void `SystemClock_Config` (void)  
*System Clock Configuration. This code was created by CubeMX and configures the system clock to match the configuration in the bootloader's configuration (blt\_conf.h), specifically the macros: BOOT\_CPU\_SYSTEM\_SPEED\_KHZ and BOOT\_CPU\_XTAL\_SPEED\_KHZ. Note that the Lower Layer drivers were selected in CubeMX for the RCC subsystem.*
- void `main` (void)  
*This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.*
- void `HAL_MspInit` (void)  
*Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).*
- void `HAL_MspDeInit` (void)  
*DeInitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level de-initialization (GPIOs, clock, DMA, interrupt).*

### 7.1062.1 Detailed Description

Bootloader application source file.

### 7.1062.2 Function Documentation

#### 7.1062.2.1 HAL\_MspDeInit()

```
void HAL_MspDeInit (
 void)
```

DeInitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level de-initialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

#### 7.1062.2.2 HAL\_MspInit()

```
void HAL_MspInit (
 void)
```

Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

### 7.1062.2.3 Init()

```
static void Init (
 void) [static]
```

Initializes the microcontroller.

#### Returns

none.

Referenced by main().

### 7.1062.2.4 main()

```
void main (
 void)
```

This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.

#### Returns

none.

### 7.1062.2.5 SystemClock\_Config()

```
static void SystemClock_Config (
 void) [static]
```

System Clock Configuration. This code was created by CubeMX and configures the system clock to match the configuration in the bootloader's configuration (blt\_conf.h), specifically the macros: BOOT\_CPU\_SYSTEM\_SPEED\_KHZ and BOOT\_CPU\_XTAL\_SPEED\_KHZ. Note that the Lower Layer drivers were selected in CubeMX for the RCC subsystem.

#### Returns

none.

Referenced by Init().

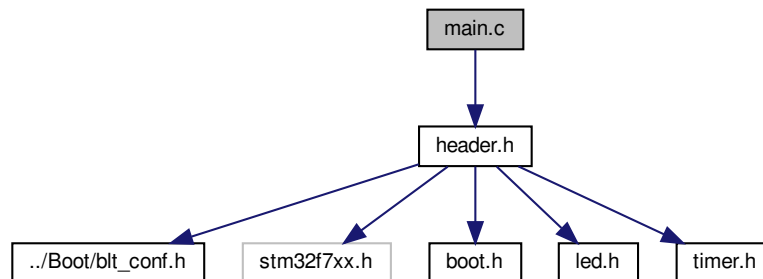


## 7.1063 main.c File Reference

Demo program application source file.

```
#include "header.h"
```

Include dependency graph for ARMCM7\_STM32F7\_Nucleo\_F746ZG\_IAR/Prog/main.c:



### Functions

- static void [Init](#) (void)  
*Initializes the microcontroller.*
- static void [SystemClock\\_Config](#) (void)  
*System Clock Configuration. This code was created by CubeMX and configures the system clock.*
- static void [VectorBase\\_Config](#) (void)  
*Vector base address configuration. It should no longer be at the start of flash memory but moved forward because the first part of flash is reserved for the bootloader. Note that this is already done by the bootloader before starting this program. Unfortunately, function SystemInit() overwrites this change again.*
- void [main](#) (void)  
*This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.*
- void [HAL\\_MspInit](#) (void)  
*Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).*
- void [HAL\\_MspDeInit](#) (void)  
*Deinitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level Deinitialization (GPIOs, clock, DMA, interrupt).*

### 7.1063.1 Detailed Description

Demo program application source file.

### 7.1063.2 Function Documentation

### 7.1063.2.1 HAL\_MspDeInit()

```
void HAL_MspDeInit (
 void)
```

Deinitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level Deinitialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

### 7.1063.2.2 HAL\_MspInit()

```
void HAL_MspInit (
 void)
```

Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

### 7.1063.2.3 Init()

```
static void Init (
 void) [static]
```

Initializes the microcontroller.

#### Returns

none.

Referenced by main().

### 7.1063.2.4 main()

```
void main (
 void)
```

This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.

#### Returns

none.

## 7.1063.2.5 SystemClock\_Config()

```
static void SystemClock_Config (
 void) [static]
```

System Clock Configuration. This code was created by CubeMX and configures the system clock.

## Returns

none.

Referenced by Init().

## 7.1063.2.6 VectorBase\_Config()

```
static void VectorBase_Config (
 void) [static]
```

Vector base address configuration. It should no longer be at the start of flash memory but moved forward because the first part of flash is reserved for the bootloader. Note that this is already done by the bootloader before starting this program. Unfortunately, function SystemInit() overwrites this change again.

## Returns

none.

Referenced by Init().

## 7.1064 main.c File Reference

Bootloader application source file.

```
#include "boot.h"
#include "stm32f7xx.h"
#include "stm32f7xx_ll_pwr.h"
#include "stm32f7xx_ll_rcc.h"
#include "stm32f7xx_ll_bus.h"
#include "stm32f7xx_ll_system.h"
#include "stm32f7xx_ll_utils.h"
#include "stm32f7xx_ll_usart.h"
#include "stm32f7xx_ll_gpio.h"
```

Include dependency graph for ARMCM7\_STM32F7\_Nucleo\_F746ZG\_Keil/Boot/main.c:



## Functions

- static void `Init` (void)  
*Initializes the microcontroller.*
- static void `SystemClock_Config` (void)  
*System Clock Configuration. This code was created by CubeMX and configures the system clock to match the configuration in the bootloader's configuration (blt\_conf.h), specifically the macros: BOOT\_CPU\_SYSTEM\_SPEED\_KHZ and BOOT\_CPU\_XTAL\_SPEED\_KHZ. Note that the Lower Layer drivers were selected in CubeMX for the RCC subsystem.*
- int `main` (void)  
*This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.*
- void `HAL_MspInit` (void)  
*Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).*
- void `HAL_MspDeInit` (void)  
*DeInitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level de-initialization (GPIOs, clock, DMA, interrupt).*

### 7.1064.1 Detailed Description

Bootloader application source file.

### 7.1064.2 Function Documentation

#### 7.1064.2.1 HAL\_MspDeInit()

```
void HAL_MspDeInit (
 void)
```

DeInitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level de-initialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

#### 7.1064.2.2 HAL\_MspInit()

```
void HAL_MspInit (
 void)
```

Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

### 7.1064.2.3 Init()

```
static void Init (
 void) [static]
```

Initializes the microcontroller.

#### Returns

none.

Referenced by main().

### 7.1064.2.4 main()

```
int main (
 void)
```

This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.

#### Returns

Program exit code.

### 7.1064.2.5 SystemClock\_Config()

```
static void SystemClock_Config (
 void) [static]
```

System Clock Configuration. This code was created by CubeMX and configures the system clock to match the configuration in the bootloader's configuration (blt\_conf.h), specifically the macros: BOOT\_CPU\_SYSTEM\_SPEED\_KHZ and BOOT\_CPU\_XTAL\_SPEED\_KHZ. Note that the Lower Layer drivers were selected in CubeMX for the RCC subsystem.

#### Returns

none.

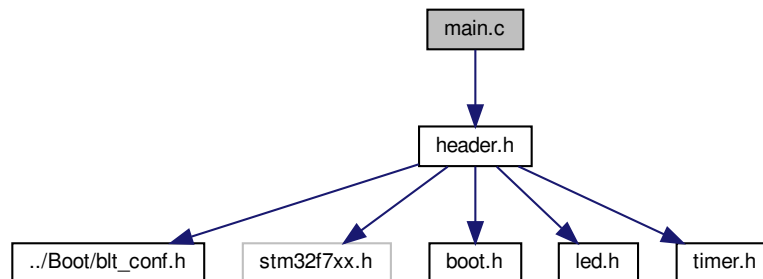
Referenced by Init().

## 7.1065 main.c File Reference

Demo program application source file.

```
#include "header.h"
```

Include dependency graph for ARMC7\_STM32F7\_Nucleo\_F746ZG\_Keil/Prog/main.c:



### Functions

- static void [Init](#) (void)  
*Initializes the microcontroller.*
- static void [SystemClock\\_Config](#) (void)  
*System Clock Configuration. This code was created by CubeMX and configures the system clock.*
- static void [VectorBase\\_Config](#) (void)  
*Vector base address configuration. It should no longer be at the start of flash memory but moved forward because the first part of flash is reserved for the bootloader. Note that this is already done by the bootloader before starting this program. Unfortunately, function SystemInit() overwrites this change again.*
- int [main](#) (void)  
*This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.*
- void [HAL\\_MspInit](#) (void)  
*Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).*
- void [HAL\\_MspDeInit](#) (void)  
*Deinitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level Deinitialization (GPIOs, clock, DMA, interrupt).*

### 7.1065.1 Detailed Description

Demo program application source file.

### 7.1065.2 Function Documentation

### 7.1065.2.1 HAL\_MspDeInit()

```
void HAL_MspDeInit (
 void)
```

Deinitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level Deinitialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

### 7.1065.2.2 HAL\_MspInit()

```
void HAL_MspInit (
 void)
```

Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

### 7.1065.2.3 Init()

```
static void Init (
 void) [static]
```

Initializes the microcontroller.

#### Returns

none.

Referenced by main().

### 7.1065.2.4 main()

```
int main (
 void)
```

This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.

#### Returns

Program exit code.

### 7.1065.2.5 SystemClock\_Config()

```
static void SystemClock_Config (
 void) [static]
```

System Clock Configuration. This code was created by CubeMX and configures the system clock.

#### Returns

none.

Referenced by Init().

### 7.1065.2.6 VectorBase\_Config()

```
static void VectorBase_Config (
 void) [static]
```

Vector base address configuration. It should no longer be at the start of flash memory but moved forward because the first part of flash is reserved for the bootloader. Note that this is already done by the bootloader before starting this program. Unfortunately, function SystemInit() overwrites this change again.

#### Returns

none.

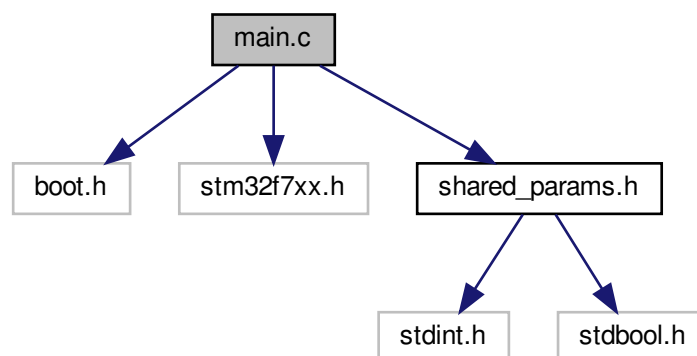
Referenced by Init().

## 7.1066 main.c File Reference

Bootloader application source file.

```
#include "boot.h"
#include "stm32f7xx.h"
#include "shared_params.h"
```

Include dependency graph for ARMC7\_STM32F7\_Nucleo\_F767ZI\_GCC/Boot/main.c:





## Functions

- static void `Init` (void)  
*Initializes the microcontroller.*
- static void `SystemClock_Config` (void)  
*System Clock Configuration. This code was created by CubeMX and configures the system clock to match the configuration in the bootloader's configuration (blt\_conf.h), specifically the macros: BOOT\_CPU\_SYSTEM\_SPEED\_KHZ and BOOT\_CPU\_XTAL\_SPEED\_KHZ. Note that the Lower Layer drivers were selected in CubeMX for the RCC subsystem.*
- int `main` (void)  
*This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.*
- void `HAL_MspInit` (void)  
*Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).*
- void `HAL_MspDeInit` (void)  
*DeInitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level de-initialization (GPIOs, clock, DMA, interrupt).*

### 7.1066.1 Detailed Description

Bootloader application source file.

### 7.1066.2 Function Documentation

#### 7.1066.2.1 HAL\_MspDeInit()

```
void HAL_MspDeInit (
 void)
```

DeInitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level de-initialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

#### 7.1066.2.2 HAL\_MspInit()

```
void HAL_MspInit (
 void)
```

Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

### 7.1066.2.3 Init()

```
static void Init (
 void) [static]
```

Initializes the microcontroller.

#### Returns

none.

Referenced by main().

### 7.1066.2.4 main()

```
int main (
 void)
```

This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.

#### Returns

Program return code.

### 7.1066.2.5 SystemClock\_Config()

```
static void SystemClock_Config (
 void) [static]
```

System Clock Configuration. This code was created by CubeMX and configures the system clock to match the configuration in the bootloader's configuration (blt\_conf.h), specifically the macros: BOOT\_CPU\_SYSTEM\_SPEED\_KHZ and BOOT\_CPU\_XTAL\_SPEED\_KHZ. Note that the Lower Layer drivers were selected in CubeMX for the RCC subsystem.

#### Returns

none.

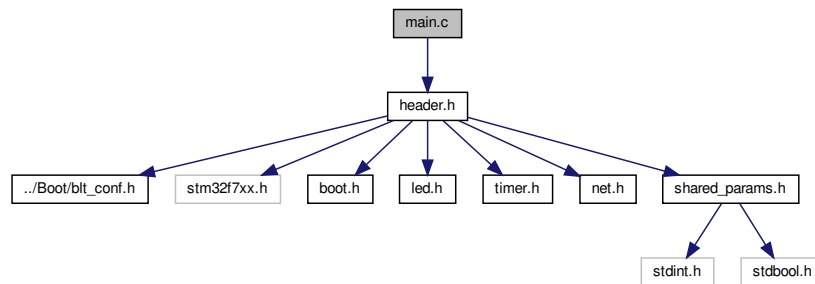
Referenced by Init().

## 7.1067 main.c File Reference

Demo program application source file.

```
#include "header.h"
```

Include dependency graph for ARMCM7\_STM32F7\_Nucleo\_F767ZI\_GCC/Prog/main.c:



### Functions

- static void [Init](#) (void)  
*Initializes the microcontroller.*
- static void [SystemClock\\_Config](#) (void)  
*System Clock Configuration. This code was created by CubeMX and configures the system clock.*
- static void [VectorBase\\_Config](#) (void)  
*Vector base address configuration. It should no longer be at the start of flash memory but moved forward because the first part of flash is reserved for the bootloader. Note that this is already done by the bootloader before starting this program. Unfortunately, function SystemInit() overwrites this change again.*
- int [main](#) (void)  
*This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.*
- void [HAL\\_MspInit](#) (void)  
*Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).*
- void [HAL\\_MspDeInit](#) (void)  
*Deinitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level Deinitialization (GPIOs, clock, DMA, interrupt).*

### 7.1067.1 Detailed Description

Demo program application source file.

### 7.1067.2 Function Documentation

### 7.1067.2.1 HAL\_MspDeInit()

```
void HAL_MspDeInit (
 void)
```

Deinitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level Deinitialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

### 7.1067.2.2 HAL\_MspInit()

```
void HAL_MspInit (
 void)
```

Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

### 7.1067.2.3 Init()

```
static void Init (
 void) [static]
```

Initializes the microcontroller.

#### Returns

none.

Referenced by main().

### 7.1067.2.4 main()

```
int main (
 void)
```

This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.

#### Returns

Program exit code.

### 7.1067.2.5 SystemClock\_Config()

```
static void SystemClock_Config (
 void) [static]
```

System Clock Configuration. This code was created by CubeMX and configures the system clock.

#### Returns

none.

Referenced by Init().

### 7.1067.2.6 VectorBase\_Config()

```
static void VectorBase_Config (
 void) [static]
```

Vector base address configuration. It should no longer be at the start of flash memory but moved forward because the first part of flash is reserved for the bootloader. Note that this is already done by the bootloader before starting this program. Unfortunately, function SystemInit() overwrites this change again.

#### Returns

none.

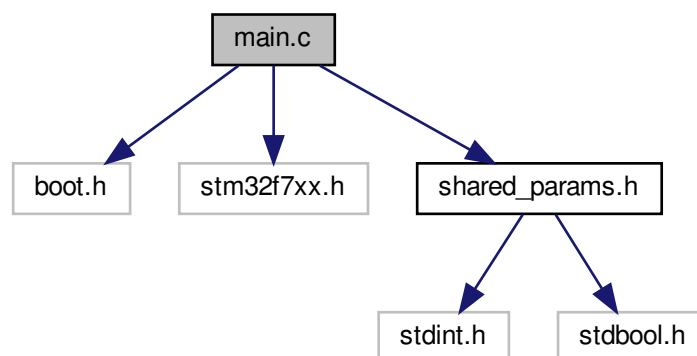
Referenced by Init().

## 7.1068 main.c File Reference

Bootloader application source file.

```
#include "boot.h"
#include "stm32f7xx.h"
#include "shared_params.h"
```

Include dependency graph for ARMC7\_STM32F7\_Nucleo\_F767ZI\_IAR/Boot/main.c:



## Functions

- static void `Init` (void)  
*Initializes the microcontroller.*
- static void `SystemClock_Config` (void)  
*System Clock Configuration. This code was created by CubeMX and configures the system clock to match the configuration in the bootloader's configuration (blt\_conf.h), specifically the macros: BOOT\_CPU\_SYSTEM\_SPEED\_KHZ and BOOT\_CPU\_XTAL\_SPEED\_KHZ. Note that the Lower Layer drivers were selected in CubeMX for the RCC subsystem.*
- void `main` (void)  
*This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.*
- void `HAL_MspInit` (void)  
*Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).*
- void `HAL_MspDeInit` (void)  
*DeInitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level de-initialization (GPIOs, clock, DMA, interrupt).*

### 7.1068.1 Detailed Description

Bootloader application source file.

### 7.1068.2 Function Documentation

#### 7.1068.2.1 HAL\_MspDeInit()

```
void HAL_MspDeInit (
 void)
```

DeInitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level de-initialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

#### 7.1068.2.2 HAL\_MspInit()

```
void HAL_MspInit (
 void)
```

Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

### 7.1068.2.3 Init()

```
static void Init (
 void) [static]
```

Initializes the microcontroller.

#### Returns

none.

Referenced by main().

### 7.1068.2.4 main()

```
void main (
 void)
```

This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.

#### Returns

None.

### 7.1068.2.5 SystemClock\_Config()

```
static void SystemClock_Config (
 void) [static]
```

System Clock Configuration. This code was created by CubeMX and configures the system clock to match the configuration in the bootloader's configuration (blt\_conf.h), specifically the macros: BOOT\_CPU\_SYSTEM\_SPEED\_KHZ and BOOT\_CPU\_XTAL\_SPEED\_KHZ. Note that the Lower Layer drivers were selected in CubeMX for the RCC subsystem.

#### Returns

none.

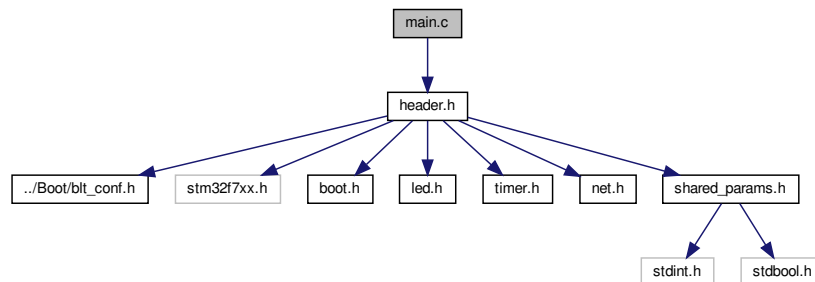
Referenced by Init().

## 7.1069 main.c File Reference

Demo program application source file.

```
#include "header.h"
```

Include dependency graph for ARMCM7\_STM32F7\_Nucleo\_F767ZI\_IAR/Prog/main.c:



## Functions

- static void [Init](#) (void)  
*Initializes the microcontroller.*
- static void [SystemClock\\_Config](#) (void)  
*System Clock Configuration. This code was created by CubeMX and configures the system clock.*
- static void [VectorBase\\_Config](#) (void)  
*Vector base address configuration. It should no longer be at the start of flash memory but moved forward because the first part of flash is reserved for the bootloader. Note that this is already done by the bootloader before starting this program. Unfortunately, function SystemInit() overwrites this change again.*
- void [main](#) (void)  
*This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.*
- void [HAL\\_MspInit](#) (void)  
*Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).*
- void [HAL\\_MspDeInit](#) (void)  
*Deinitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level Deinitialization (GPIOs, clock, DMA, interrupt).*

### 7.1069.1 Detailed Description

Demo program application source file.

### 7.1069.2 Function Documentation



### 7.1069.2.1 HAL\_MspDeInit()

```
void HAL_MspDeInit (
 void)
```

Deinitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level Deinitialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

### 7.1069.2.2 HAL\_MspInit()

```
void HAL_MspInit (
 void)
```

Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

### 7.1069.2.3 Init()

```
static void Init (
 void) [static]
```

Initializes the microcontroller.

#### Returns

none.

Referenced by main().

### 7.1069.2.4 main()

```
void main (
 void)
```

This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.

#### Returns

None.

### 7.1069.2.5 SystemClock\_Config()

```
static void SystemClock_Config (
 void) [static]
```

System Clock Configuration. This code was created by CubeMX and configures the system clock.

#### Returns

none.

Referenced by Init().

### 7.1069.2.6 VectorBase\_Config()

```
static void VectorBase_Config (
 void) [static]
```

Vector base address configuration. It should no longer be at the start of flash memory but moved forward because the first part of flash is reserved for the bootloader. Note that this is already done by the bootloader before starting this program. Unfortunately, function SystemInit() overwrites this change again.

#### Returns

none.

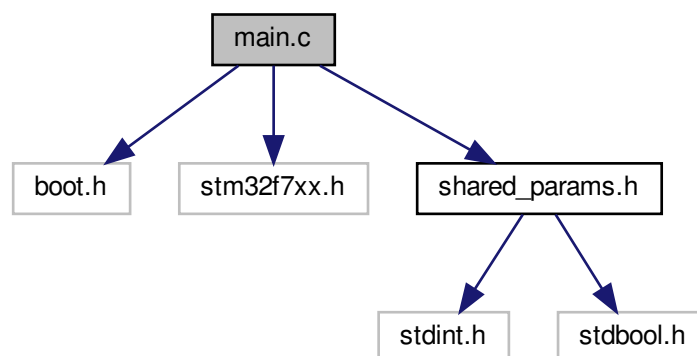
Referenced by Init().

## 7.1070 main.c File Reference

Bootloader application source file.

```
#include "boot.h"
#include "stm32f7xx.h"
#include "shared_params.h"
```

Include dependency graph for ARMC7\_STM32F7\_Nucleo\_F767ZI\_Keil/Boot/main.c:



## Functions

- static void `Init` (void)  
*Initializes the microcontroller.*
- static void `SystemClock_Config` (void)  
*System Clock Configuration. This code was created by CubeMX and configures the system clock to match the configuration in the bootloader's configuration (blt\_conf.h), specifically the macros: BOOT\_CPU\_SYSTEM\_SPEED\_KHZ and BOOT\_CPU\_XTAL\_SPEED\_KHZ. Note that the Lower Layer drivers were selected in CubeMX for the RCC subsystem.*
- int `main` (void)  
*This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.*
- void `HAL_MspInit` (void)  
*Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).*
- void `HAL_MspDeInit` (void)  
*DeInitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level de-initialization (GPIOs, clock, DMA, interrupt).*

### 7.1070.1 Detailed Description

Bootloader application source file.

### 7.1070.2 Function Documentation

#### 7.1070.2.1 HAL\_MspDeInit()

```
void HAL_MspDeInit (
 void)
```

DeInitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level de-initialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

#### 7.1070.2.2 HAL\_MspInit()

```
void HAL_MspInit (
 void)
```

Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

### 7.1070.2.3 Init()

```
static void Init (
 void) [static]
```

Initializes the microcontroller.

#### Returns

none.

Referenced by main().

### 7.1070.2.4 main()

```
int main (
 void)
```

This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.

#### Returns

Program return code.

### 7.1070.2.5 SystemClock\_Config()

```
static void SystemClock_Config (
 void) [static]
```

System Clock Configuration. This code was created by CubeMX and configures the system clock to match the configuration in the bootloader's configuration (blt\_conf.h), specifically the macros: BOOT\_CPU\_SYSTEM\_SPEED\_KHZ and BOOT\_CPU\_XTAL\_SPEED\_KHZ. Note that the Lower Layer drivers were selected in CubeMX for the RCC subsystem.

#### Returns

none.

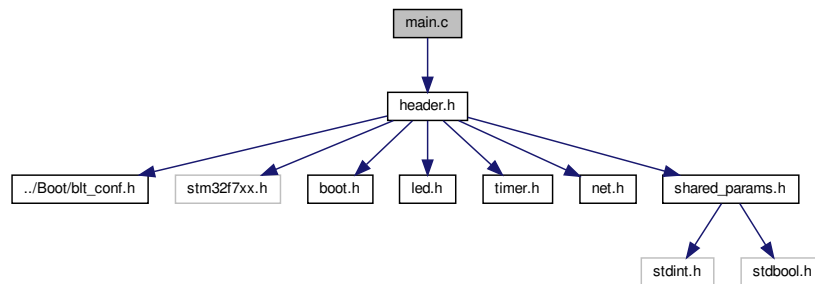
Referenced by Init().

## 7.1071 main.c File Reference

Demo program application source file.

```
#include "header.h"
```

Include dependency graph for ARMC7\_STM32F7\_Nucleo\_F767ZI\_Keil/Prog/main.c:



### Functions

- static void [Init](#) (void)  
*Initializes the microcontroller.*
- static void [SystemClock\\_Config](#) (void)  
*System Clock Configuration. This code was created by CubeMX and configures the system clock.*
- static void [VectorBase\\_Config](#) (void)  
*Vector base address configuration. It should no longer be at the start of flash memory but moved forward because the first part of flash is reserved for the bootloader. Note that this is already done by the bootloader before starting this program. Unfortunately, function SystemInit() overwrites this change again.*
- int [main](#) (void)  
*This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.*
- void [HAL\\_MspInit](#) (void)  
*Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).*
- void [HAL\\_MspDeInit](#) (void)  
*Deinitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level Deinitialization (GPIOs, clock, DMA, interrupt).*

### 7.1071.1 Detailed Description

Demo program application source file.

### 7.1071.2 Function Documentation

### 7.1071.2.1 HAL\_MspDeInit()

```
void HAL_MspDeInit (
 void)
```

Deinitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level Deinitialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

### 7.1071.2.2 HAL\_MspInit()

```
void HAL_MspInit (
 void)
```

Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

### 7.1071.2.3 Init()

```
static void Init (
 void) [static]
```

Initializes the microcontroller.

#### Returns

none.

Referenced by main().

### 7.1071.2.4 main()

```
int main (
 void)
```

This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.

#### Returns

Program exit code.

### 7.1071.2.5 SystemClock\_Config()

```
static void SystemClock_Config (
 void) [static]
```

System Clock Configuration. This code was created by CubeMX and configures the system clock.

#### Returns

none.

Referenced by Init().

### 7.1071.2.6 VectorBase\_Config()

```
static void VectorBase_Config (
 void) [static]
```

Vector base address configuration. It should no longer be at the start of flash memory but moved forward because the first part of flash is reserved for the bootloader. Note that this is already done by the bootloader before starting this program. Unfortunately, function SystemInit() overwrites this change again.

#### Returns

none.

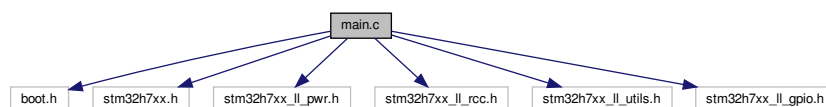
Referenced by Init().

## 7.1072 main.c File Reference

Bootloader application source file.

```
#include "boot.h"
#include "stm32h7xx.h"
#include "stm32h7xx_ll_pwr.h"
#include "stm32h7xx_ll_rcc.h"
#include "stm32h7xx_ll_utils.h"
#include "stm32h7xx_ll_gpio.h"
```

Include dependency graph for ARMC7\_STM32H7\_Nucleo\_H743ZI\_GCC/Boot/main.c:



## Functions

- static void `Init` (void)  
*Initializes the microcontroller.*
- static void `SystemClock_Config` (void)  
*System Clock Configuration. This code was created by CubeMX and configures the system clock to match the configuration in the bootloader's configuration (blt\_conf.h), specifically the macros: BOOT\_CPU\_SYSTEM\_SPEED\_KHZ and BOOT\_CPU\_XTAL\_SPEED\_KHZ. Note that the Lower Layer drivers were selected in CubeMX for the RCC subsystem.*
- int `main` (void)  
*This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.*
- void `HAL_MspInit` (void)  
*Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).*
- void `HAL_MspDeInit` (void)  
*DeInitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level de-initialization (GPIOs, clock, DMA, interrupt).*

### 7.1072.1 Detailed Description

Bootloader application source file.

### 7.1072.2 Function Documentation

#### 7.1072.2.1 HAL\_MspDeInit()

```
void HAL_MspDeInit (
 void)
```

DeInitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level de-initialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

#### 7.1072.2.2 HAL\_MspInit()

```
void HAL_MspInit (
 void)
```

Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.



### 7.1072.2.3 Init()

```
static void Init (
 void) [static]
```

Initializes the microcontroller.

#### Returns

none.

Referenced by main().

### 7.1072.2.4 main()

```
int main (
 void)
```

This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.

#### Returns

Program return code.

### 7.1072.2.5 SystemClock\_Config()

```
static void SystemClock_Config (
 void) [static]
```

System Clock Configuration. This code was created by CubeMX and configures the system clock to match the configuration in the bootloader's configuration (blt\_conf.h), specifically the macros: BOOT\_CPU\_SYSTEM\_SPEED\_KHZ and BOOT\_CPU\_XTAL\_SPEED\_KHZ. Note that the Lower Layer drivers were selected in CubeMX for the RCC subsystem.

#### Returns

none.

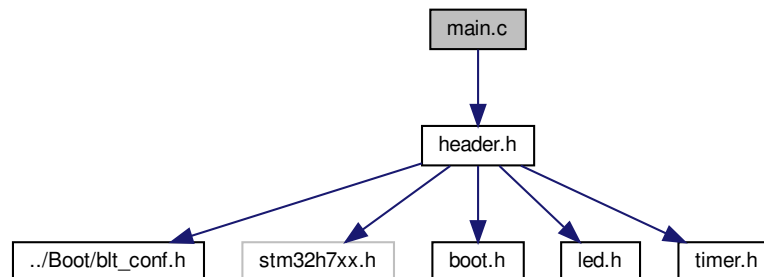
Referenced by Init().

## 7.1073 main.c File Reference

Demo program application source file.

```
#include "header.h"
```

Include dependency graph for ARMCM7\_STM32H7\_Nucleo\_H743ZI\_GCC/Prog/main.c:



### Functions

- static void `Init` (void)  
*Initializes the microcontroller.*
- static void `SystemClock_Config` (void)  
*System Clock Configuration. This code was created by CubeMX and configures the system clock.*
- static void `VectorBase_Config` (void)  
*Vector base address configuration. It should no longer be at the start of flash memory but moved forward because the first part of flash is reserved for the bootloader. Note that this is already done by the bootloader before starting this program. Unfortunately, function `SystemInit()` overwrites this change again.*
- int `main` (void)  
*This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.*
- void `HAL_MspInit` (void)  
*Initializes the Global MSP. This function is called from `HAL_Init()` function to perform system level initialization (GPIOs, clock, DMA, interrupt).*
- void `HAL_MspDeInit` (void)  
*Deinitializes the Global MSP. This function is called from `HAL_DeInit()` function to perform system level Deinitialization (GPIOs, clock, DMA, interrupt).*

### 7.1073.1 Detailed Description

Demo program application source file.

### 7.1073.2 Function Documentation

### 7.1073.2.1 HAL\_MspDeInit()

```
void HAL_MspDeInit (
 void)
```

Deinitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level Deinitialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

### 7.1073.2.2 HAL\_MspInit()

```
void HAL_MspInit (
 void)
```

Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

### 7.1073.2.3 Init()

```
static void Init (
 void) [static]
```

Initializes the microcontroller.

#### Returns

none.

Referenced by main().

### 7.1073.2.4 main()

```
int main (
 void)
```

This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.

#### Returns

Program exit code.

### 7.1073.2.5 SystemClock\_Config()

```
static void SystemClock_Config (
 void) [static]
```

System Clock Configuration. This code was created by CubeMX and configures the system clock.

#### Returns

none.

Referenced by Init().

### 7.1073.2.6 VectorBase\_Config()

```
static void VectorBase_Config (
 void) [static]
```

Vector base address configuration. It should no longer be at the start of flash memory but moved forward because the first part of flash is reserved for the bootloader. Note that this is already done by the bootloader before starting this program. Unfortunately, function SystemInit() overwrites this change again.

#### Returns

none.

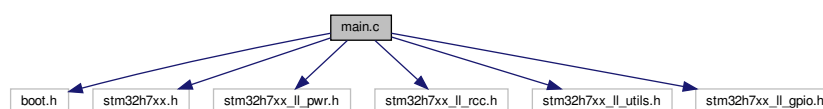
Referenced by Init().

## 7.1074 main.c File Reference

Bootloader application source file.

```
#include "boot.h"
#include "stm32h7xx.h"
#include "stm32h7xx_ll_pwr.h"
#include "stm32h7xx_ll_rcc.h"
#include "stm32h7xx_ll_utils.h"
#include "stm32h7xx_ll_gpio.h"
```

Include dependency graph for ARMC7\_STM32H7\_Nucleo\_H743ZI\_IAR/Boot/main.c:



## Functions

- static void `Init` (void)  
*Initializes the microcontroller.*
- static void `SystemClock_Config` (void)  
*System Clock Configuration. This code was created by CubeMX and configures the system clock to match the configuration in the bootloader's configuration (blt\_conf.h), specifically the macros: BOOT\_CPU\_SYSTEM\_SPEED\_KHZ and BOOT\_CPU\_XTAL\_SPEED\_KHZ. Note that the Lower Layer drivers were selected in CubeMX for the RCC subsystem.*
- void `main` (void)  
*This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.*
- void `HAL_MspInit` (void)  
*Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).*
- void `HAL_MspDeInit` (void)  
*DeInitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level de-initialization (GPIOs, clock, DMA, interrupt).*

### 7.1074.1 Detailed Description

Bootloader application source file.

### 7.1074.2 Function Documentation

#### 7.1074.2.1 HAL\_MspDeInit()

```
void HAL_MspDeInit (
 void)
```

DeInitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level de-initialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

#### 7.1074.2.2 HAL\_MspInit()

```
void HAL_MspInit (
 void)
```

Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

### 7.1074.2.3 Init()

```
static void Init (
 void) [static]
```

Initializes the microcontroller.

#### Returns

none.

Referenced by main().

### 7.1074.2.4 main()

```
void main (
 void)
```

This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.

#### Returns

none.

### 7.1074.2.5 SystemClock\_Config()

```
static void SystemClock_Config (
 void) [static]
```

System Clock Configuration. This code was created by CubeMX and configures the system clock to match the configuration in the bootloader's configuration (blt\_conf.h), specifically the macros: BOOT\_CPU\_SYSTEM\_SPEED\_KHZ and BOOT\_CPU\_XTAL\_SPEED\_KHZ. Note that the Lower Layer drivers were selected in CubeMX for the RCC subsystem.

#### Returns

none.

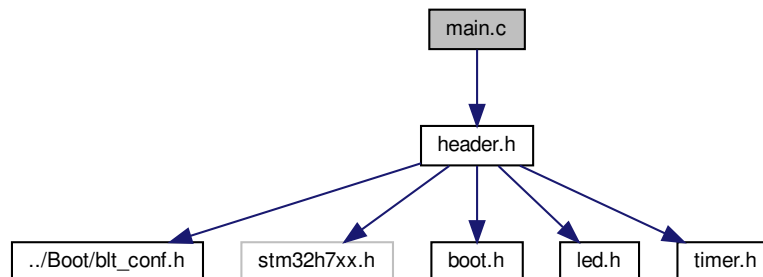
Referenced by Init().

## 7.1075 main.c File Reference

Demo program application source file.

```
#include "header.h"
```

Include dependency graph for ARMCM7\_STM32H7\_Nucleo\_H743ZI\_IAR/Prog/main.c:



### Functions

- static void `Init` (void)  
*Initializes the microcontroller.*
- static void `SystemClock_Config` (void)  
*System Clock Configuration. This code was created by CubeMX and configures the system clock.*
- static void `VectorBase_Config` (void)  
*Vector base address configuration. It should no longer be at the start of flash memory but moved forward because the first part of flash is reserved for the bootloader. Note that this is already done by the bootloader before starting this program. Unfortunately, function `SystemInit()` overwrites this change again.*
- void `main` (void)  
*This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.*
- void `HAL_MspInit` (void)  
*Initializes the Global MSP. This function is called from `HAL_Init()` function to perform system level initialization (GPIOs, clock, DMA, interrupt).*
- void `HAL_MspDeInit` (void)  
*Deinitializes the Global MSP. This function is called from `HAL_DeInit()` function to perform system level Deinitialization (GPIOs, clock, DMA, interrupt).*

### 7.1075.1 Detailed Description

Demo program application source file.

### 7.1075.2 Function Documentation

### 7.1075.2.1 HAL\_MspDeInit()

```
void HAL_MspDeInit (
 void)
```

Deinitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level Deinitialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

### 7.1075.2.2 HAL\_MspInit()

```
void HAL_MspInit (
 void)
```

Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

### 7.1075.2.3 Init()

```
static void Init (
 void) [static]
```

Initializes the microcontroller.

#### Returns

none.

Referenced by main().

### 7.1075.2.4 main()

```
void main (
 void)
```

This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.

#### Returns

none.



### 7.1075.2.5 SystemClock\_Config()

```
static void SystemClock_Config (
 void) [static]
```

System Clock Configuration. This code was created by CubeMX and configures the system clock.

#### Returns

none.

Referenced by Init().

### 7.1075.2.6 VectorBase\_Config()

```
static void VectorBase_Config (
 void) [static]
```

Vector base address configuration. It should no longer be at the start of flash memory but moved forward because the first part of flash is reserved for the bootloader. Note that this is already done by the bootloader before starting this program. Unfortunately, function SystemInit() overwrites this change again.

#### Returns

none.

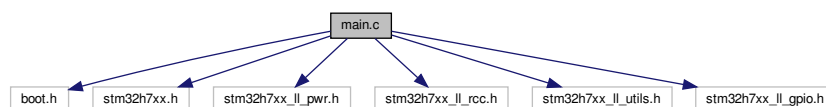
Referenced by Init().

## 7.1076 main.c File Reference

Bootloader application source file.

```
#include "boot.h"
#include "stm32h7xx.h"
#include "stm32h7xx_ll_pwr.h"
#include "stm32h7xx_ll_rcc.h"
#include "stm32h7xx_ll_utils.h"
#include "stm32h7xx_ll_gpio.h"
```

Include dependency graph for ARMC7\_STM32H7\_Nucleo\_H743ZI\_Keil/Boot/main.c:



## Functions

- static void `Init` (void)  
*Initializes the microcontroller.*
- static void `SystemClock_Config` (void)  
*System Clock Configuration. This code was created by CubeMX and configures the system clock to match the configuration in the bootloader's configuration (blt\_conf.h), specifically the macros: BOOT\_CPU\_SYSTEM\_SPEED\_KHZ and BOOT\_CPU\_XTAL\_SPEED\_KHZ. Note that the Lower Layer drivers were selected in CubeMX for the RCC subsystem.*
- int `main` (void)  
*This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.*
- void `HAL_MspInit` (void)  
*Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).*
- void `HAL_MspDeInit` (void)  
*DeInitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level de-initialization (GPIOs, clock, DMA, interrupt).*

### 7.1076.1 Detailed Description

Bootloader application source file.

### 7.1076.2 Function Documentation

#### 7.1076.2.1 HAL\_MspDeInit()

```
void HAL_MspDeInit (
 void)
```

DeInitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level de-initialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

#### 7.1076.2.2 HAL\_MspInit()

```
void HAL_MspInit (
 void)
```

Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

### 7.1076.2.3 Init()

```
static void Init (
 void) [static]
```

Initializes the microcontroller.

#### Returns

none.

Referenced by main().

### 7.1076.2.4 main()

```
int main (
 void)
```

This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.

#### Returns

Program return code.

### 7.1076.2.5 SystemClock\_Config()

```
static void SystemClock_Config (
 void) [static]
```

System Clock Configuration. This code was created by CubeMX and configures the system clock to match the configuration in the bootloader's configuration (blt\_conf.h), specifically the macros: BOOT\_CPU\_SYSTEM\_SPEED\_KHZ and BOOT\_CPU\_XTAL\_SPEED\_KHZ. Note that the Lower Layer drivers were selected in CubeMX for the RCC subsystem.

#### Returns

none.

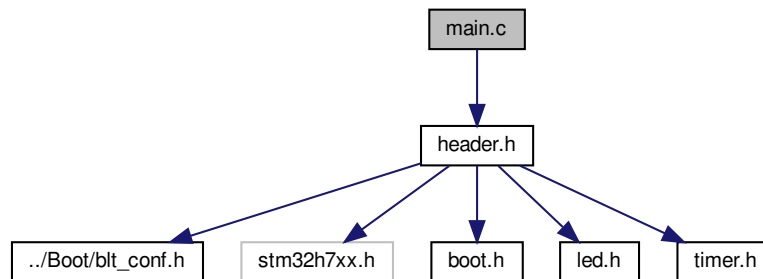
Referenced by Init().

## 7.1077 main.c File Reference

Demo program application source file.

```
#include "header.h"
```

Include dependency graph for ARMCM7\_STM32H7\_Nucleo\_H743ZI\_Keil/Prog/main.c:



### Functions

- static void `Init` (void)  
*Initializes the microcontroller.*
- static void `SystemClock_Config` (void)  
*System Clock Configuration. This code was created by CubeMX and configures the system clock.*
- static void `VectorBase_Config` (void)  
*Vector base address configuration. It should no longer be at the start of flash memory but moved forward because the first part of flash is reserved for the bootloader. Note that this is already done by the bootloader before starting this program. Unfortunately, function `SystemInit()` overwrites this change again.*
- int `main` (void)  
*This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.*
- void `HAL_MspInit` (void)  
*Initializes the Global MSP. This function is called from `HAL_Init()` function to perform system level initialization (GPIOs, clock, DMA, interrupt).*
- void `HAL_MspDeInit` (void)  
*Deinitializes the Global MSP. This function is called from `HAL_DeInit()` function to perform system level Deinitialization (GPIOs, clock, DMA, interrupt).*

### 7.1077.1 Detailed Description

Demo program application source file.

### 7.1077.2 Function Documentation

### 7.1077.2.1 HAL\_MspDeInit()

```
void HAL_MspDeInit (
 void)
```

Deinitializes the Global MSP. This function is called from HAL\_DeInit() function to perform system level Deinitialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

### 7.1077.2.2 HAL\_MspInit()

```
void HAL_MspInit (
 void)
```

Initializes the Global MSP. This function is called from HAL\_Init() function to perform system level initialization (GPIOs, clock, DMA, interrupt).

#### Returns

none.

### 7.1077.2.3 Init()

```
static void Init (
 void) [static]
```

Initializes the microcontroller.

#### Returns

none.

Referenced by main().

### 7.1077.2.4 main()

```
int main (
 void)
```

This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.

#### Returns

Program exit code.

### 7.1077.2.5 SystemClock\_Config()

```
static void SystemClock_Config (
 void) [static]
```

System Clock Configuration. This code was created by CubeMX and configures the system clock.

#### Returns

none.

Referenced by Init().

### 7.1077.2.6 VectorBase\_Config()

```
static void VectorBase_Config (
 void) [static]
```

Vector base address configuration. It should no longer be at the start of flash memory but moved forward because the first part of flash is reserved for the bootloader. Note that this is already done by the bootloader before starting this program. Unfortunately, function SystemInit() overwrites this change again.

#### Returns

none.

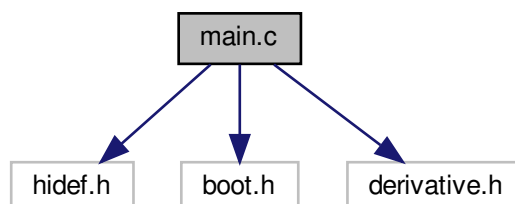
Referenced by Init().

## 7.1078 main.c File Reference

Demo program application source file.

```
#include <hidef.h>
#include "boot.h"
#include "derivative.h"
```

Include dependency graph for HCS12\_DevKit\_S12G128\_CodeWarrior/Boot/main.c:



## Functions

- static void `Init` (void)  
*Initializes the microcontroller.*
- static void `SystemClockInit` (void)  
*Initializes the clock configuration of the microcontroller.*
- void `main` (void)  
*This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.*

### 7.1078.1 Detailed Description

Demo program application source file.

### 7.1078.2 Function Documentation

#### 7.1078.2.1 `Init()`

```
static void Init (
 void) [static]
```

Initializes the microcontroller.

##### Returns

none.

Referenced by `main()`.

#### 7.1078.2.2 `main()`

```
void main (
 void)
```

This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.

##### Returns

none.

### 7.1078.2.3 SystemClockInit()

```
static void SystemClockInit (
 void) [static]
```

Initializes the clock configuration of the microcontroller.

#### Returns

none.

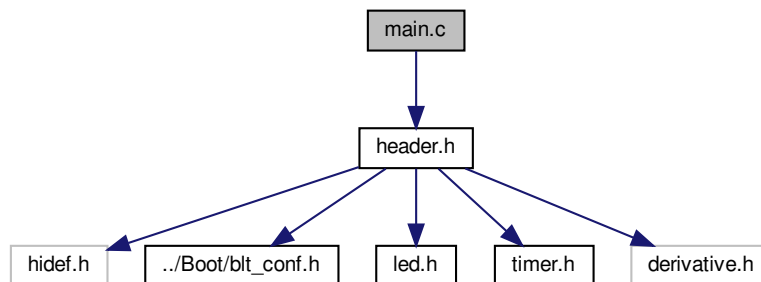
Referenced by Init().

## 7.1079 main.c File Reference

Demo program application source file.

```
#include "header.h"
```

Include dependency graph for HCS12\_DevKit\_S12G128\_CodeWarrior/Prog/main.c:



## Functions

- static void `Init` (void)  
*Initializes the microcontroller.*
- static void `SystemClockInit` (void)  
*Initializes the clock configuration of the microcontroller.*
- void `main` (void)  
*This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.*

### 7.1079.1 Detailed Description

Demo program application source file.



## 7.1079.2 Function Documentation

### 7.1079.2.1 Init()

```
static void Init (
 void) [static]
```

Initializes the microcontroller.

#### Returns

none.

Referenced by main().

### 7.1079.2.2 main()

```
void main (
 void)
```

This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.

#### Returns

none.

### 7.1079.2.3 SystemClockInit()

```
static void SystemClockInit (
 void) [static]
```

Initializes the clock configuration of the microcontroller.

#### Returns

none.

Referenced by Init().

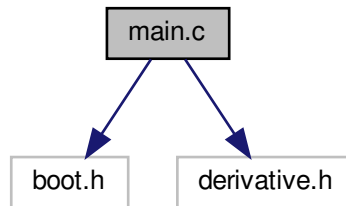
## 7.1080 main.c File Reference

Bootloader application source file.

```
#include "boot.h"
```

```
#include "derivative.h"
```

Include dependency graph for HCS12\_Evbplus\_Dragon12p\_CodeWarrior/Boot/main.c:



### Functions

- static void `Init` (void)  
*Initializes the microcontroller.*
- void `main` (void)  
*This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.*

### 7.1080.1 Detailed Description

Bootloader application source file.

### 7.1080.2 Function Documentation

#### 7.1080.2.1 Init()

```
static void Init (
 void) [static]
```

Initializes the microcontroller.

#### Returns

none.

Referenced by `main()`.

## 7.1080.2.2 main()

```
void main (
 void)
```

This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.

## Returns

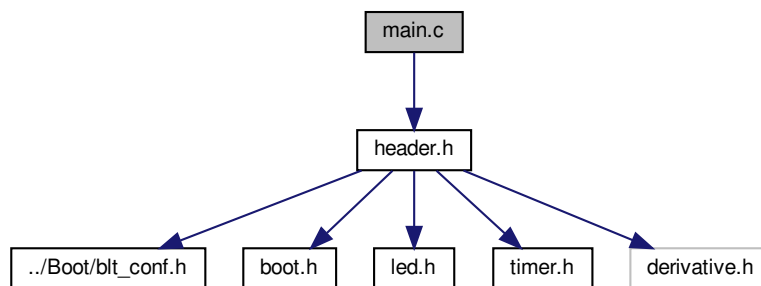
none.

## 7.1081 main.c File Reference

Demo program application source file.

```
#include "header.h"
```

Include dependency graph for HCS12\_Evbplus\_Dragon12p\_CodeWarrior/Prog/main.c:



## Functions

- static void `Init` (void)  
*Initializes the microcontroller.*
- static void `SysClockInit` (void)  
*Initializes the microcontroller.*
- void `main` (void)  
*This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.*

## 7.1081.1 Detailed Description

Demo program application source file.

## 7.1081.2 Function Documentation

### 7.1081.2.1 Init()

```
static void Init (
 void) [static]
```

Initializes the microcontroller.

#### Returns

none.

Referenced by main().

### 7.1081.2.2 main()

```
void main (
 void)
```

This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.

#### Returns

none.

### 7.1081.2.3 SysClockInit()

```
static void SysClockInit (
 void) [static]
```

Initializes the microcontroller.

#### Returns

none.

Referenced by Init().



## 7.1082.2 Function Documentation

### 7.1082.2.1 NetApp()

```
void NetApp (
 void)
```

The uIP network application that detects the XCP connect command on the port used by the bootloader. This indicates that the bootloader should be activated.

#### Returns

none.

### 7.1082.2.2 NetInit()

```
void NetInit (
 void)
```

Initializes the TCP/IP network communication interface.

#### Returns

none.

Referenced by AppInit(), ComInit(), and main().

### 7.1082.2.3 NetTask()

```
void NetTask (
 void)
```

Runs the TCP/IP server task.

#### Returns

none.

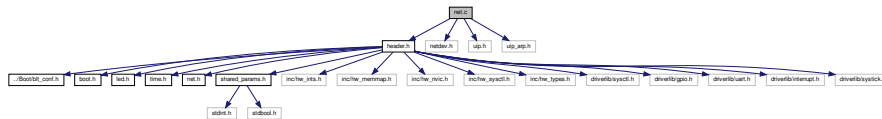
Referenced by AppTask(), and main().

## 7.1083 net.c File Reference

Network application for the uIP TCP/IP stack.

```
#include "header.h"
#include "netdev.h"
#include "uip.h"
#include "uip_arp.h"
```

Include dependency graph for Demo/ARMCM3\_LM3S\_EK\_LM3S6965\_IAR/Prog/net.c:



### Macros

- `#define NET_UIP_PERIODIC_TIMER_MS (500)`  
*Delta time for the uIP periodic timer.*
- `#define NET_UIP_ARP_TIMER_MS (10000)`  
*Delta time for the uIP ARP timer.*
- `#define NET_UIP_HEADER_BUF ((struct uip_eth_hdr *)&uip_buf[0])`  
*Macro for accessing the Ethernet header information in the buffer.*

### Functions

- void `NetInit` (void)  
*Initializes the TCP/IP network communication interface.*
- void `NetApp` (void)  
*The uIP network application that detects the XCP connect command on the port used by the bootloader. This indicates that the bootloader should be activated.*
- void `NetTask` (void)  
*Runs the TCP/IP server task.*

### Variables

- static unsigned long `periodicTimerTimeout`  
*Holds the time out value of the uIP periodic timer.*
- static unsigned long `ARPTimerTimeout`  
*Holds the time out value of the uIP ARP timer.*
- static struct uip\_eth\_addr `macAddress`  
*Holds the MAC address which is used by the DHCP client.*

#### 7.1083.1 Detailed Description

Network application for the uIP TCP/IP stack.

## 7.1083.2 Function Documentation

### 7.1083.2.1 NetApp()

```
void NetApp (
 void)
```

The uIP network application that detects the XCP connect command on the port used by the bootloader. This indicates that the bootloader should be activated.

#### Returns

none.

### 7.1083.2.2 NetInit()

```
void NetInit (
 void)
```

Initializes the TCP/IP network communication interface.

#### Returns

none.

### 7.1083.2.3 NetTask()

```
void NetTask (
 void)
```

Runs the TCP/IP server task.

#### Returns

none.

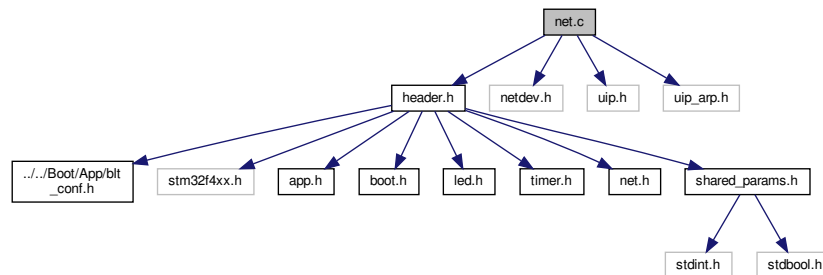


## 7.1084 net.c File Reference

Network application for the uIP TCP/IP stack.

```
#include "header.h"
#include "netdev.h"
#include "uip.h"
#include "uip_arp.h"
```

Include dependency graph for Demo/ARMCM4\_STM32F4\_Nucleo\_F429ZI\_CubeIDE/Prog/App/net.c:



### Macros

- `#define NET_UIP_PERIODIC_TIMER_MS (500)`  
*Delta time for the uIP periodic timer.*
- `#define NET_UIP_ARP_TIMER_MS (10000)`  
*Delta time for the uIP ARP timer.*
- `#define NET_UIP_HEADER_BUF ((struct uip_eth_hdr *)&uip_buf[0])`  
*Macro for accessing the Ethernet header information in the buffer.*

### Functions

- void `NetInit` (void)  
*Initializes the TCP/IP network communication interface.*
- void `NetApp` (void)  
*The uIP network application that detects the XCP connect command on the port used by the bootloader. This indicates that the bootloader should be activated.*
- void `NetTask` (void)  
*Runs the TCP/IP server task.*

### Variables

- static unsigned long `periodicTimerTimeOut`  
*Holds the time out value of the uIP periodic timer.*
- static unsigned long `ARPTimerTimeOut`  
*Holds the time out value of the uIP ARP timer.*
- static struct uip\_eth\_addr `macAddress`  
*Holds the MAC address which is used by the DHCP client.*

### 7.1084.1 Detailed Description

Network application for the uIP TCP/IP stack.

### 7.1084.2 Function Documentation

#### 7.1084.2.1 NetApp()

```
void NetApp (
 void)
```

The uIP network application that detects the XCP connect command on the port used by the bootloader. This indicates that the bootloader should be activated.

##### Returns

none.

#### 7.1084.2.2 NetInit()

```
void NetInit (
 void)
```

Initializes the TCP/IP network communication interface.

##### Returns

none.

#### 7.1084.2.3 NetTask()

```
void NetTask (
 void)
```

Runs the TCP/IP server task.

##### Returns

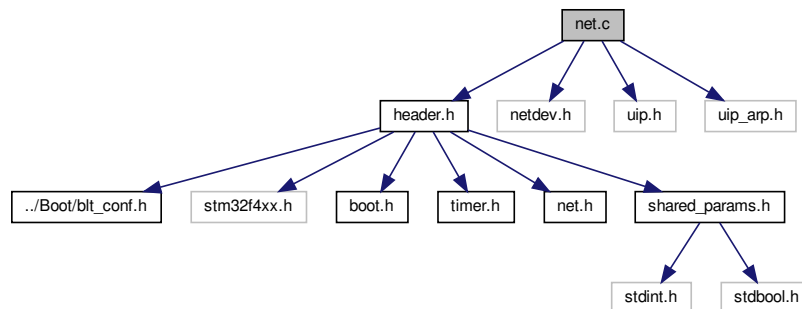
none.

## 7.1085 net.c File Reference

Network application for the uIP TCP/IP stack.

```
#include "header.h"
#include "netdev.h"
#include "uip.h"
#include "uip_arp.h"
```

Include dependency graph for Demo/ARMCM4\_STM32F4\_Nucleo\_F429ZI\_GCC/Prog/net.c:



### Macros

- `#define NET_UIP_PERIODIC_TIMER_MS (500)`  
*Delta time for the uIP periodic timer.*
- `#define NET_UIP_ARP_TIMER_MS (10000)`  
*Delta time for the uIP ARP timer.*
- `#define NET_UIP_HEADER_BUF ((struct uip_eth_hdr *)&uip_buf[0])`  
*Macro for accessing the Ethernet header information in the buffer.*

### Functions

- void `NetInit` (void)  
*Initializes the TCP/IP network communication interface.*
- void `NetApp` (void)  
*The uIP network application that detects the XCP connect command on the port used by the bootloader. This indicates that the bootloader should be activated.*
- void `NetTask` (void)  
*Runs the TCP/IP server task.*

### Variables

- static unsigned long `periodicTimerTimeOut`  
*Holds the time out value of the uIP periodic timer.*
- static unsigned long `ARPTimerTimeOut`  
*Holds the time out value of the uIP ARP timer.*
- static struct uip\_eth\_addr `macAddress`  
*Holds the MAC address which is used by the DHCP client.*

### 7.1085.1 Detailed Description

Network application for the uIP TCP/IP stack.

### 7.1085.2 Function Documentation

#### 7.1085.2.1 NetApp()

```
void NetApp (
 void)
```

The uIP network application that detects the XCP connect command on the port used by the bootloader. This indicates that the bootloader should be activated.

##### Returns

none.

#### 7.1085.2.2 NetInit()

```
void NetInit (
 void)
```

Initializes the TCP/IP network communication interface.

##### Returns

none.

#### 7.1085.2.3 NetTask()

```
void NetTask (
 void)
```

Runs the TCP/IP server task.

##### Returns

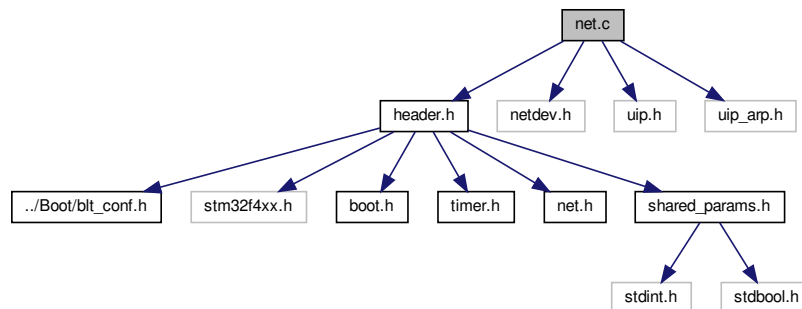
none.

## 7.1086 net.c File Reference

Network application for the uIP TCP/IP stack.

```
#include "header.h"
#include "netdev.h"
#include "uip.h"
#include "uip_arp.h"
```

Include dependency graph for Demo/ARMCM4\_STM32F4\_Nucleo\_F429ZI\_IAR/Prog/net.c:



### Macros

- `#define NET_UIP_PERIODIC_TIMER_MS (500)`  
*Delta time for the uIP periodic timer.*
- `#define NET_UIP_ARP_TIMER_MS (10000)`  
*Delta time for the uIP ARP timer.*
- `#define NET_UIP_HEADER_BUF ((struct uip_eth_hdr *)&uip_buf[0])`  
*Macro for accessing the Ethernet header information in the buffer.*

### Functions

- void `NetInit` (void)  
*Initializes the TCP/IP network communication interface.*
- void `NetApp` (void)  
*The uIP network application that detects the XCP connect command on the port used by the bootloader. This indicates that the bootloader should be activated.*
- void `NetTask` (void)  
*Runs the TCP/IP server task.*

### Variables

- static unsigned long `periodicTimerTimeOut`  
*Holds the time out value of the uIP periodic timer.*
- static unsigned long `ARPTimerTimeOut`  
*Holds the time out value of the uIP ARP timer.*
- static struct uip\_eth\_addr `macAddress`  
*Holds the MAC address which is used by the DHCP client.*

### 7.1086.1 Detailed Description

Network application for the uIP TCP/IP stack.

### 7.1086.2 Function Documentation

#### 7.1086.2.1 NetApp()

```
void NetApp (
 void)
```

The uIP network application that detects the XCP connect command on the port used by the bootloader. This indicates that the bootloader should be activated.

##### Returns

none.

#### 7.1086.2.2 NetInit()

```
void NetInit (
 void)
```

Initializes the TCP/IP network communication interface.

##### Returns

none.

#### 7.1086.2.3 NetTask()

```
void NetTask (
 void)
```

Runs the TCP/IP server task.

##### Returns

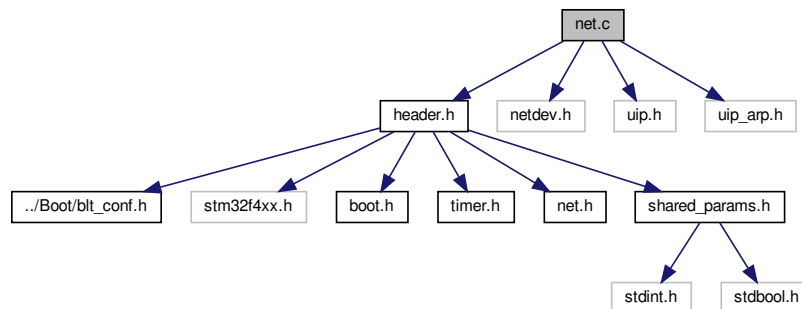
none.

## 7.1087 net.c File Reference

Network application for the uIP TCP/IP stack.

```
#include "header.h"
#include "netdev.h"
#include "uip.h"
#include "uip_arp.h"
```

Include dependency graph for Demo/ARMCM4\_STM32F4\_Nucleo\_F429ZI\_Keil/Prog/net.c:



### Macros

- `#define NET_UIP_PERIODIC_TIMER_MS (500)`  
*Delta time for the uIP periodic timer.*
- `#define NET_UIP_ARP_TIMER_MS (10000)`  
*Delta time for the uIP ARP timer.*
- `#define NET_UIP_HEADER_BUF ((struct uip_eth_hdr *)&uip_buf[0])`  
*Macro for accessing the Ethernet header information in the buffer.*

### Functions

- void `NetInit` (void)  
*Initializes the TCP/IP network communication interface.*
- void `NetApp` (void)  
*The uIP network application that detects the XCP connect command on the port used by the bootloader. This indicates that the bootloader should be activated.*
- void `NetTask` (void)  
*Runs the TCP/IP server task.*

### Variables

- static unsigned long `periodicTimerTimeOut`  
*Holds the time out value of the uIP periodic timer.*
- static unsigned long `ARPTimerTimeOut`  
*Holds the time out value of the uIP ARP timer.*
- static struct uip\_eth\_addr `macAddress`  
*Holds the MAC address which is used by the DHCP client.*

### 7.1087.1 Detailed Description

Network application for the uIP TCP/IP stack.

### 7.1087.2 Function Documentation

#### 7.1087.2.1 NetApp()

```
void NetApp (
 void)
```

The uIP network application that detects the XCP connect command on the port used by the bootloader. This indicates that the bootloader should be activated.

##### Returns

none.

#### 7.1087.2.2 NetInit()

```
void NetInit (
 void)
```

Initializes the TCP/IP network communication interface.

##### Returns

none.

#### 7.1087.2.3 NetTask()

```
void NetTask (
 void)
```

Runs the TCP/IP server task.

##### Returns

none.

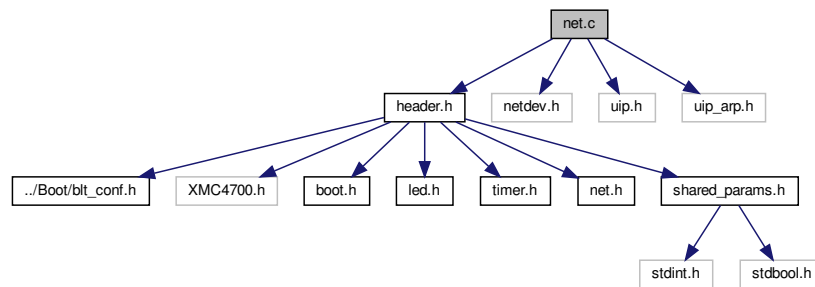


## 7.1088 net.c File Reference

Network application for the uIP TCP/IP stack.

```
#include "header.h"
#include "netdev.h"
#include "uip.h"
#include "uip_arp.h"
```

Include dependency graph for Demo/ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_GCC/Prog/net.c:



### Macros

- `#define NET_UIP_PERIODIC_TIMER_MS (500)`  
*Delta time for the uIP periodic timer.*
- `#define NET_UIP_ARP_TIMER_MS (10000)`  
*Delta time for the uIP ARP timer.*
- `#define NET_UIP_HEADER_BUF ((struct uip_eth_hdr *)&uip_buf[0])`  
*Macro for accessing the Ethernet header information in the buffer.*

### Functions

- void `NetInit` (void)  
*Initializes the TCP/IP network communication interface.*
- void `NetApp` (void)  
*The uIP network application that detects the XCP connect command on the port used by the bootloader. This indicates that the bootloader should be activated.*
- void `NetTask` (void)  
*Runs the TCP/IP server task.*

### Variables

- static unsigned long `periodicTimerTimeOut`  
*Holds the time out value of the uIP periodic timer.*
- static unsigned long `ARPTimerTimeOut`  
*Holds the time out value of the uIP ARP timer.*
- static struct uip\_eth\_addr `macAddress`  
*Holds the MAC address which is used by the DHCP client.*

### 7.1088.1 Detailed Description

Network application for the uIP TCP/IP stack.

### 7.1088.2 Function Documentation

#### 7.1088.2.1 NetApp()

```
void NetApp (
 void)
```

The uIP network application that detects the XCP connect command on the port used by the bootloader. This indicates that the bootloader should be activated.

##### Returns

none.

#### 7.1088.2.2 NetInit()

```
void NetInit (
 void)
```

Initializes the TCP/IP network communication interface.

##### Returns

none.

#### 7.1088.2.3 NetTask()

```
void NetTask (
 void)
```

Runs the TCP/IP server task.

##### Returns

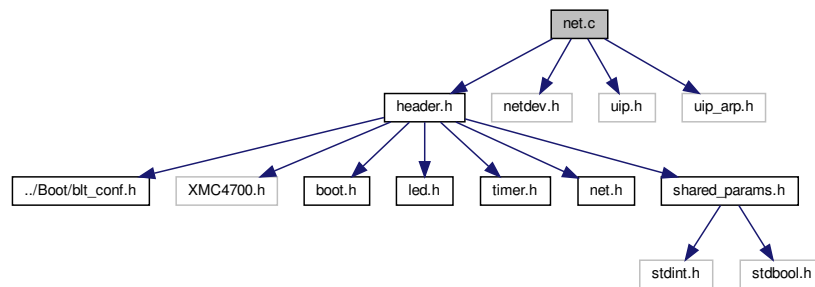
none.

## 7.1089 net.c File Reference

Network application for the uIP TCP/IP stack.

```
#include "header.h"
#include "netdev.h"
#include "uip.h"
#include "uip_arp.h"
```

Include dependency graph for Demo/ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_IAR/Prog/net.c:



### Macros

- `#define NET_UIP_PERIODIC_TIMER_MS (500)`  
*Delta time for the uIP periodic timer.*
- `#define NET_UIP_ARP_TIMER_MS (10000)`  
*Delta time for the uIP ARP timer.*
- `#define NET_UIP_HEADER_BUF ((struct uip_eth_hdr *)&uip_buf[0])`  
*Macro for accessing the Ethernet header information in the buffer.*

### Functions

- void `NetInit` (void)  
*Initializes the TCP/IP network communication interface.*
- void `NetApp` (void)  
*The uIP network application that detects the XCP connect command on the port used by the bootloader. This indicates that the bootloader should be activated.*
- void `NetTask` (void)  
*Runs the TCP/IP server task.*

### Variables

- static unsigned long `periodicTimerTimeOut`  
*Holds the time out value of the uIP periodic timer.*
- static unsigned long `ARPTimerTimeOut`  
*Holds the time out value of the uIP ARP timer.*
- static struct uip\_eth\_addr `macAddress`  
*Holds the MAC address which is used by the DHCP client.*

### 7.1089.1 Detailed Description

Network application for the uIP TCP/IP stack.

### 7.1089.2 Function Documentation

#### 7.1089.2.1 NetApp()

```
void NetApp (
 void)
```

The uIP network application that detects the XCP connect command on the port used by the bootloader. This indicates that the bootloader should be activated.

##### Returns

none.

#### 7.1089.2.2 NetInit()

```
void NetInit (
 void)
```

Initializes the TCP/IP network communication interface.

##### Returns

none.

#### 7.1089.2.3 NetTask()

```
void NetTask (
 void)
```

Runs the TCP/IP server task.

##### Returns

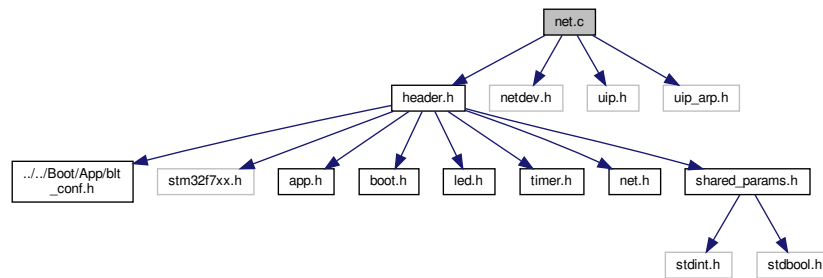
none.

## 7.1090 net.c File Reference

Network application for the uIP TCP/IP stack.

```
#include "header.h"
#include "netdev.h"
#include "uip.h"
#include "uip_arp.h"
```

Include dependency graph for Demo/ARMCM7\_STM32F7\_Nucleo\_F767ZI\_CubeIDE/Prog/App/net.c:



### Macros

- `#define NET_UIP_PERIODIC_TIMER_MS (500)`  
*Delta time for the uIP periodic timer.*
- `#define NET_UIP_ARP_TIMER_MS (10000)`  
*Delta time for the uIP ARP timer.*
- `#define NET_UIP_HEADER_BUF ((struct uip_eth_hdr *)&uip_buf[0])`  
*Macro for accessing the Ethernet header information in the buffer.*

### Functions

- void `NetInit` (void)  
*Initializes the TCP/IP network communication interface.*
- void `NetApp` (void)  
*The uIP network application that detects the XCP connect command on the port used by the bootloader. This indicates that the bootloader should be activated.*
- void `NetTask` (void)  
*Runs the TCP/IP server task.*

### Variables

- static unsigned long `periodicTimerTimeOut`  
*Holds the time out value of the uIP periodic timer.*
- static unsigned long `ARPTimerTimeOut`  
*Holds the time out value of the uIP ARP timer.*
- static struct uip\_eth\_addr `macAddress`  
*Holds the MAC address which is used by the DHCP client.*

### 7.1090.1 Detailed Description

Network application for the uIP TCP/IP stack.

### 7.1090.2 Function Documentation

#### 7.1090.2.1 NetApp()

```
void NetApp (
 void)
```

The uIP network application that detects the XCP connect command on the port used by the bootloader. This indicates that the bootloader should be activated.

##### Returns

none.

#### 7.1090.2.2 NetInit()

```
void NetInit (
 void)
```

Initializes the TCP/IP network communication interface.

##### Returns

none.

#### 7.1090.2.3 NetTask()

```
void NetTask (
 void)
```

Runs the TCP/IP server task.

##### Returns

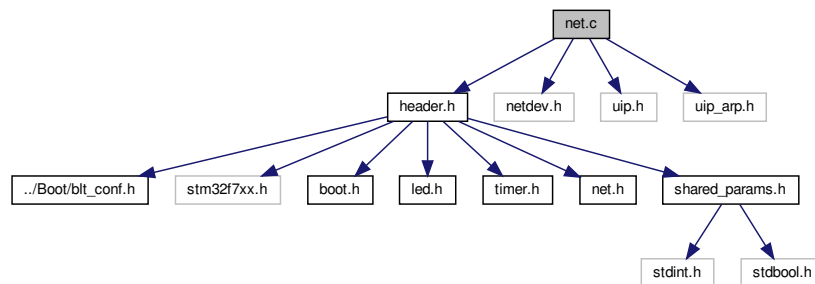
none.

## 7.1091 net.c File Reference

Network application for the uIP TCP/IP stack.

```
#include "header.h"
#include "netdev.h"
#include "uip.h"
#include "uip_arp.h"
```

Include dependency graph for Demo/ARMCM7\_STM32F7\_Nucleo\_F767ZI\_GCC/Prog/net.c:



### Macros

- `#define NET_UIP_PERIODIC_TIMER_MS (500)`  
*Delta time for the uIP periodic timer.*
- `#define NET_UIP_ARP_TIMER_MS (10000)`  
*Delta time for the uIP ARP timer.*
- `#define NET_UIP_HEADER_BUF ((struct uip_eth_hdr *)&uip_buf[0])`  
*Macro for accessing the Ethernet header information in the buffer.*

### Functions

- void `NetInit` (void)  
*Initializes the TCP/IP network communication interface.*
- void `NetApp` (void)  
*The uIP network application that detects the XCP connect command on the port used by the bootloader. This indicates that the bootloader should be activated.*
- void `NetTask` (void)  
*Runs the TCP/IP server task.*

### Variables

- static unsigned long `periodicTimerTimeOut`  
*Holds the time out value of the uIP periodic timer.*
- static unsigned long `ARPTimerTimeOut`  
*Holds the time out value of the uIP ARP timer.*
- static struct uip\_eth\_addr `macAddress`  
*Holds the MAC address which is used by the DHCP client.*

### 7.1091.1 Detailed Description

Network application for the uIP TCP/IP stack.

### 7.1091.2 Function Documentation

#### 7.1091.2.1 NetApp()

```
void NetApp (
 void)
```

The uIP network application that detects the XCP connect command on the port used by the bootloader. This indicates that the bootloader should be activated.

##### Returns

none.

#### 7.1091.2.2 NetInit()

```
void NetInit (
 void)
```

Initializes the TCP/IP network communication interface.

##### Returns

none.

#### 7.1091.2.3 NetTask()

```
void NetTask (
 void)
```

Runs the TCP/IP server task.

##### Returns

none.

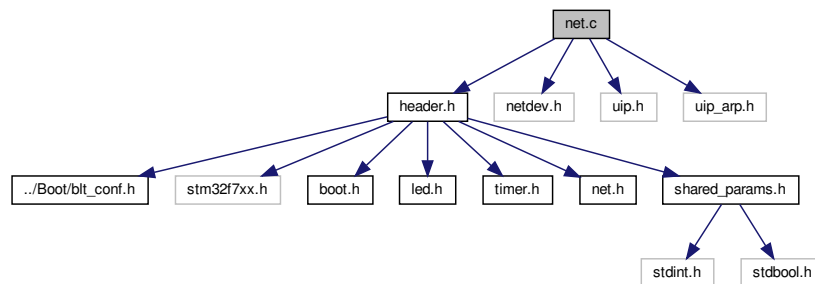


## 7.1092 net.c File Reference

Network application for the uIP TCP/IP stack.

```
#include "header.h"
#include "netdev.h"
#include "uip.h"
#include "uip_arp.h"
```

Include dependency graph for Demo/ARMCM7\_STM32F7\_Nucleo\_F767ZI\_IAR/Prog/net.c:



### Macros

- `#define NET_UIP_PERIODIC_TIMER_MS (500)`  
*Delta time for the uIP periodic timer.*
- `#define NET_UIP_ARP_TIMER_MS (10000)`  
*Delta time for the uIP ARP timer.*
- `#define NET_UIP_HEADER_BUF ((struct uip_eth_hdr *)&uip_buf[0])`  
*Macro for accessing the Ethernet header information in the buffer.*

### Functions

- void `NetInit` (void)  
*Initializes the TCP/IP network communication interface.*
- void `NetApp` (void)  
*The uIP network application that detects the XCP connect command on the port used by the bootloader. This indicates that the bootloader should be activated.*
- void `NetTask` (void)  
*Runs the TCP/IP server task.*

### Variables

- static unsigned long `periodicTimerTimeOut`  
*Holds the time out value of the uIP periodic timer.*
- static unsigned long `ARPTimerTimeOut`  
*Holds the time out value of the uIP ARP timer.*
- static struct uip\_eth\_addr `macAddress`  
*Holds the MAC address which is used by the DHCP client.*

### 7.1092.1 Detailed Description

Network application for the uIP TCP/IP stack.

### 7.1092.2 Function Documentation

#### 7.1092.2.1 NetApp()

```
void NetApp (
 void)
```

The uIP network application that detects the XCP connect command on the port used by the bootloader. This indicates that the bootloader should be activated.

##### Returns

none.

#### 7.1092.2.2 NetInit()

```
void NetInit (
 void)
```

Initializes the TCP/IP network communication interface.

##### Returns

none.

#### 7.1092.2.3 NetTask()

```
void NetTask (
 void)
```

Runs the TCP/IP server task.

##### Returns

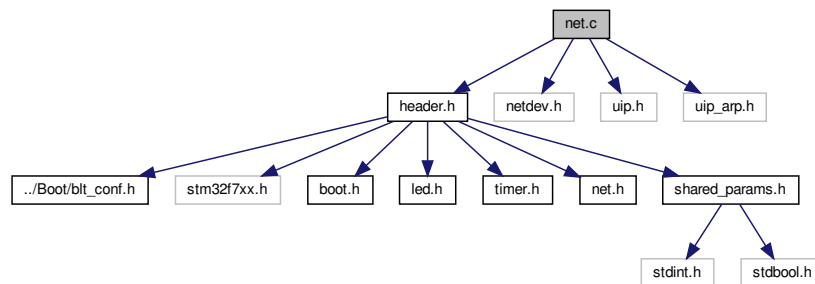
none.

## 7.1093 net.c File Reference

Network application for the uIP TCP/IP stack.

```
#include "header.h"
#include "netdev.h"
#include "uip.h"
#include "uip_arp.h"
```

Include dependency graph for Demo/ARMCM7\_STM32F7\_Nucleo\_F767ZI\_Keil/Prog/net.c:



### Macros

- `#define NET_UIP_PERIODIC_TIMER_MS (500)`  
*Delta time for the uIP periodic timer.*
- `#define NET_UIP_ARP_TIMER_MS (10000)`  
*Delta time for the uIP ARP timer.*
- `#define NET_UIP_HEADER_BUF ((struct uip_eth_hdr *)&uip_buf[0])`  
*Macro for accessing the Ethernet header information in the buffer.*

### Functions

- void `NetInit` (void)  
*Initializes the TCP/IP network communication interface.*
- void `NetApp` (void)  
*The uIP network application that detects the XCP connect command on the port used by the bootloader. This indicates that the bootloader should be activated.*
- void `NetTask` (void)  
*Runs the TCP/IP server task.*

### Variables

- static unsigned long `periodicTimerTimeOut`  
*Holds the time out value of the uIP periodic timer.*
- static unsigned long `ARPTimerTimeOut`  
*Holds the time out value of the uIP ARP timer.*
- static struct uip\_eth\_addr `macAddress`  
*Holds the MAC address which is used by the DHCP client.*

### 7.1093.1 Detailed Description

Network application for the uIP TCP/IP stack.

### 7.1093.2 Function Documentation

#### 7.1093.2.1 NetApp()

```
void NetApp (
 void)
```

The uIP network application that detects the XCP connect command on the port used by the bootloader. This indicates that the bootloader should be activated.

##### Returns

none.

#### 7.1093.2.2 NetInit()

```
void NetInit (
 void)
```

Initializes the TCP/IP network communication interface.

##### Returns

none.

#### 7.1093.2.3 NetTask()

```
void NetTask (
 void)
```

Runs the TCP/IP server task.

##### Returns

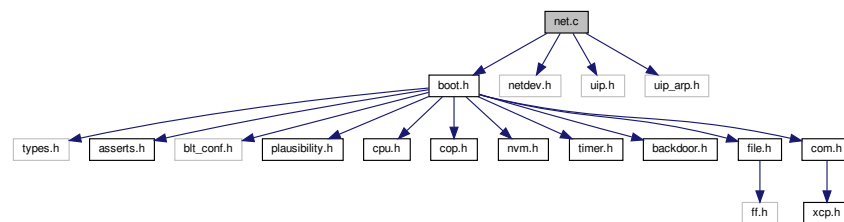
none.

## 7.1094 net.c File Reference

Bootloader TCP/IP network communication interface source file.

```
#include "boot.h"
#include "netdev.h"
#include "uip.h"
#include "uip_arp.h"
```

Include dependency graph for Source/net.c:



### Macros

- `#define NET_UIP_PERIODIC_TIMER_MS (500)`  
*Delta time for the uIP periodic timer.*
- `#define NET_UIP_ARP_TIMER_MS (10000)`  
*Delta time for the uIP ARP timer.*
- `#define NET_UIP_HEADER_BUF ((struct uip_eth_hdr *)&uip_buf[0])`  
*Macro for accessing the Ethernet header information in the buffer.*

### Functions

- static void `NetServerTask` (void)  
*Runs the TCP/IP server task.*
- void `NetInit` (void)  
*Initializes the TCP/IP network communication interface.*
- void `NetTransmitPacket` (blt\_int8u \*data, blt\_int8u len)  
*Transmits a packet formatted for the communication interface.*
- blt\_bool `NetReceivePacket` (blt\_int8u \*data, blt\_int8u \*len)  
*Receives a communication interface packet if one is present.*
- void `NetApp` (void)  
*The uIP network application that implements XCP on TCP/IP. Note that this application make use of the fact that XCP is request/response based. So no new request will come in when a response is pending for transmission, if so, the transmission of the pending response is aborted.*

### Variables

- static blt\_int32u `periodicTimerTimeOut`  
*Holds the time out value of the uIP periodic timer.*
- static blt\_int32u `ARPTimerTimeOut`  
*Holds the time out value of the uIP ARP timer.*
- static blt\_bool `netInitializedFlag = BLT_FALSE`  
*Boolean flag to determine if the module was initialized or not.*
- static blt\_bool `netInitializationDeferred = BLT_FALSE`  
*Boolean flag initialized such that the normal initialization via `NetInit()` proceeds as usual.*

### 7.1094.1 Detailed Description

Bootloader TCP/IP network communication interface source file.

### 7.1094.2 Function Documentation

#### 7.1094.2.1 NetApp()

```
void NetApp (
 void)
```

The uIP network application that implements XCP on TCP/IP. Note that this application make use of the fact that XCP is request/response based. So no new request will come in when a response is pending for transmission, if so, the transmission of the pending response is aborted.

The uIP network application that detects the XCP connect command on the port used by the bootloader. This indicates that the bootloader should be activated.

#### Returns

none.

#### 7.1094.2.2 NetInit()

```
void NetInit (
 void)
```

Initializes the TCP/IP network communication interface.

#### Returns

none.

#### 7.1094.2.3 NetReceivePacket()

```
blt_bool NetReceivePacket (
 blt_int8u * data,
 blt_int8u * len)
```

Receives a communication interface packet if one is present.

**Parameters**

|             |                                                         |
|-------------|---------------------------------------------------------|
| <i>data</i> | Pointer to byte array where the data is to be stored.   |
| <i>len</i>  | Pointer where the length of the packet is to be stored. |

**Returns**

BLT\_TRUE if a packet was received, BLT\_FALSE otherwise.

Referenced by ComTask().

**7.1094.2.4 NetServerTask()**

```
static void NetServerTask (
 void) [static]
```

Runs the TCP/IP server task.

**Returns**

none.

Referenced by NetReceivePacket().

**7.1094.2.5 NetTransmitPacket()**

```
void NetTransmitPacket (
 blt_int8u * data,
 blt_int8u len)
```

Transmits a packet formatted for the communication interface.

**Parameters**

|             |                                                            |
|-------------|------------------------------------------------------------|
| <i>data</i> | Pointer to byte array with data that it to be transmitted. |
| <i>len</i>  | Number of bytes that are to be transmitted.                |

**Returns**

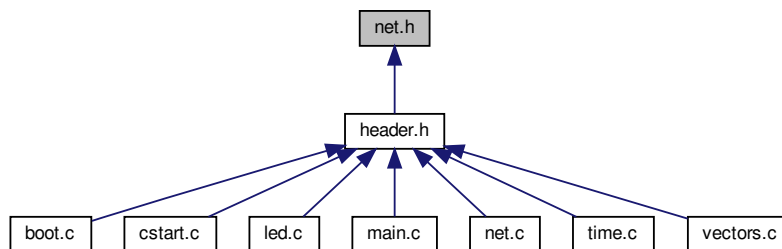
none.

Referenced by ComTransmitPacket().

## 7.1095 net.h File Reference

Network application for the uIP TCP/IP stack.

This graph shows which files directly or indirectly include this file:



### Data Structures

- struct [uip\\_tcp\\_appstate\\_t](#)

Define the [uip\\_tcp\\_appstate\\_t](#) datatype. This is the state of our tcp/ip application, and the memory required for this state is allocated together with each TCP connection. One application state for each TCP connection.

### Functions

- void [NetInit](#) (void)  
*Initializes the TCP/IP network communication interface.*
- void [NetApp](#) (void)  
*The uIP network application that detects the XCP connect command on the port used by the bootloader. This indicates that the bootloader should be activated.*
- void [NetTask](#) (void)  
*Runs the TCP/IP server task.*

#### 7.1095.1 Detailed Description

Network application for the uIP TCP/IP stack.

#### 7.1095.2 Function Documentation



### 7.1095.2.1 NetApp()

```
void NetApp (
 void)
```

The uIP network application that detects the XCP connect command on the port used by the bootloader. This indicates that the bootloader should be activated.

#### Returns

none.

The uIP network application that detects the XCP connect command on the port used by the bootloader. This indicates that the bootloader should be activated.

#### Returns

none.

### 7.1095.2.2 NetInit()

```
void NetInit (
 void)
```

Initializes the TCP/IP network communication interface.

#### Returns

none.

### 7.1095.2.3 NetTask()

```
void NetTask (
 void)
```

Runs the TCP/IP server task.

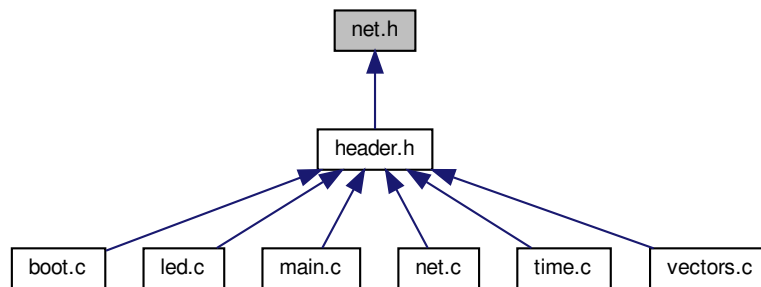
#### Returns

none.

## 7.1096 net.h File Reference

Network application for the uIP TCP/IP stack.

This graph shows which files directly or indirectly include this file:



### Data Structures

- struct [uip\\_tcp\\_appstate\\_t](#)

Define the [uip\\_tcp\\_appstate\\_t](#) datatype. This is the state of our tcp/ip application, and the memory required for this state is allocated together with each TCP connection. One application state for each TCP connection.

### Functions

- void [NetInit](#) (void)  
*Initializes the TCP/IP network communication interface.*
- void [NetApp](#) (void)  
*The uIP network application that detects the XCP connect command on the port used by the bootloader. This indicates that the bootloader should be activated.*
- void [NetTask](#) (void)  
*Runs the TCP/IP server task.*

#### 7.1096.1 Detailed Description

Network application for the uIP TCP/IP stack.

#### 7.1096.2 Function Documentation

### 7.1096.2.1 NetApp()

```
void NetApp (
 void)
```

The uIP network application that detects the XCP connect command on the port used by the bootloader. This indicates that the bootloader should be activated.

#### Returns

none.

The uIP network application that detects the XCP connect command on the port used by the bootloader. This indicates that the bootloader should be activated.

#### Returns

none.

### 7.1096.2.2 NetInit()

```
void NetInit (
 void)
```

Initializes the TCP/IP network communication interface.

#### Returns

none.

### 7.1096.2.3 NetTask()

```
void NetTask (
 void)
```

Runs the TCP/IP server task.

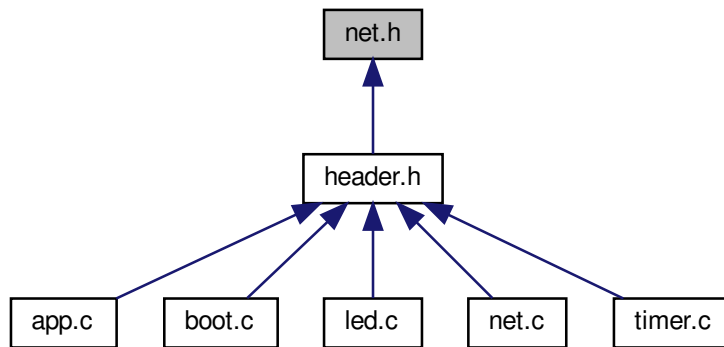
#### Returns

none.

## 7.1097 net.h File Reference

Network application for the uIP TCP/IP stack.

This graph shows which files directly or indirectly include this file:



### Data Structures

- struct [uip\\_tcp\\_appstate\\_t](#)

Define the [uip\\_tcp\\_appstate\\_t](#) datatype. This is the state of our tcp/ip application, and the memory required for this state is allocated together with each TCP connection. One application state for each TCP connection.

### Functions

- void [NetInit](#) (void)  
*Initializes the TCP/IP network communication interface.*
- void [NetApp](#) (void)  
*The uIP network application that detects the XCP connect command on the port used by the bootloader. This indicates that the bootloader should be activated.*
- void [NetTask](#) (void)  
*Runs the TCP/IP server task.*

### 7.1097.1 Detailed Description

Network application for the uIP TCP/IP stack.

### 7.1097.2 Function Documentation

### 7.1097.2.1 NetApp()

```
void NetApp (
 void)
```

The uIP network application that detects the XCP connect command on the port used by the bootloader. This indicates that the bootloader should be activated.

#### Returns

none.

The uIP network application that detects the XCP connect command on the port used by the bootloader. This indicates that the bootloader should be activated.

#### Returns

none.

### 7.1097.2.2 NetInit()

```
void NetInit (
 void)
```

Initializes the TCP/IP network communication interface.

#### Returns

none.

### 7.1097.2.3 NetTask()

```
void NetTask (
 void)
```

Runs the TCP/IP server task.

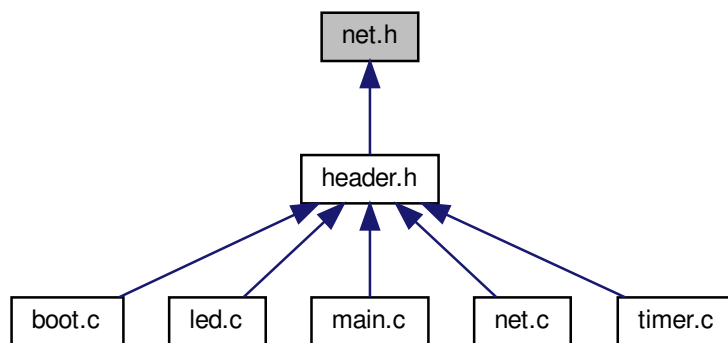
#### Returns

none.

## 7.1098 net.h File Reference

Network application for the uIP TCP/IP stack.

This graph shows which files directly or indirectly include this file:



### Data Structures

- struct [uip\\_tcp\\_appstate\\_t](#)

Define the [uip\\_tcp\\_appstate\\_t](#) datatype. This is the state of our tcp/ip application, and the memory required for this state is allocated together with each TCP connection. One application state for each TCP connection.

### Functions

- void [NetInit](#) (void)

Initializes the TCP/IP network communication interface.

- void [NetApp](#) (void)

The uIP network application that detects the XCP connect command on the port used by the bootloader. This indicates that the bootloader should be activated.

- void [NetTask](#) (void)

Runs the TCP/IP server task.

### 7.1098.1 Detailed Description

Network application for the uIP TCP/IP stack.

### 7.1098.2 Function Documentation

### 7.1098.2.1 NetApp()

```
void NetApp (
 void)
```

The uIP network application that detects the XCP connect command on the port used by the bootloader. This indicates that the bootloader should be activated.

#### Returns

none.

The uIP network application that detects the XCP connect command on the port used by the bootloader. This indicates that the bootloader should be activated.

#### Returns

none.

### 7.1098.2.2 NetInit()

```
void NetInit (
 void)
```

Initializes the TCP/IP network communication interface.

#### Returns

none.

### 7.1098.2.3 NetTask()

```
void NetTask (
 void)
```

Runs the TCP/IP server task.

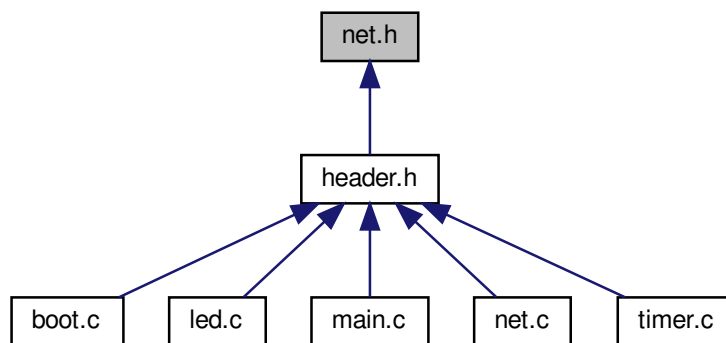
#### Returns

none.

## 7.1099 net.h File Reference

Network application for the uIP TCP/IP stack.

This graph shows which files directly or indirectly include this file:



### Data Structures

- struct [uip\\_tcp\\_appstate\\_t](#)

Define the [uip\\_tcp\\_appstate\\_t](#) datatype. This is the state of our tcp/ip application, and the memory required for this state is allocated together with each TCP connection. One application state for each TCP connection.

### Functions

- void [NetInit](#) (void)  
*Initializes the TCP/IP network communication interface.*
- void [NetApp](#) (void)  
*The uIP network application that detects the XCP connect command on the port used by the bootloader. This indicates that the bootloader should be activated.*
- void [NetTask](#) (void)  
*Runs the TCP/IP server task.*

### 7.1099.1 Detailed Description

Network application for the uIP TCP/IP stack.

### 7.1099.2 Function Documentation



### 7.1099.2.1 NetApp()

```
void NetApp (
 void)
```

The uIP network application that detects the XCP connect command on the port used by the bootloader. This indicates that the bootloader should be activated.

#### Returns

none.

The uIP network application that detects the XCP connect command on the port used by the bootloader. This indicates that the bootloader should be activated.

#### Returns

none.

### 7.1099.2.2 NetInit()

```
void NetInit (
 void)
```

Initializes the TCP/IP network communication interface.

#### Returns

none.

### 7.1099.2.3 NetTask()

```
void NetTask (
 void)
```

Runs the TCP/IP server task.

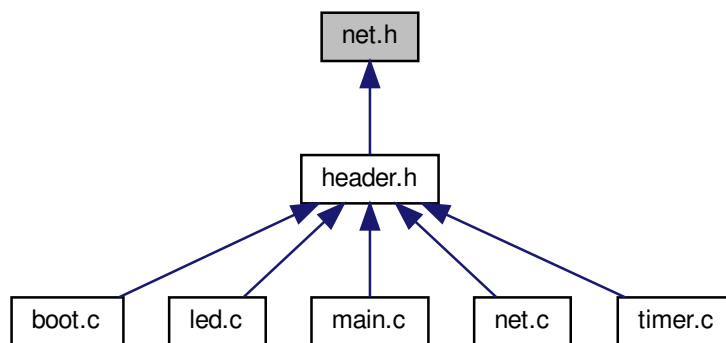
#### Returns

none.

## 7.1100 net.h File Reference

Network application for the uIP TCP/IP stack.

This graph shows which files directly or indirectly include this file:



### Data Structures

- struct [uip\\_tcp\\_appstate\\_t](#)

Define the [uip\\_tcp\\_appstate\\_t](#) datatype. This is the state of our tcp/ip application, and the memory required for this state is allocated together with each TCP connection. One application state for each TCP connection.

### Functions

- void [NetInit](#) (void)  
*Initializes the TCP/IP network communication interface.*
- void [NetApp](#) (void)  
*The uIP network application that detects the XCP connect command on the port used by the bootloader. This indicates that the bootloader should be activated.*
- void [NetTask](#) (void)  
*Runs the TCP/IP server task.*

### 7.1100.1 Detailed Description

Network application for the uIP TCP/IP stack.

### 7.1100.2 Function Documentation

### 7.1100.2.1 NetApp()

```
void NetApp (
 void)
```

The uIP network application that detects the XCP connect command on the port used by the bootloader. This indicates that the bootloader should be activated.

#### Returns

none.

The uIP network application that detects the XCP connect command on the port used by the bootloader. This indicates that the bootloader should be activated.

#### Returns

none.

### 7.1100.2.2 NetInit()

```
void NetInit (
 void)
```

Initializes the TCP/IP network communication interface.

#### Returns

none.

### 7.1100.2.3 NetTask()

```
void NetTask (
 void)
```

Runs the TCP/IP server task.

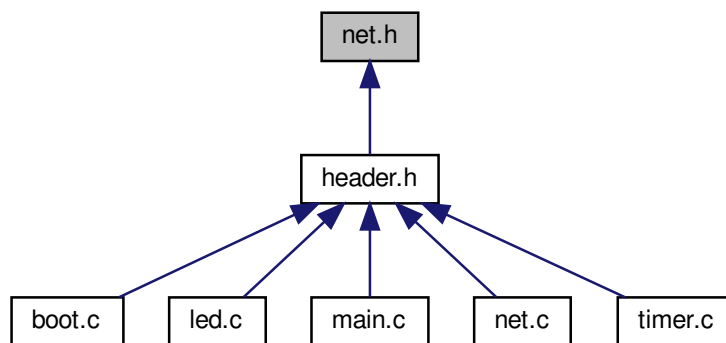
#### Returns

none.

## 7.1101 net.h File Reference

Network application for the uIP TCP/IP stack.

This graph shows which files directly or indirectly include this file:



### Data Structures

- struct [uip\\_tcp\\_appstate\\_t](#)

Define the [uip\\_tcp\\_appstate\\_t](#) datatype. This is the state of our tcp/ip application, and the memory required for this state is allocated together with each TCP connection. One application state for each TCP connection.

### Functions

- void [NetInit](#) (void)  
*Initializes the TCP/IP network communication interface.*
- void [NetApp](#) (void)  
*The uIP network application that detects the XCP connect command on the port used by the bootloader. This indicates that the bootloader should be activated.*
- void [NetTask](#) (void)  
*Runs the TCP/IP server task.*

#### 7.1101.1 Detailed Description

Network application for the uIP TCP/IP stack.

#### 7.1101.2 Function Documentation

### 7.1101.2.1 NetApp()

```
void NetApp (
 void)
```

The uIP network application that detects the XCP connect command on the port used by the bootloader. This indicates that the bootloader should be activated.

#### Returns

none.

The uIP network application that detects the XCP connect command on the port used by the bootloader. This indicates that the bootloader should be activated.

#### Returns

none.

### 7.1101.2.2 NetInit()

```
void NetInit (
 void)
```

Initializes the TCP/IP network communication interface.

#### Returns

none.

### 7.1101.2.3 NetTask()

```
void NetTask (
 void)
```

Runs the TCP/IP server task.

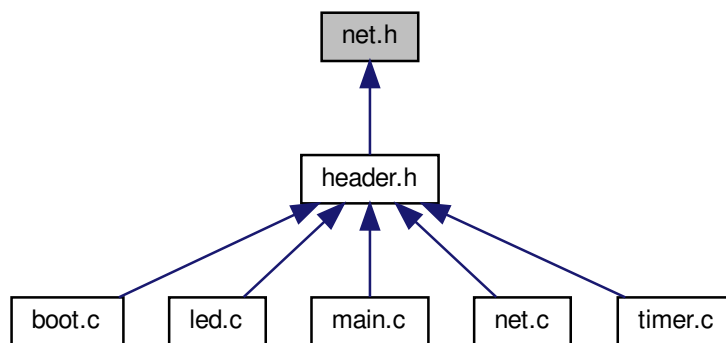
#### Returns

none.

## 7.1102 net.h File Reference

Network application for the uIP TCP/IP stack.

This graph shows which files directly or indirectly include this file:



### Data Structures

- struct [uip\\_tcp\\_appstate\\_t](#)

Define the [uip\\_tcp\\_appstate\\_t](#) datatype. This is the state of our tcp/ip application, and the memory required for this state is allocated together with each TCP connection. One application state for each TCP connection.

### Functions

- void [NetInit](#) (void)  
*Initializes the TCP/IP network communication interface.*
- void [NetApp](#) (void)  
*The uIP network application that detects the XCP connect command on the port used by the bootloader. This indicates that the bootloader should be activated.*
- void [NetTask](#) (void)  
*Runs the TCP/IP server task.*

#### 7.1102.1 Detailed Description

Network application for the uIP TCP/IP stack.

#### 7.1102.2 Function Documentation

### 7.1102.2.1 NetApp()

```
void NetApp (
 void)
```

The uIP network application that detects the XCP connect command on the port used by the bootloader. This indicates that the bootloader should be activated.

#### Returns

none.

The uIP network application that detects the XCP connect command on the port used by the bootloader. This indicates that the bootloader should be activated.

#### Returns

none.

### 7.1102.2.2 NetInit()

```
void NetInit (
 void)
```

Initializes the TCP/IP network communication interface.

#### Returns

none.

### 7.1102.2.3 NetTask()

```
void NetTask (
 void)
```

Runs the TCP/IP server task.

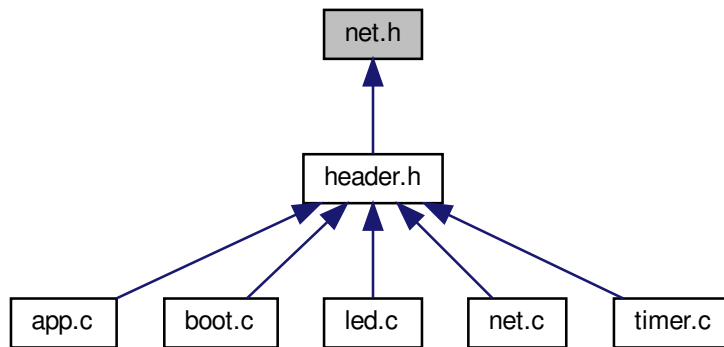
#### Returns

none.

## 7.1103 net.h File Reference

Network application for the uIP TCP/IP stack.

This graph shows which files directly or indirectly include this file:



### Data Structures

- struct [uip\\_tcp\\_appstate\\_t](#)

Define the [uip\\_tcp\\_appstate\\_t](#) datatype. This is the state of our tcp/ip application, and the memory required for this state is allocated together with each TCP connection. One application state for each TCP connection.

### Functions

- void [NetInit](#) (void)  
*Initializes the TCP/IP network communication interface.*
- void [NetApp](#) (void)  
*The uIP network application that detects the XCP connect command on the port used by the bootloader. This indicates that the bootloader should be activated.*
- void [NetTask](#) (void)  
*Runs the TCP/IP server task.*

### 7.1103.1 Detailed Description

Network application for the uIP TCP/IP stack.

### 7.1103.2 Function Documentation



### 7.1103.2.1 NetApp()

```
void NetApp (
 void)
```

The uIP network application that detects the XCP connect command on the port used by the bootloader. This indicates that the bootloader should be activated.

#### Returns

none.

The uIP network application that detects the XCP connect command on the port used by the bootloader. This indicates that the bootloader should be activated.

#### Returns

none.

### 7.1103.2.2 NetInit()

```
void NetInit (
 void)
```

Initializes the TCP/IP network communication interface.

#### Returns

none.

### 7.1103.2.3 NetTask()

```
void NetTask (
 void)
```

Runs the TCP/IP server task.

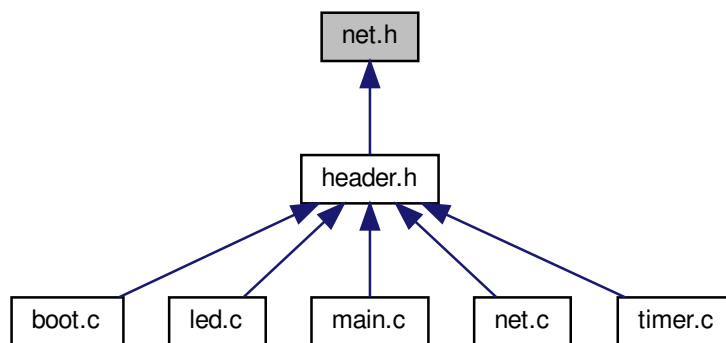
#### Returns

none.

## 7.1104 net.h File Reference

Network application for the uIP TCP/IP stack.

This graph shows which files directly or indirectly include this file:



### Data Structures

- struct [uip\\_tcp\\_appstate\\_t](#)

Define the [uip\\_tcp\\_appstate\\_t](#) datatype. This is the state of our tcp/ip application, and the memory required for this state is allocated together with each TCP connection. One application state for each TCP connection.

### Functions

- void [NetInit](#) (void)  
*Initializes the TCP/IP network communication interface.*
- void [NetApp](#) (void)  
*The uIP network application that detects the XCP connect command on the port used by the bootloader. This indicates that the bootloader should be activated.*
- void [NetTask](#) (void)  
*Runs the TCP/IP server task.*

#### 7.1104.1 Detailed Description

Network application for the uIP TCP/IP stack.

#### 7.1104.2 Function Documentation

### 7.1104.2.1 NetApp()

```
void NetApp (
 void)
```

The uIP network application that detects the XCP connect command on the port used by the bootloader. This indicates that the bootloader should be activated.

#### Returns

none.

The uIP network application that detects the XCP connect command on the port used by the bootloader. This indicates that the bootloader should be activated.

#### Returns

none.

### 7.1104.2.2 NetInit()

```
void NetInit (
 void)
```

Initializes the TCP/IP network communication interface.

#### Returns

none.

### 7.1104.2.3 NetTask()

```
void NetTask (
 void)
```

Runs the TCP/IP server task.

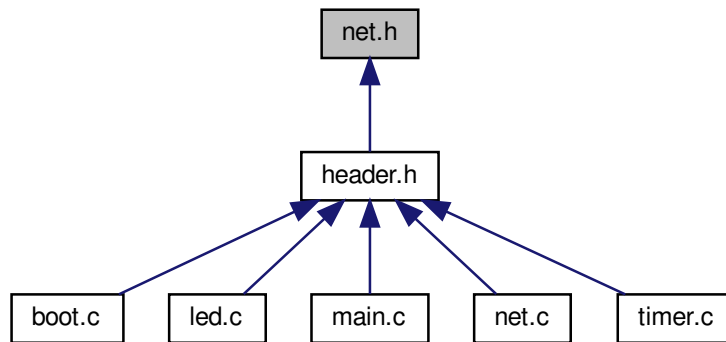
#### Returns

none.

## 7.1105 net.h File Reference

Network application for the uIP TCP/IP stack.

This graph shows which files directly or indirectly include this file:



### Data Structures

- struct [uip\\_tcp\\_appstate\\_t](#)

Define the [uip\\_tcp\\_appstate\\_t](#) datatype. This is the state of our tcp/ip application, and the memory required for this state is allocated together with each TCP connection. One application state for each TCP connection.

### Functions

- void [NetInit](#) (void)  
*Initializes the TCP/IP network communication interface.*
- void [NetApp](#) (void)  
*The uIP network application that detects the XCP connect command on the port used by the bootloader. This indicates that the bootloader should be activated.*
- void [NetTask](#) (void)  
*Runs the TCP/IP server task.*

#### 7.1105.1 Detailed Description

Network application for the uIP TCP/IP stack.

#### 7.1105.2 Function Documentation

### 7.1105.2.1 NetApp()

```
void NetApp (
 void)
```

The uIP network application that detects the XCP connect command on the port used by the bootloader. This indicates that the bootloader should be activated.

#### Returns

none.

The uIP network application that detects the XCP connect command on the port used by the bootloader. This indicates that the bootloader should be activated.

#### Returns

none.

### 7.1105.2.2 NetInit()

```
void NetInit (
 void)
```

Initializes the TCP/IP network communication interface.

#### Returns

none.

### 7.1105.2.3 NetTask()

```
void NetTask (
 void)
```

Runs the TCP/IP server task.

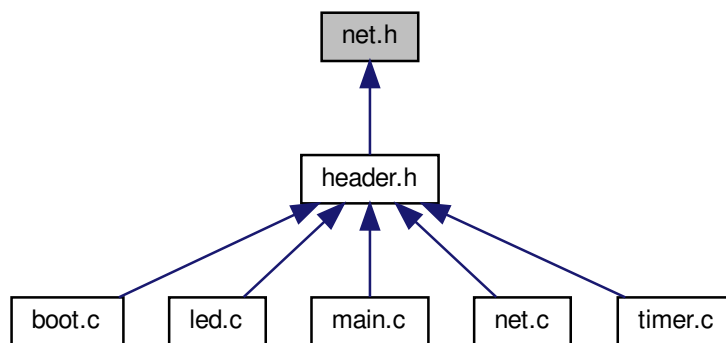
#### Returns

none.

## 7.1106 net.h File Reference

Network application for the uIP TCP/IP stack.

This graph shows which files directly or indirectly include this file:



### Data Structures

- struct [uip\\_tcp\\_appstate\\_t](#)

Define the [uip\\_tcp\\_appstate\\_t](#) datatype. This is the state of our tcp/ip application, and the memory required for this state is allocated together with each TCP connection. One application state for each TCP connection.

### Functions

- void [NetInit](#) (void)

Initializes the TCP/IP network communication interface.

- void [NetApp](#) (void)

The uIP network application that detects the XCP connect command on the port used by the bootloader. This indicates that the bootloader should be activated.

- void [NetTask](#) (void)

Runs the TCP/IP server task.

### 7.1106.1 Detailed Description

Network application for the uIP TCP/IP stack.

### 7.1106.2 Function Documentation

### 7.1106.2.1 NetApp()

```
void NetApp (
 void)
```

The uIP network application that detects the XCP connect command on the port used by the bootloader. This indicates that the bootloader should be activated.

#### Returns

none.

The uIP network application that detects the XCP connect command on the port used by the bootloader. This indicates that the bootloader should be activated.

#### Returns

none.

### 7.1106.2.2 NetInit()

```
void NetInit (
 void)
```

Initializes the TCP/IP network communication interface.

#### Returns

none.

### 7.1106.2.3 NetTask()

```
void NetTask (
 void)
```

Runs the TCP/IP server task.

#### Returns

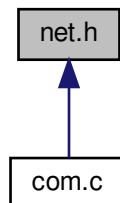
none.

Referenced by AppTask(), and main().

## 7.1107 net.h File Reference

Bootloader TCP/IP network communication interface header file.

This graph shows which files directly or indirectly include this file:



### Data Structures

- struct [uiptcp\\_appstate\\_t](#)

Define the [uiptcp\\_appstate\\_t](#) datatype. This is the state of our tcp/ip application, and the memory required for this state is allocated together with each TCP connection. One application state for each TCP connection.

### Functions

- void [NetInit](#) (void)  
*Initializes the TCP/IP network communication interface.*
- void [NetApp](#) (void)  
*The uIP network application that detects the XCP connect command on the port used by the bootloader. This indicates that the bootloader should be activated.*
- void [NetTransmitPacket](#) (blt\_int8u \*data, blt\_int8u len)  
*Transmits a packet formatted for the communication interface.*
- blt\_bool [NetReceivePacket](#) (blt\_int8u \*data, blt\_int8u \*len)  
*Receives a communication interface packet if one is present.*

### 7.1107.1 Detailed Description

Bootloader TCP/IP network communication interface header file.

### 7.1107.2 Function Documentation



### 7.1107.2.1 NetApp()

```
void NetApp (
 void)
```

The uIP network application that detects the XCP connect command on the port used by the bootloader. This indicates that the bootloader should be activated.

#### Returns

none.

The uIP network application that detects the XCP connect command on the port used by the bootloader. This indicates that the bootloader should be activated.

#### Returns

none.

### 7.1107.2.2 NetInit()

```
void NetInit (
 void)
```

Initializes the TCP/IP network communication interface.

#### Returns

none.

Referenced by ApplInit(), ComInit(), and main().

### 7.1107.2.3 NetReceivePacket()

```
blt_bool NetReceivePacket (
 blt_int8u * data,
 blt_int8u * len)
```

Receives a communication interface packet if one is present.

#### Parameters

|             |                                                         |
|-------------|---------------------------------------------------------|
| <i>data</i> | Pointer to byte array where the data is to be stored.   |
| <i>len</i>  | Pointer where the length of the packet is to be stored. |

**Returns**

BLT\_TRUE if a packet was received, BLT\_FALSE otherwise.

Referenced by ComTask().

**7.1107.2.4 NetTransmitPacket()**

```
void NetTransmitPacket (
 blt_int8u * data,
 blt_int8u len)
```

Transmits a packet formatted for the communication interface.

**Parameters**

|             |                                                            |
|-------------|------------------------------------------------------------|
| <i>data</i> | Pointer to byte array with data that it to be transmitted. |
| <i>len</i>  | Number of bytes that are to be transmitted.                |

**Returns**

none.

Referenced by ComTransmitPacket().

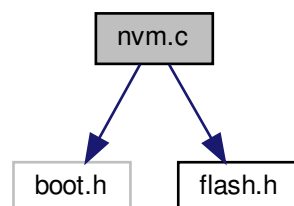
**7.1108 nvm.c File Reference**

Bootloader non-volatile memory driver source file.

```
#include "boot.h"
```

```
#include "flash.h"
```

Include dependency graph for \_template/nvm.c:



## Functions

- void [NvmInitHook](#) (void)  
*Callback that gets called at the start of the internal NVM driver initialization routine.*
- void [NvmReinitHook](#) (void)  
*Callback that gets called at the start of a firmware update to reinitialize the NVM driver.*
- [blt\\_int8u NvmWriteHook](#) ([blt\\_addr](#) addr, [blt\\_int32u](#) len, [blt\\_int8u](#) \*data)  
*Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the data hasn't been written yet.*
- [blt\\_int8u NvmEraseHook](#) ([blt\\_addr](#) addr, [blt\\_int32u](#) len)  
*Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the memory hasn't been erased yet.*
- [blt\\_bool NvmDoneHook](#) (void)  
*Callback that gets called at the end of the NVM programming session.*
- void [NvmInit](#) (void)  
*Initializes the NVM driver.*
- void [NvmReinit](#) (void)  
*Reinitializes the NVM driver. This function is called at the start of each firmware update as opposed to NvmInit, which is only called once during power on.*
- [blt\\_bool NvmWrite](#) ([blt\\_addr](#) addr, [blt\\_int32u](#) len, [blt\\_int8u](#) \*data)  
*Programs the non-volatile memory.*
- [blt\\_bool NvmErase](#) ([blt\\_addr](#) addr, [blt\\_int32u](#) len)  
*Erases the non-volatile memory.*
- [blt\\_bool NvmVerifyChecksum](#) (void)  
*Verifies the checksum, which indicates that a valid user program is present and can be started.*
- [blt\\_addr NvmGetUserProgBaseAddress](#) (void)  
*Obtains the base address of the non-volatile memory available to the user program. This is typically that start of the vector table.*
- [blt\\_bool NvmDone](#) (void)  
*Once all erase and programming operations are completed, this function is called, so at the end of the programming session and right before a software reset is performed. It is used to calculate a checksum and program this into flash. This checksum is later used to determine if a valid user program is present in flash.*

### 7.1108.1 Detailed Description

Bootloader non-volatile memory driver source file.

### 7.1108.2 Function Documentation

### 7.1108.2.1 NvmDone()

```
blt_bool NvmDone (
 void)
```

Once all erase and programming operations are completed, this function is called, so at the end of the programming session and right before a software reset is performed. It is used to calculate a checksum and program this into flash. This checksum is later used to determine if a valid user program is present in flash.

#### Returns

BLT\_TRUE if successful, BLT\_FALSE otherwise.

Referenced by FileTask(), and XcpCmdProgram().

### 7.1108.2.2 NvmDoneHook()

```
blt_bool NvmDoneHook (
 void)
```

Callback that gets called at the end of the NVM programming session.

#### Returns

BLT\_TRUE is successful, BLT\_FALSE otherwise.

Referenced by NvmDone().

### 7.1108.2.3 NvmErase()

```
blt_bool NvmErase (
 blt_addr addr,
 blt_int32u len)
```

Erases the non-volatile memory.

#### Parameters

|             |                  |
|-------------|------------------|
| <i>addr</i> | Start address.   |
| <i>len</i>  | Length in bytes. |

#### Returns

BLT\_TRUE if successful, BLT\_FALSE otherwise.

Referenced by FileTask(), and XcpCmdProgramClear().

#### 7.1108.2.4 NvmEraseHook()

```
blt_int8u NvmEraseHook (
 blt_addr addr,
 blt_int32u len)
```

Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the memory hasn't been erased yet.

##### Parameters

|             |                  |
|-------------|------------------|
| <i>addr</i> | Start address.   |
| <i>len</i>  | Length in bytes. |

##### Returns

BLT\_NVM\_OKAY if successful, BLT\_NVM\_NOT\_IN\_RANGE if the address is not within the supported memory range, or BLT\_NVM\_ERROR if the erase operation failed.

Referenced by NvmErase().

#### 7.1108.2.5 NvmGetUserProgBaseAddress()

```
blt_addr NvmGetUserProgBaseAddress (
 void)
```

Obtains the base address of the non-volatile memory available to the user program. This is typically that start of the vector table.

##### Returns

Base address.

Referenced by CpuStartUserProgram().

#### 7.1108.2.6 NvmInit()

```
void NvmInit (
 void)
```

Initializes the NVM driver.

##### Returns

none.

Referenced by BootInit(), FileTask(), and XcpCmdConnect().

### 7.1108.2.7 NvmInitHook()

```
void NvmInitHook (
 void)
```

Callback that gets called at the start of the internal NVM driver initialization routine.

#### Returns

none.

Referenced by NvmInit().

### 7.1108.2.8 NvmReinit()

```
void NvmReinit (
 void)
```

Reinitializes the NVM driver. This function is called at the start of each firmware update as opposed to NvmInit, which is only called once during power on.

#### Returns

none.

### 7.1108.2.9 NvmReinitHook()

```
void NvmReinitHook (
 void)
```

Callback that gets called at the start of a firmware update to reinitialize the NVM driver.

#### Returns

none.

Referenced by NvmReinit().

### 7.1108.2.10 NvmVerifyChecksum()

```
blt_bool NvmVerifyChecksum (
 void)
```

Verifies the checksum, which indicates that a valid user program is present and can be started.

#### Returns

BLT\_TRUE if successful, BLT\_FALSE otherwise.

Referenced by CpuStartUserProgram().

### 7.1108.2.11 NvmWrite()

```
blt_bool NvmWrite (
 blt_addr addr,
 blt_int32u len,
 blt_int8u * data)
```

Programs the non-volatile memory.

#### Parameters

|             |                             |
|-------------|-----------------------------|
| <i>addr</i> | Start address.              |
| <i>len</i>  | Length in bytes.            |
| <i>data</i> | Pointer to the data buffer. |

#### Returns

BLT\_TRUE if successful, BLT\_FALSE otherwise.

Referenced by FileTask(), XcpCmdProgram(), and XcpCmdProgramMax().

### 7.1108.2.12 NvmWriteHook()

```
blt_int8u NvmWriteHook (
 blt_addr addr,
 blt_int32u len,
 blt_int8u * data)
```

Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the data hasn't been written yet.

## Parameters

|             |                             |
|-------------|-----------------------------|
| <i>addr</i> | Start address.              |
| <i>len</i>  | Length in bytes.            |
| <i>data</i> | Pointer to the data buffer. |

## Returns

BLT\_NVM\_OKAY if successful, BLT\_NVM\_NOT\_IN\_RANGE if the address is not within the supported memory range, or BLT\_NVM\_ERROR if the write operation failed.

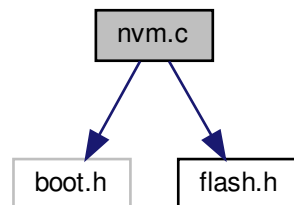
Referenced by NvmWrite().

## 7.1109 nvm.c File Reference

Bootloader non-volatile memory driver source file.

```
#include "boot.h"
#include "flash.h"
```

Include dependency graph for ARMCM0\_S32K11/nvm.c:



## Functions

- void [NvmInitHook](#) (void)  
*Callback that gets called at the start of the internal NVM driver initialization routine.*
- void [NvmReinitHook](#) (void)  
*Callback that gets called at the start of a firmware update to reinitialize the NVM driver.*
- [blt\\_int8u NvmWriteHook](#) ([blt\\_addr](#) addr, [blt\\_int32u](#) len, [blt\\_int8u](#) \*data)  
*Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the data hasn't been written yet.*
- [blt\\_int8u NvmEraseHook](#) ([blt\\_addr](#) addr, [blt\\_int32u](#) len)  
*Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the memory hasn't been erased yet.*
- [blt\\_bool NvmDoneHook](#) (void)



*Callback that gets called at the end of the NVM programming session.*

- void [NvmInit](#) (void)

*Initializes the NVM driver.*

- void [NvmReinit](#) (void)

*Reinitializes the NVM driver. This function is called at the start of each firmware update as opposed to NvmInit, which is only called once during power on.*

- [blt\\_bool](#) [NvmWrite](#) ([blt\\_addr](#) addr, [blt\\_int32u](#) len, [blt\\_int8u](#) \*data)

*Programs the non-volatile memory.*

- [blt\\_bool](#) [NvmErase](#) ([blt\\_addr](#) addr, [blt\\_int32u](#) len)

*Erases the non-volatile memory.*

- [blt\\_bool](#) [NvmVerifyChecksum](#) (void)

*Verifies the checksum, which indicates that a valid user program is present and can be started.*

- [blt\\_addr](#) [NvmGetUserProgBaseAddress](#) (void)

*Obtains the base address of the non-volatile memory available to the user program. This is typically that start of the vector table.*

- [blt\\_bool](#) [NvmDone](#) (void)

*Once all erase and programming operations are completed, this function is called, so at the end of the programming session and right before a software reset is performed. It is used to calculate a checksum and program this into flash. This checksum is later used to determine if a valid user program is present in flash.*

## 7.1109.1 Detailed Description

Bootloader non-volatile memory driver source file.

## 7.1109.2 Function Documentation

### 7.1109.2.1 NvmDone()

```
blt_bool NvmDone (
 void)
```

Once all erase and programming operations are completed, this function is called, so at the end of the programming session and right before a software reset is performed. It is used to calculate a checksum and program this into flash. This checksum is later used to determine if a valid user program is present in flash.

#### Returns

BLT\_TRUE if successful, BLT\_FALSE otherwise.

### 7.1109.2.2 NvmDoneHook()

```
blt_bool NvmDoneHook (
 void)
```

Callback that gets called at the end of the NVM programming session.

#### Returns

BLT\_TRUE is successful, BLT\_FALSE otherwise.

Referenced by NvmDone().

### 7.1109.2.3 NvmErase()

```
blt_bool NvmErase (
 blt_addr addr,
 blt_int32u len)
```

Erases the non-volatile memory.

#### Parameters

|             |                  |
|-------------|------------------|
| <i>addr</i> | Start address.   |
| <i>len</i>  | Length in bytes. |

#### Returns

BLT\_TRUE if successful, BLT\_FALSE otherwise.

### 7.1109.2.4 NvmEraseHook()

```
blt_int8u NvmEraseHook (
 blt_addr addr,
 blt_int32u len)
```

Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the memory hasn't been erased yet.

#### Parameters

|             |                  |
|-------------|------------------|
| <i>addr</i> | Start address.   |
| <i>len</i>  | Length in bytes. |

**Returns**

BLT\_NVM\_OKAY if successful, BLT\_NVM\_NOT\_IN\_RANGE if the address is not within the supported memory range, or BLT\_NVM\_ERROR is the erase operation failed.

Referenced by NvmErase().

**7.1109.2.5 NvmGetUserProgBaseAddress()**

```
blt_addr NvmGetUserProgBaseAddress (
 void)
```

Obtains the base address of the non-volatile memory available to the user program. This is typically that start of the vector table.

**Returns**

Base address.

**7.1109.2.6 NvmInit()**

```
void NvmInit (
 void)
```

Initializes the NVM driver.

**Returns**

none.

**7.1109.2.7 NvmInitHook()**

```
void NvmInitHook (
 void)
```

Callback that gets called at the start of the internal NVM driver initialization routine.

**Returns**

none.

Referenced by NvmInit().

#### 7.1109.2.8 NvmReinit()

```
void NvmReinit (
 void)
```

Reinitializes the NVM driver. This function is called at the start of each firmware update as opposed to NvmInit, which is only called once during power on.

##### Returns

none.

#### 7.1109.2.9 NvmReinitHook()

```
void NvmReinitHook (
 void)
```

Callback that gets called at the start of a firmware update to reinitialize the NVM driver.

##### Returns

none.

Referenced by NvmReinit().

#### 7.1109.2.10 NvmVerifyChecksum()

```
blt_bool NvmVerifyChecksum (
 void)
```

Verifies the checksum, which indicates that a valid user program is present and can be started.

##### Returns

BLT\_TRUE if successful, BLT\_FALSE otherwise.

#### 7.1109.2.11 NvmWrite()

```
blt_bool NvmWrite (
 blt_addr addr,
 blt_int32u len,
 blt_int8u * data)
```

Programs the non-volatile memory.

## Parameters

|             |                             |
|-------------|-----------------------------|
| <i>addr</i> | Start address.              |
| <i>len</i>  | Length in bytes.            |
| <i>data</i> | Pointer to the data buffer. |

## Returns

BLT\_TRUE if successful, BLT\_FALSE otherwise.

## 7.1109.2.12 NvmWriteHook()

```
blt_int8u NvmWriteHook (
 blt_addr addr,
 blt_int32u len,
 blt_int8u * data)
```

Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the data hasn't been written yet.

## Parameters

|             |                             |
|-------------|-----------------------------|
| <i>addr</i> | Start address.              |
| <i>len</i>  | Length in bytes.            |
| <i>data</i> | Pointer to the data buffer. |

## Returns

BLT\_NVM\_OKAY if successful, BLT\_NVM\_NOT\_IN\_RANGE if the address is not within the supported memory range, or BLT\_NVM\_ERROR if the write operation failed.

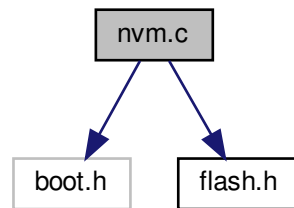
Referenced by NvmWrite().

## 7.1110 nvm.c File Reference

Bootloader non-volatile memory driver source file.

```
#include "boot.h"
#include "flash.h"
```

Include dependency graph for ARMCM0\_STM32F0/nvm.c:



## Functions

- void [NvmInitHook](#) (void)  
*Callback that gets called at the start of the internal NVM driver initialization routine.*
- void [NvmReinitHook](#) (void)  
*Callback that gets called at the start of a firmware update to reinitialize the NVM driver.*
- [blt\\_int8u NvmWriteHook](#) ([blt\\_addr](#) addr, [blt\\_int32u](#) len, [blt\\_int8u](#) \*data)  
*Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the data hasn't been written yet.*
- [blt\\_int8u NvmEraseHook](#) ([blt\\_addr](#) addr, [blt\\_int32u](#) len)  
*Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the memory hasn't been erased yet.*
- [blt\\_bool NvmDoneHook](#) (void)  
*Callback that gets called at the end of the NVM programming session.*
- void [NvmInit](#) (void)  
*Initializes the NVM driver.*
- void [NvmReinit](#) (void)  
*Reinitializes the NVM driver. This function is called at the start of each firmware update as opposed to NvmInit, which is only called once during power on.*
- [blt\\_bool NvmWrite](#) ([blt\\_addr](#) addr, [blt\\_int32u](#) len, [blt\\_int8u](#) \*data)  
*Programs the non-volatile memory.*
- [blt\\_bool NvmErase](#) ([blt\\_addr](#) addr, [blt\\_int32u](#) len)  
*Erases the non-volatile memory.*
- [blt\\_bool NvmVerifyChecksum](#) (void)  
*Verifies the checksum, which indicates that a valid user program is present and can be started.*
- [blt\\_addr NvmGetUserProgBaseAddress](#) (void)  
*Obtains the base address of the non-volatile memory available to the user program. This is typically that start of the vector table.*
- [blt\\_bool NvmDone](#) (void)  
*Once all erase and programming operations are completed, this function is called, so at the end of the programming session and right before a software reset is performed. It is used to calculate a checksum and program this into flash. This checksum is later used to determine if a valid user program is present in flash.*

## 7.1110.1 Detailed Description

Bootloader non-volatile memory driver source file.

## 7.1110.2 Function Documentation

### 7.1110.2.1 NvmDone()

```
blt_bool NvmDone (
 void)
```

Once all erase and programming operations are completed, this function is called, so at the end of the programming session and right before a software reset is performed. It is used to calculate a checksum and program this into flash. This checksum is later used to determine if a valid user program is present in flash.

#### Returns

BLT\_TRUE if successful, BLT\_FALSE otherwise.

### 7.1110.2.2 NvmDoneHook()

```
blt_bool NvmDoneHook (
 void)
```

Callback that gets called at the end of the NVM programming session.

#### Returns

BLT\_TRUE is successful, BLT\_FALSE otherwise.

Referenced by NvmDone().

### 7.1110.2.3 NvmErase()

```
blt_bool NvmErase (
 blt_addr addr,
 blt_int32u len)
```

Erases the non-volatile memory.

**Parameters**

|             |                  |
|-------------|------------------|
| <i>addr</i> | Start address.   |
| <i>len</i>  | Length in bytes. |

**Returns**

BLT\_TRUE if successful, BLT\_FALSE otherwise.

**7.1110.2.4 NvmEraseHook()**

```
blt_int8u NvmEraseHook (
 blt_addr addr,
 blt_int32u len)
```

Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the memory hasn't been erased yet.

**Parameters**

|             |                  |
|-------------|------------------|
| <i>addr</i> | Start address.   |
| <i>len</i>  | Length in bytes. |

**Returns**

BLT\_NVM\_OKAY if successful, BLT\_NVM\_NOT\_IN\_RANGE if the address is not within the supported memory range, or BLT\_NVM\_ERROR is the erase operation failed.

Referenced by NvmErase().

**7.1110.2.5 NvmGetUserProgBaseAddress()**

```
blt_addr NvmGetUserProgBaseAddress (
 void)
```

Obtains the base address of the non-volatile memory available to the user program. This is typically that start of the vector table.

**Returns**

Base address.



#### 7.1110.2.6 NvmInit()

```
void NvmInit (
 void)
```

Initializes the NVM driver.

##### Returns

none.

#### 7.1110.2.7 NvmInitHook()

```
void NvmInitHook (
 void)
```

Callback that gets called at the start of the internal NVM driver initialization routine.

##### Returns

none.

Referenced by NvmInit().

#### 7.1110.2.8 NvmReinit()

```
void NvmReinit (
 void)
```

Reinitializes the NVM driver. This function is called at the start of each firmware update as opposed to NvmInit, which is only called once during power on.

##### Returns

none.

#### 7.1110.2.9 NvmReinitHook()

```
void NvmReinitHook (
 void)
```

Callback that gets called at the start of a firmware update to reinitialize the NVM driver.

##### Returns

none.

Referenced by NvmReinit().

### 7.1110.2.10 NvmVerifyChecksum()

```
blt_bool NvmVerifyChecksum (
 void)
```

Verifies the checksum, which indicates that a valid user program is present and can be started.

#### Returns

BLT\_TRUE if successful, BLT\_FALSE otherwise.

### 7.1110.2.11 NvmWrite()

```
blt_bool NvmWrite (
 blt_addr addr,
 blt_int32u len,
 blt_int8u * data)
```

Programs the non-volatile memory.

#### Parameters

|             |                             |
|-------------|-----------------------------|
| <i>addr</i> | Start address.              |
| <i>len</i>  | Length in bytes.            |
| <i>data</i> | Pointer to the data buffer. |

#### Returns

BLT\_TRUE if successful, BLT\_FALSE otherwise.

### 7.1110.2.12 NvmWriteHook()

```
blt_int8u NvmWriteHook (
 blt_addr addr,
 blt_int32u len,
 blt_int8u * data)
```

Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the data hasn't been written yet.

#### Parameters

|             |                             |
|-------------|-----------------------------|
| <i>addr</i> | Start address.              |
| <i>len</i>  | Length in bytes.            |
| <i>data</i> | Pointer to the data buffer. |

**Returns**

BLT\_NVM\_OKAY if successful, BLT\_NVM\_NOT\_IN\_RANGE if the address is not within the supported memory range, or BLT\_NVM\_ERROR if the write operation failed.

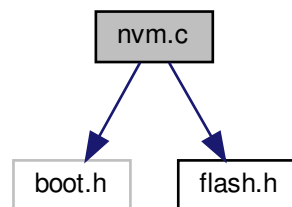
Referenced by NvmWrite().

**7.1111 nvm.c File Reference**

Bootloader non-volatile memory driver source file.

```
#include "boot.h"
#include "flash.h"
```

Include dependency graph for ARMCM0\_STM32G0/nvm.c:

**Functions**

- void [NvmInitHook](#) (void)  
*Callback that gets called at the start of the internal NVM driver initialization routine.*
- void [NvmReinitHook](#) (void)  
*Callback that gets called at the start of a firmware update to reinitialize the NVM driver.*
- [blt\\_int8u NvmWriteHook](#) ([blt\\_addr](#) addr, [blt\\_int32u](#) len, [blt\\_int8u](#) \*data)  
*Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the data hasn't been written yet.*
- [blt\\_int8u NvmEraseHook](#) ([blt\\_addr](#) addr, [blt\\_int32u](#) len)  
*Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the memory hasn't been erased yet.*
- [blt\\_bool NvmDoneHook](#) (void)  
*Callback that gets called at the end of the NVM programming session.*
- void [NvmInit](#) (void)  
*Initializes the NVM driver.*
- void [NvmReinit](#) (void)  
*Reinitializes the NVM driver. This function is called at the start of each firmware update as opposed to NvmInit, which is only called once during power on.*
- [blt\\_bool NvmWrite](#) ([blt\\_addr](#) addr, [blt\\_int32u](#) len, [blt\\_int8u](#) \*data)

- Programs the non-volatile memory.*
- [blt\\_bool NvmErase](#) ([blt\\_addr](#) addr, [blt\\_int32u](#) len)
- Erases the non-volatile memory.*
- [blt\\_bool NvmVerifyChecksum](#) (void)
- Verifies the checksum, which indicates that a valid user program is present and can be started.*
- [blt\\_addr NvmGetUserProgBaseAddress](#) (void)
- Obtains the base address of the non-volatile memory available to the user program. This is typically that start of the vector table.*
- [blt\\_bool NvmDone](#) (void)
- Once all erase and programming operations are completed, this function is called, so at the end of the programming session and right before a software reset is performed. It is used to calculate a checksum and program this into flash. This checksum is later used to determine if a valid user program is present in flash.*

### 7.1111.1 Detailed Description

Bootloader non-volatile memory driver source file.

### 7.1111.2 Function Documentation

#### 7.1111.2.1 NvmDone()

```
blt_bool NvmDone (
 void)
```

Once all erase and programming operations are completed, this function is called, so at the end of the programming session and right before a software reset is performed. It is used to calculate a checksum and program this into flash. This checksum is later used to determine if a valid user program is present in flash.

##### Returns

BLT\_TRUE if successful, BLT\_FALSE otherwise.

#### 7.1111.2.2 NvmDoneHook()

```
blt_bool NvmDoneHook (
 void)
```

Callback that gets called at the end of the NVM programming session.

##### Returns

BLT\_TRUE is successful, BLT\_FALSE otherwise.

Referenced by [NvmDone\(\)](#).

#### 7.1111.2.3 NvmErase()

```
blt_bool NvmErase (
 blt_addr addr,
 blt_int32u len)
```

Erases the non-volatile memory.

**Parameters**

|             |                  |
|-------------|------------------|
| <i>addr</i> | Start address.   |
| <i>len</i>  | Length in bytes. |

**Returns**

BLT\_TRUE if successful, BLT\_FALSE otherwise.

**7.1111.2.4 NvmEraseHook()**

```
blt_int8u NvmEraseHook (
 blt_addr addr,
 blt_int32u len)
```

Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the memory hasn't been erased yet.

**Parameters**

|             |                  |
|-------------|------------------|
| <i>addr</i> | Start address.   |
| <i>len</i>  | Length in bytes. |

**Returns**

BLT\_NVM\_OKAY if successful, BLT\_NVM\_NOT\_IN\_RANGE if the address is not within the supported memory range, or BLT\_NVM\_ERROR is the erase operation failed.

Referenced by NvmErase().

**7.1111.2.5 NvmGetUserProgBaseAddress()**

```
blt_addr NvmGetUserProgBaseAddress (
 void)
```

Obtains the base address of the non-volatile memory available to the user program. This is typically that start of the vector table.

**Returns**

Base address.

#### 7.1111.2.6 NvmInit()

```
void NvmInit (
 void)
```

Initializes the NVM driver.

##### Returns

none.

#### 7.1111.2.7 NvmInitHook()

```
void NvmInitHook (
 void)
```

Callback that gets called at the start of the internal NVM driver initialization routine.

##### Returns

none.

Referenced by NvmInit().

#### 7.1111.2.8 NvmReinit()

```
void NvmReinit (
 void)
```

Reinitializes the NVM driver. This function is called at the start of each firmware update as opposed to NvmInit, which is only called once during power on.

##### Returns

none.

#### 7.1111.2.9 NvmReinitHook()

```
void NvmReinitHook (
 void)
```

Callback that gets called at the start of a firmware update to reinitialize the NVM driver.

##### Returns

none.

Referenced by NvmReinit().

### 7.1111.2.10 NvmVerifyChecksum()

```
blt_bool NvmVerifyChecksum (
 void)
```

Verifies the checksum, which indicates that a valid user program is present and can be started.

#### Returns

BLT\_TRUE if successful, BLT\_FALSE otherwise.

### 7.1111.2.11 NvmWrite()

```
blt_bool NvmWrite (
 blt_addr addr,
 blt_int32u len,
 blt_int8u * data)
```

Programs the non-volatile memory.

#### Parameters

|             |                             |
|-------------|-----------------------------|
| <i>addr</i> | Start address.              |
| <i>len</i>  | Length in bytes.            |
| <i>data</i> | Pointer to the data buffer. |

#### Returns

BLT\_TRUE if successful, BLT\_FALSE otherwise.

### 7.1111.2.12 NvmWriteHook()

```
blt_int8u NvmWriteHook (
 blt_addr addr,
 blt_int32u len,
 blt_int8u * data)
```

Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the data hasn't been written yet.

#### Parameters

|             |                             |
|-------------|-----------------------------|
| <i>addr</i> | Start address.              |
| <i>len</i>  | Length in bytes.            |
| <i>data</i> | Pointer to the data buffer. |

### Returns

BLT\_NVM\_OKAY if successful, BLT\_NVM\_NOT\_IN\_RANGE if the address is not within the supported memory range, or BLT\_NVM\_ERROR if the write operation failed.

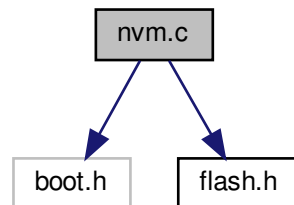
Referenced by NvmWrite().

## 7.1112 nvm.c File Reference

Bootloader non-volatile memory driver source file.

```
#include "boot.h"
#include "flash.h"
```

Include dependency graph for ARMCM0\_XMC1/nvm.c:



### Functions

- void [NvmInitHook](#) (void)  
*Callback that gets called at the start of the internal NVM driver initialization routine.*
- void [NvmReinitHook](#) (void)  
*Callback that gets called at the start of a firmware update to reinitialize the NVM driver.*
- [blt\\_int8u NvmWriteHook](#) ([blt\\_addr](#) addr, [blt\\_int32u](#) len, [blt\\_int8u](#) \*data)  
*Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the data hasn't been written yet.*
- [blt\\_int8u NvmEraseHook](#) ([blt\\_addr](#) addr, [blt\\_int32u](#) len)  
*Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the memory hasn't been erased yet.*
- [blt\\_bool NvmDoneHook](#) (void)  
*Callback that gets called at the end of the NVM programming session.*
- void [NvmInit](#) (void)  
*Initializes the NVM driver.*
- void [NvmReinit](#) (void)  
*Reinitializes the NVM driver. This function is called at the start of each firmware update as opposed to NvmInit, which is only called once during power on.*
- [blt\\_bool NvmWrite](#) ([blt\\_addr](#) addr, [blt\\_int32u](#) len, [blt\\_int8u](#) \*data)



- Programs the non-volatile memory.*
- [blt\\_bool NvmErase](#) ([blt\\_addr](#) addr, [blt\\_int32u](#) len)
- Erases the non-volatile memory.*
- [blt\\_bool NvmVerifyChecksum](#) (void)
- Verifies the checksum, which indicates that a valid user program is present and can be started.*
- [blt\\_addr NvmGetUserProgBaseAddress](#) (void)
- Obtains the base address of the non-volatile memory available to the user program. This is typically that start of the vector table.*
- [blt\\_bool NvmDone](#) (void)
- Once all erase and programming operations are completed, this function is called, so at the end of the programming session and right before a software reset is performed. It is used to calculate a checksum and program this into flash. This checksum is later used to determine if a valid user program is present in flash.*

### 7.1112.1 Detailed Description

Bootloader non-volatile memory driver source file.

### 7.1112.2 Function Documentation

#### 7.1112.2.1 NvmDone()

```
blt_bool NvmDone (
 void)
```

Once all erase and programming operations are completed, this function is called, so at the end of the programming session and right before a software reset is performed. It is used to calculate a checksum and program this into flash. This checksum is later used to determine if a valid user program is present in flash.

##### Returns

BLT\_TRUE if successful, BLT\_FALSE otherwise.

#### 7.1112.2.2 NvmDoneHook()

```
blt_bool NvmDoneHook (
 void)
```

Callback that gets called at the end of the NVM programming session.

##### Returns

BLT\_TRUE is successful, BLT\_FALSE otherwise.

Referenced by [NvmDone\(\)](#).

#### 7.1112.2.3 NvmErase()

```
blt_bool NvmErase (
 blt_addr addr,
 blt_int32u len)
```

Erases the non-volatile memory.

**Parameters**

|             |                  |
|-------------|------------------|
| <i>addr</i> | Start address.   |
| <i>len</i>  | Length in bytes. |

**Returns**

BLT\_TRUE if successful, BLT\_FALSE otherwise.

**7.1112.2.4 NvmEraseHook()**

```
blt_int8u NvmEraseHook (
 blt_addr addr,
 blt_int32u len)
```

Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the memory hasn't been erased yet.

**Parameters**

|             |                  |
|-------------|------------------|
| <i>addr</i> | Start address.   |
| <i>len</i>  | Length in bytes. |

**Returns**

BLT\_NVM\_OKAY if successful, BLT\_NVM\_NOT\_IN\_RANGE if the address is not within the supported memory range, or BLT\_NVM\_ERROR is the erase operation failed.

Referenced by NvmErase().

**7.1112.2.5 NvmGetUserProgBaseAddress()**

```
blt_addr NvmGetUserProgBaseAddress (
 void)
```

Obtains the base address of the non-volatile memory available to the user program. This is typically that start of the vector table.

**Returns**

Base address.

#### 7.1112.2.6 NvmInit()

```
void NvmInit (
 void)
```

Initializes the NVM driver.

##### Returns

none.

#### 7.1112.2.7 NvmInitHook()

```
void NvmInitHook (
 void)
```

Callback that gets called at the start of the internal NVM driver initialization routine.

##### Returns

none.

Referenced by NvmInit().

#### 7.1112.2.8 NvmReinit()

```
void NvmReinit (
 void)
```

Reinitializes the NVM driver. This function is called at the start of each firmware update as opposed to NvmInit, which is only called once during power on.

##### Returns

none.

#### 7.1112.2.9 NvmReinitHook()

```
void NvmReinitHook (
 void)
```

Callback that gets called at the start of a firmware update to reinitialize the NVM driver.

##### Returns

none.

Referenced by NvmReinit().

#### 7.1112.2.10 NvmVerifyChecksum()

```
blt_bool NvmVerifyChecksum (
 void)
```

Verifies the checksum, which indicates that a valid user program is present and can be started.

##### Returns

BLT\_TRUE if successful, BLT\_FALSE otherwise.

#### 7.1112.2.11 NvmWrite()

```
blt_bool NvmWrite (
 blt_addr addr,
 blt_int32u len,
 blt_int8u * data)
```

Programs the non-volatile memory.

##### Parameters

|             |                             |
|-------------|-----------------------------|
| <i>addr</i> | Start address.              |
| <i>len</i>  | Length in bytes.            |
| <i>data</i> | Pointer to the data buffer. |

##### Returns

BLT\_TRUE if successful, BLT\_FALSE otherwise.

#### 7.1112.2.12 NvmWriteHook()

```
blt_int8u NvmWriteHook (
 blt_addr addr,
 blt_int32u len,
 blt_int8u * data)
```

Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the data hasn't been written yet.

##### Parameters

|             |                             |
|-------------|-----------------------------|
| <i>addr</i> | Start address.              |
| <i>len</i>  | Length in bytes.            |
| <i>data</i> | Pointer to the data buffer. |

## Returns

BLT\_NVM\_OKAY if successful, BLT\_NVM\_NOT\_IN\_RANGE if the address is not within the supported memory range, or BLT\_NVM\_ERROR if the write operation failed.

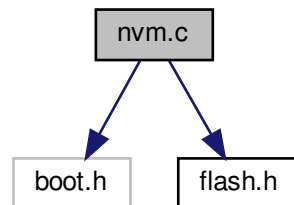
Referenced by NvmWrite().

## 7.1113 nvm.c File Reference

Bootloader non-volatile memory driver source file.

```
#include "boot.h"
#include "flash.h"
```

Include dependency graph for ARMCM33\_STM32L5/nvm.c:



## Functions

- void [NvmInitHook](#) (void)  
*Callback that gets called at the start of the internal NVM driver initialization routine.*
- void [NvmReinitHook](#) (void)  
*Callback that gets called at the start of a firmware update to reinitialize the NVM driver.*
- [blt\\_int8u NvmWriteHook](#) ([blt\\_addr](#) addr, [blt\\_int32u](#) len, [blt\\_int8u](#) \*data)  
*Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the data hasn't been written yet.*
- [blt\\_int8u NvmEraseHook](#) ([blt\\_addr](#) addr, [blt\\_int32u](#) len)  
*Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the memory hasn't been erased yet.*
- [blt\\_bool NvmDoneHook](#) (void)  
*Callback that gets called at the end of the NVM programming session.*
- void [NvmInit](#) (void)  
*Initializes the NVM driver.*
- void [NvmReinit](#) (void)  
*Reinitializes the NVM driver. This function is called at the start of each firmware update as opposed to NvmInit, which is only called once during power on.*
- [blt\\_bool NvmWrite](#) ([blt\\_addr](#) addr, [blt\\_int32u](#) len, [blt\\_int8u](#) \*data)

*Programs the non-volatile memory.*

- [blt\\_bool NvmErase](#) ([blt\\_addr](#) addr, [blt\\_int32u](#) len)

*Erases the non-volatile memory.*

- [blt\\_bool NvmVerifyChecksum](#) (void)

*Verifies the checksum, which indicates that a valid user program is present and can be started.*

- [blt\\_addr NvmGetUserProgBaseAddress](#) (void)

*Obtains the base address of the non-volatile memory available to the user program. This is typically that start of the vector table.*

- [blt\\_bool NvmDone](#) (void)

*Once all erase and programming operations are completed, this function is called, so at the end of the programming session and right before a software reset is performed. It is used to calculate a checksum and program this into flash. This checksum is later used to determine if a valid user program is present in flash.*

### 7.1113.1 Detailed Description

Bootloader non-volatile memory driver source file.

### 7.1113.2 Function Documentation

#### 7.1113.2.1 NvmDone()

```
blt_bool NvmDone (
 void)
```

Once all erase and programming operations are completed, this function is called, so at the end of the programming session and right before a software reset is performed. It is used to calculate a checksum and program this into flash. This checksum is later used to determine if a valid user program is present in flash.

##### Returns

BLT\_TRUE if successful, BLT\_FALSE otherwise.

#### 7.1113.2.2 NvmDoneHook()

```
blt_bool NvmDoneHook (
 void)
```

Callback that gets called at the end of the NVM programming session.

##### Returns

BLT\_TRUE is successful, BLT\_FALSE otherwise.

Referenced by NvmDone().

#### 7.1113.2.3 NvmErase()

```
blt_bool NvmErase (
 blt_addr addr,
 blt_int32u len)
```

Erases the non-volatile memory.

**Parameters**

|             |                  |
|-------------|------------------|
| <i>addr</i> | Start address.   |
| <i>len</i>  | Length in bytes. |

**Returns**

BLT\_TRUE if successful, BLT\_FALSE otherwise.

**7.1113.2.4 NvmEraseHook()**

```
blt_int8u NvmEraseHook (
 blt_addr addr,
 blt_int32u len)
```

Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the memory hasn't been erased yet.

**Parameters**

|             |                  |
|-------------|------------------|
| <i>addr</i> | Start address.   |
| <i>len</i>  | Length in bytes. |

**Returns**

BLT\_NVM\_OKAY if successful, BLT\_NVM\_NOT\_IN\_RANGE if the address is not within the supported memory range, or BLT\_NVM\_ERROR is the erase operation failed.

Referenced by NvmErase().

**7.1113.2.5 NvmGetUserProgBaseAddress()**

```
blt_addr NvmGetUserProgBaseAddress (
 void)
```

Obtains the base address of the non-volatile memory available to the user program. This is typically that start of the vector table.

**Returns**

Base address.

#### 7.1113.2.6 NvmInit()

```
void NvmInit (
 void)
```

Initializes the NVM driver.

##### Returns

none.

#### 7.1113.2.7 NvmInitHook()

```
void NvmInitHook (
 void)
```

Callback that gets called at the start of the internal NVM driver initialization routine.

##### Returns

none.

Referenced by NvmInit().

#### 7.1113.2.8 NvmReinit()

```
void NvmReinit (
 void)
```

Reinitializes the NVM driver. This function is called at the start of each firmware update as opposed to NvmInit, which is only called once during power on.

##### Returns

none.

#### 7.1113.2.9 NvmReinitHook()

```
void NvmReinitHook (
 void)
```

Callback that gets called at the start of a firmware update to reinitialize the NVM driver.

##### Returns

none.

Referenced by NvmReinit().



### 7.1113.2.10 NvmVerifyChecksum()

```
blt_bool NvmVerifyChecksum (
 void)
```

Verifies the checksum, which indicates that a valid user program is present and can be started.

#### Returns

BLT\_TRUE if successful, BLT\_FALSE otherwise.

### 7.1113.2.11 NvmWrite()

```
blt_bool NvmWrite (
 blt_addr addr,
 blt_int32u len,
 blt_int8u * data)
```

Programs the non-volatile memory.

#### Parameters

|             |                             |
|-------------|-----------------------------|
| <i>addr</i> | Start address.              |
| <i>len</i>  | Length in bytes.            |
| <i>data</i> | Pointer to the data buffer. |

#### Returns

BLT\_TRUE if successful, BLT\_FALSE otherwise.

### 7.1113.2.12 NvmWriteHook()

```
blt_int8u NvmWriteHook (
 blt_addr addr,
 blt_int32u len,
 blt_int8u * data)
```

Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the data hasn't been written yet.

#### Parameters

|             |                             |
|-------------|-----------------------------|
| <i>addr</i> | Start address.              |
| <i>len</i>  | Length in bytes.            |
| <i>data</i> | Pointer to the data buffer. |

## Returns

BLT\_NVM\_OKAY if successful, BLT\_NVM\_NOT\_IN\_RANGE if the address is not within the supported memory range, or BLT\_NVM\_ERROR if the write operation failed.

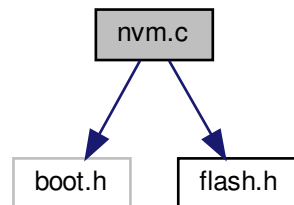
Referenced by NvmWrite().

## 7.1114 nvm.c File Reference

Bootloader non-volatile memory driver source file.

```
#include "boot.h"
#include "flash.h"
```

Include dependency graph for ARMC32\_EFM32/nvm.c:



## Functions

- void [NvmInitHook](#) (void)  
*Callback that gets called at the start of the internal NVM driver initialization routine.*
- void [NvmReinitHook](#) (void)  
*Callback that gets called at the start of a firmware update to reinitialize the NVM driver.*
- [blt\\_int8u NvmWriteHook](#) ([blt\\_addr](#) addr, [blt\\_int32u](#) len, [blt\\_int8u](#) \*data)  
*Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the data hasn't been written yet.*
- [blt\\_int8u NvmEraseHook](#) ([blt\\_addr](#) addr, [blt\\_int32u](#) len)  
*Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the memory hasn't been erased yet.*
- [blt\\_bool NvmDoneHook](#) (void)  
*Callback that gets called at the end of the NVM programming session.*
- void [NvmInit](#) (void)  
*Initializes the NVM driver.*
- void [NvmReinit](#) (void)  
*Reinitializes the NVM driver. This function is called at the start of each firmware update as opposed to NvmInit, which is only called once during power on.*
- [blt\\_bool NvmWrite](#) ([blt\\_addr](#) addr, [blt\\_int32u](#) len, [blt\\_int8u](#) \*data)

- Programs the non-volatile memory.*
- [blt\\_bool NvmErase](#) ([blt\\_addr](#) addr, [blt\\_int32u](#) len)
- Erases the non-volatile memory.*
- [blt\\_bool NvmVerifyChecksum](#) (void)
- Verifies the checksum, which indicates that a valid user program is present and can be started.*
- [blt\\_addr NvmGetUserProgBaseAddress](#) (void)
- Obtains the base address of the non-volatile memory available to the user program. This is typically that start of the vector table.*
- [blt\\_bool NvmDone](#) (void)
- Once all erase and programming operations are completed, this function is called, so at the end of the programming session and right before a software reset is performed. It is used to calculate a checksum and program this into flash. This checksum is later used to determine if a valid user program is present in flash.*

### 7.1114.1 Detailed Description

Bootloader non-volatile memory driver source file.

### 7.1114.2 Function Documentation

#### 7.1114.2.1 NvmDone()

```
blt_bool NvmDone (
 void)
```

Once all erase and programming operations are completed, this function is called, so at the end of the programming session and right before a software reset is performed. It is used to calculate a checksum and program this into flash. This checksum is later used to determine if a valid user program is present in flash.

##### Returns

BLT\_TRUE if successful, BLT\_FALSE otherwise.

#### 7.1114.2.2 NvmDoneHook()

```
blt_bool NvmDoneHook (
 void)
```

Callback that gets called at the end of the NVM programming session.

##### Returns

BLT\_TRUE is successful, BLT\_FALSE otherwise.

Referenced by [NvmDone\(\)](#).

#### 7.1114.2.3 NvmErase()

```
blt_bool NvmErase (
 blt_addr addr,
 blt_int32u len)
```

Erases the non-volatile memory.

**Parameters**

|             |                  |
|-------------|------------------|
| <i>addr</i> | Start address.   |
| <i>len</i>  | Length in bytes. |

**Returns**

BLT\_TRUE if successful, BLT\_FALSE otherwise.

**7.1114.2.4 NvmEraseHook()**

```
blt_int8u NvmEraseHook (
 blt_addr addr,
 blt_int32u len)
```

Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the memory hasn't been erased yet.

**Parameters**

|             |                  |
|-------------|------------------|
| <i>addr</i> | Start address.   |
| <i>len</i>  | Length in bytes. |

**Returns**

BLT\_NVM\_OKAY if successful, BLT\_NVM\_NOT\_IN\_RANGE if the address is not within the supported memory range, or BLT\_NVM\_ERROR is the erase operation failed.

Referenced by NvmErase().

**7.1114.2.5 NvmGetUserProgBaseAddress()**

```
blt_addr NvmGetUserProgBaseAddress (
 void)
```

Obtains the base address of the non-volatile memory available to the user program. This is typically that start of the vector table.

**Returns**

Base address.

#### 7.1114.2.6 NvmInit()

```
void NvmInit (
 void)
```

Initializes the NVM driver.

##### Returns

none.

#### 7.1114.2.7 NvmInitHook()

```
void NvmInitHook (
 void)
```

Callback that gets called at the start of the internal NVM driver initialization routine.

##### Returns

none.

Referenced by NvmInit().

#### 7.1114.2.8 NvmReinit()

```
void NvmReinit (
 void)
```

Reinitializes the NVM driver. This function is called at the start of each firmware update as opposed to NvmInit, which is only called once during power on.

##### Returns

none.

#### 7.1114.2.9 NvmReinitHook()

```
void NvmReinitHook (
 void)
```

Callback that gets called at the start of a firmware update to reinitialize the NVM driver.

##### Returns

none.

Referenced by NvmReinit().

#### 7.1114.2.10 NvmVerifyChecksum()

```
blt_bool NvmVerifyChecksum (
 void)
```

Verifies the checksum, which indicates that a valid user program is present and can be started.

##### Returns

BLT\_TRUE if successful, BLT\_FALSE otherwise.

#### 7.1114.2.11 NvmWrite()

```
blt_bool NvmWrite (
 blt_addr addr,
 blt_int32u len,
 blt_int8u * data)
```

Programs the non-volatile memory.

##### Parameters

|             |                             |
|-------------|-----------------------------|
| <i>addr</i> | Start address.              |
| <i>len</i>  | Length in bytes.            |
| <i>data</i> | Pointer to the data buffer. |

##### Returns

BLT\_TRUE if successful, BLT\_FALSE otherwise.

#### 7.1114.2.12 NvmWriteHook()

```
blt_int8u NvmWriteHook (
 blt_addr addr,
 blt_int32u len,
 blt_int8u * data)
```

Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the data hasn't been written yet.

##### Parameters

|             |                             |
|-------------|-----------------------------|
| <i>addr</i> | Start address.              |
| <i>len</i>  | Length in bytes.            |
| <i>data</i> | Pointer to the data buffer. |

## Returns

BLT\_NVM\_OKAY if successful, BLT\_NVM\_NOT\_IN\_RANGE if the address is not within the supported memory range, or BLT\_NVM\_ERROR if the write operation failed.

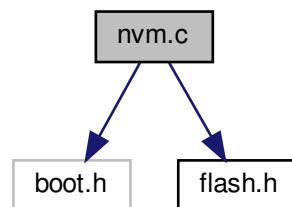
Referenced by NvmWrite().

## 7.1115 nvm.c File Reference

Bootloader non-volatile memory driver source file.

```
#include "boot.h"
#include "flash.h"
```

Include dependency graph for ARMCM3\_LM3S/nvm.c:



## Functions

- void [NvmInitHook](#) (void)  
*Callback that gets called at the start of the internal NVM driver initialization routine.*
- void [NvmReinitHook](#) (void)  
*Callback that gets called at the start of a firmware update to reinitialize the NVM driver.*
- [blt\\_int8u NvmWriteHook](#) ([blt\\_addr](#) addr, [blt\\_int32u](#) len, [blt\\_int8u](#) \*data)  
*Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the data hasn't been written yet.*
- [blt\\_int8u NvmEraseHook](#) ([blt\\_addr](#) addr, [blt\\_int32u](#) len)  
*Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the memory hasn't been erased yet.*
- [blt\\_bool NvmDoneHook](#) (void)  
*Callback that gets called at the end of the NVM programming session.*
- void [NvmInit](#) (void)  
*Initializes the NVM driver.*
- void [NvmReinit](#) (void)  
*Reinitializes the NVM driver. This function is called at the start of each firmware update as opposed to NvmInit, which is only called once during power on.*
- [blt\\_bool NvmWrite](#) ([blt\\_addr](#) addr, [blt\\_int32u](#) len, [blt\\_int8u](#) \*data)

- Programs the non-volatile memory.*
    - [blt\\_bool NvmErase](#) ([blt\\_addr](#) addr, [blt\\_int32u](#) len)
  - Erases the non-volatile memory.*
    - [blt\\_bool NvmVerifyChecksum](#) (void)
  - Verifies the checksum, which indicates that a valid user program is present and can be started.*
    - [blt\\_addr NvmGetUserProgBaseAddress](#) (void)
  - Obtains the base address of the non-volatile memory available to the user program. This is typically that start of the vector table.*
    - [blt\\_bool NvmDone](#) (void)
- Once all erase and programming operations are completed, this function is called, so at the end of the programming session and right before a software reset is performed. It is used to calculate a checksum and program this into flash. This checksum is later used to determine if a valid user program is present in flash.

### 7.1115.1 Detailed Description

Bootloader non-volatile memory driver source file.

### 7.1115.2 Function Documentation

#### 7.1115.2.1 NvmDone()

```
blt_bool NvmDone (
 void)
```

Once all erase and programming operations are completed, this function is called, so at the end of the programming session and right before a software reset is performed. It is used to calculate a checksum and program this into flash. This checksum is later used to determine if a valid user program is present in flash.

##### Returns

BLT\_TRUE if successful, BLT\_FALSE otherwise.

#### 7.1115.2.2 NvmDoneHook()

```
blt_bool NvmDoneHook (
 void)
```

Callback that gets called at the end of the NVM programming session.

##### Returns

BLT\_TRUE is successful, BLT\_FALSE otherwise.

Referenced by NvmDone().

#### 7.1115.2.3 NvmErase()

```
blt_bool NvmErase (
 blt_addr addr,
 blt_int32u len)
```

Erases the non-volatile memory.



**Parameters**

|             |                  |
|-------------|------------------|
| <i>addr</i> | Start address.   |
| <i>len</i>  | Length in bytes. |

**Returns**

BLT\_TRUE if successful, BLT\_FALSE otherwise.

**7.1115.2.4 NvmEraseHook()**

```
blt_int8u NvmEraseHook (
 blt_addr addr,
 blt_int32u len)
```

Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the memory hasn't been erased yet.

**Parameters**

|             |                  |
|-------------|------------------|
| <i>addr</i> | Start address.   |
| <i>len</i>  | Length in bytes. |

**Returns**

BLT\_NVM\_OKAY if successful, BLT\_NVM\_NOT\_IN\_RANGE if the address is not within the supported memory range, or BLT\_NVM\_ERROR is the erase operation failed.

Referenced by NvmErase().

**7.1115.2.5 NvmGetUserProgBaseAddress()**

```
blt_addr NvmGetUserProgBaseAddress (
 void)
```

Obtains the base address of the non-volatile memory available to the user program. This is typically that start of the vector table.

**Returns**

Base address.

#### 7.1115.2.6 NvmInit()

```
void NvmInit (
 void)
```

Initializes the NVM driver.

##### Returns

none.

#### 7.1115.2.7 NvmInitHook()

```
void NvmInitHook (
 void)
```

Callback that gets called at the start of the internal NVM driver initialization routine.

##### Returns

none.

Referenced by NvmInit().

#### 7.1115.2.8 NvmReinit()

```
void NvmReinit (
 void)
```

Reinitializes the NVM driver. This function is called at the start of each firmware update as opposed to NvmInit, which is only called once during power on.

##### Returns

none.

#### 7.1115.2.9 NvmReinitHook()

```
void NvmReinitHook (
 void)
```

Callback that gets called at the start of a firmware update to reinitialize the NVM driver.

##### Returns

none.

Referenced by NvmReinit().

### 7.1115.2.10 NvmVerifyChecksum()

```
blt_bool NvmVerifyChecksum (
 void)
```

Verifies the checksum, which indicates that a valid user program is present and can be started.

#### Returns

BLT\_TRUE if successful, BLT\_FALSE otherwise.

### 7.1115.2.11 NvmWrite()

```
blt_bool NvmWrite (
 blt_addr addr,
 blt_int32u len,
 blt_int8u * data)
```

Programs the non-volatile memory.

#### Parameters

|             |                             |
|-------------|-----------------------------|
| <i>addr</i> | Start address.              |
| <i>len</i>  | Length in bytes.            |
| <i>data</i> | Pointer to the data buffer. |

#### Returns

BLT\_TRUE if successful, BLT\_FALSE otherwise.

### 7.1115.2.12 NvmWriteHook()

```
blt_int8u NvmWriteHook (
 blt_addr addr,
 blt_int32u len,
 blt_int8u * data)
```

Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the data hasn't been written yet.

#### Parameters

|             |                             |
|-------------|-----------------------------|
| <i>addr</i> | Start address.              |
| <i>len</i>  | Length in bytes.            |
| <i>data</i> | Pointer to the data buffer. |

## Returns

BLT\_NVM\_OKAY if successful, BLT\_NVM\_NOT\_IN\_RANGE if the address is not within the supported memory range, or BLT\_NVM\_ERROR if the write operation failed.

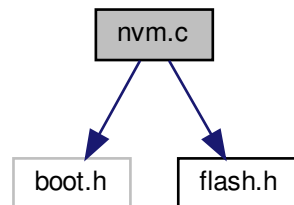
Referenced by NvmWrite().

## 7.1116 nvm.c File Reference

Bootloader non-volatile memory driver source file.

```
#include "boot.h"
#include "flash.h"
```

Include dependency graph for ARMCM3\_STM32F1/nvm.c:



## Functions

- void [NvmInitHook](#) (void)  
*Callback that gets called at the start of the internal NVM driver initialization routine.*
- void [NvmReinitHook](#) (void)  
*Callback that gets called at the start of a firmware update to reinitialize the NVM driver.*
- [blt\\_int8u NvmWriteHook](#) ([blt\\_addr](#) addr, [blt\\_int32u](#) len, [blt\\_int8u](#) \*data)  
*Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the data hasn't been written yet.*
- [blt\\_int8u NvmEraseHook](#) ([blt\\_addr](#) addr, [blt\\_int32u](#) len)  
*Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the memory hasn't been erased yet.*
- [blt\\_bool NvmDoneHook](#) (void)  
*Callback that gets called at the end of the NVM programming session.*
- void [NvmInit](#) (void)  
*Initializes the NVM driver.*
- void [NvmReinit](#) (void)  
*Reinitializes the NVM driver. This function is called at the start of each firmware update as opposed to NvmInit, which is only called once during power on.*
- [blt\\_bool NvmWrite](#) ([blt\\_addr](#) addr, [blt\\_int32u](#) len, [blt\\_int8u](#) \*data)

- Programs the non-volatile memory.*
- [blt\\_bool NvmErase](#) ([blt\\_addr](#) addr, [blt\\_int32u](#) len)
- Erases the non-volatile memory.*
- [blt\\_bool NvmVerifyChecksum](#) (void)
- Verifies the checksum, which indicates that a valid user program is present and can be started.*
- [blt\\_addr NvmGetUserProgBaseAddress](#) (void)
- Obtains the base address of the non-volatile memory available to the user program. This is typically that start of the vector table.*
- [blt\\_bool NvmDone](#) (void)
- Once all erase and programming operations are completed, this function is called, so at the end of the programming session and right before a software reset is performed. It is used to calculate a checksum and program this into flash. This checksum is later used to determine if a valid user program is present in flash.*

## 7.1116.1 Detailed Description

Bootloader non-volatile memory driver source file.

## 7.1116.2 Function Documentation

### 7.1116.2.1 NvmDone()

```
blt_bool NvmDone (
 void)
```

Once all erase and programming operations are completed, this function is called, so at the end of the programming session and right before a software reset is performed. It is used to calculate a checksum and program this into flash. This checksum is later used to determine if a valid user program is present in flash.

#### Returns

BLT\_TRUE if successful, BLT\_FALSE otherwise.

### 7.1116.2.2 NvmDoneHook()

```
blt_bool NvmDoneHook (
 void)
```

Callback that gets called at the end of the NVM programming session.

#### Returns

BLT\_TRUE is successful, BLT\_FALSE otherwise.

Referenced by [NvmDone\(\)](#).

### 7.1116.2.3 NvmErase()

```
blt_bool NvmErase (
 blt_addr addr,
 blt_int32u len)
```

Erases the non-volatile memory.

**Parameters**

|             |                  |
|-------------|------------------|
| <i>addr</i> | Start address.   |
| <i>len</i>  | Length in bytes. |

**Returns**

BLT\_TRUE if successful, BLT\_FALSE otherwise.

**7.1116.2.4 NvmEraseHook()**

```
blt_int8u NvmEraseHook (
 blt_addr addr,
 blt_int32u len)
```

Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the memory hasn't been erased yet.

**Parameters**

|             |                  |
|-------------|------------------|
| <i>addr</i> | Start address.   |
| <i>len</i>  | Length in bytes. |

**Returns**

BLT\_NVM\_OKAY if successful, BLT\_NVM\_NOT\_IN\_RANGE if the address is not within the supported memory range, or BLT\_NVM\_ERROR is the erase operation failed.

Referenced by NvmErase().

**7.1116.2.5 NvmGetUserProgBaseAddress()**

```
blt_addr NvmGetUserProgBaseAddress (
 void)
```

Obtains the base address of the non-volatile memory available to the user program. This is typically that start of the vector table.

**Returns**

Base address.

### 7.1116.2.6 NvmInit()

```
void NvmInit (
 void)
```

Initializes the NVM driver.

#### Returns

none.

### 7.1116.2.7 NvmInitHook()

```
void NvmInitHook (
 void)
```

Callback that gets called at the start of the internal NVM driver initialization routine.

#### Returns

none.

Referenced by NvmInit().

### 7.1116.2.8 NvmReinit()

```
void NvmReinit (
 void)
```

Reinitializes the NVM driver. This function is called at the start of each firmware update as opposed to NvmInit, which is only called once during power on.

#### Returns

none.

### 7.1116.2.9 NvmReinitHook()

```
void NvmReinitHook (
 void)
```

Callback that gets called at the start of a firmware update to reinitialize the NVM driver.

#### Returns

none.

Referenced by NvmReinit().

#### 7.1116.2.10 NvmVerifyChecksum()

```
blt_bool NvmVerifyChecksum (
 void)
```

Verifies the checksum, which indicates that a valid user program is present and can be started.

##### Returns

BLT\_TRUE if successful, BLT\_FALSE otherwise.

#### 7.1116.2.11 NvmWrite()

```
blt_bool NvmWrite (
 blt_addr addr,
 blt_int32u len,
 blt_int8u * data)
```

Programs the non-volatile memory.

##### Parameters

|             |                             |
|-------------|-----------------------------|
| <i>addr</i> | Start address.              |
| <i>len</i>  | Length in bytes.            |
| <i>data</i> | Pointer to the data buffer. |

##### Returns

BLT\_TRUE if successful, BLT\_FALSE otherwise.

#### 7.1116.2.12 NvmWriteHook()

```
blt_int8u NvmWriteHook (
 blt_addr addr,
 blt_int32u len,
 blt_int8u * data)
```

Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the data hasn't been written yet.

##### Parameters

|             |                             |
|-------------|-----------------------------|
| <i>addr</i> | Start address.              |
| <i>len</i>  | Length in bytes.            |
| <i>data</i> | Pointer to the data buffer. |



## Returns

BLT\_NVM\_OKAY if successful, BLT\_NVM\_NOT\_IN\_RANGE if the address is not within the supported memory range, or BLT\_NVM\_ERROR if the write operation failed.

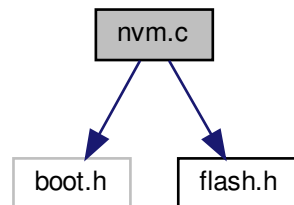
Referenced by NvmWrite().

## 7.1117 nvm.c File Reference

Bootloader non-volatile memory driver source file.

```
#include "boot.h"
#include "flash.h"
```

Include dependency graph for ARMCM3\_STM32F2/nvm.c:



### Functions

- void [NvmInitHook](#) (void)  
*Callback that gets called at the start of the internal NVM driver initialization routine.*
- void [NvmReinitHook](#) (void)  
*Callback that gets called at the start of a firmware update to reinitialize the NVM driver.*
- [blt\\_int8u NvmWriteHook](#) ([blt\\_addr](#) addr, [blt\\_int32u](#) len, [blt\\_int8u](#) \*data)  
*Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the data hasn't been written yet.*
- [blt\\_int8u NvmEraseHook](#) ([blt\\_addr](#) addr, [blt\\_int32u](#) len)  
*Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the memory hasn't been erased yet.*
- [blt\\_bool NvmDoneHook](#) (void)  
*Callback that gets called at the end of the NVM programming session.*
- void [NvmInit](#) (void)  
*Initializes the NVM driver.*
- void [NvmReinit](#) (void)  
*Reinitializes the NVM driver. This function is called at the start of each firmware update as opposed to NvmInit, which is only called once during power on.*
- [blt\\_bool NvmWrite](#) ([blt\\_addr](#) addr, [blt\\_int32u](#) len, [blt\\_int8u](#) \*data)

*Programs the non-volatile memory.*

- [blt\\_bool NvmErase](#) ([blt\\_addr](#) addr, [blt\\_int32u](#) len)

*Erases the non-volatile memory.*

- [blt\\_bool NvmVerifyChecksum](#) (void)

*Verifies the checksum, which indicates that a valid user program is present and can be started.*

- [blt\\_addr NvmGetUserProgBaseAddress](#) (void)

*Obtains the base address of the non-volatile memory available to the user program. This is typically that start of the vector table.*

- [blt\\_bool NvmDone](#) (void)

*Once all erase and programming operations are completed, this function is called, so at the end of the programming session and right before a software reset is performed. It is used to calculate a checksum and program this into flash. This checksum is later used to determine if a valid user program is present in flash.*

## 7.1117.1 Detailed Description

Bootloader non-volatile memory driver source file.

## 7.1117.2 Function Documentation

### 7.1117.2.1 NvmDone()

```
blt_bool NvmDone (
 void)
```

Once all erase and programming operations are completed, this function is called, so at the end of the programming session and right before a software reset is performed. It is used to calculate a checksum and program this into flash. This checksum is later used to determine if a valid user program is present in flash.

#### Returns

BLT\_TRUE if successful, BLT\_FALSE otherwise.

### 7.1117.2.2 NvmDoneHook()

```
blt_bool NvmDoneHook (
 void)
```

Callback that gets called at the end of the NVM programming session.

#### Returns

BLT\_TRUE is successful, BLT\_FALSE otherwise.

Referenced by [NvmDone\(\)](#).

### 7.1117.2.3 NvmErase()

```
blt_bool NvmErase (
 blt_addr addr,
 blt_int32u len)
```

Erases the non-volatile memory.

**Parameters**

|             |                  |
|-------------|------------------|
| <i>addr</i> | Start address.   |
| <i>len</i>  | Length in bytes. |

**Returns**

BLT\_TRUE if successful, BLT\_FALSE otherwise.

**7.1117.2.4 NvmEraseHook()**

```
blt_int8u NvmEraseHook (
 blt_addr addr,
 blt_int32u len)
```

Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the memory hasn't been erased yet.

**Parameters**

|             |                  |
|-------------|------------------|
| <i>addr</i> | Start address.   |
| <i>len</i>  | Length in bytes. |

**Returns**

BLT\_NVM\_OKAY if successful, BLT\_NVM\_NOT\_IN\_RANGE if the address is not within the supported memory range, or BLT\_NVM\_ERROR is the erase operation failed.

Referenced by NvmErase().

**7.1117.2.5 NvmGetUserProgBaseAddress()**

```
blt_addr NvmGetUserProgBaseAddress (
 void)
```

Obtains the base address of the non-volatile memory available to the user program. This is typically that start of the vector table.

**Returns**

Base address.

**7.1117.2.6 NvmInit()**

```
void NvmInit (
 void)
```

Initializes the NVM driver.

**Returns**

none.

**7.1117.2.7 NvmInitHook()**

```
void NvmInitHook (
 void)
```

Callback that gets called at the start of the internal NVM driver initialization routine.

**Returns**

none.

Referenced by NvmInit().

**7.1117.2.8 NvmReinit()**

```
void NvmReinit (
 void)
```

Reinitializes the NVM driver. This function is called at the start of each firmware update as opposed to NvmInit, which is only called once during power on.

**Returns**

none.

**7.1117.2.9 NvmReinitHook()**

```
void NvmReinitHook (
 void)
```

Callback that gets called at the start of a firmware update to reinitialize the NVM driver.

**Returns**

none.

Referenced by NvmReinit().

### 7.1117.2.10 NvmVerifyChecksum()

```
blt_bool NvmVerifyChecksum (
 void)
```

Verifies the checksum, which indicates that a valid user program is present and can be started.

#### Returns

BLT\_TRUE if successful, BLT\_FALSE otherwise.

### 7.1117.2.11 NvmWrite()

```
blt_bool NvmWrite (
 blt_addr addr,
 blt_int32u len,
 blt_int8u * data)
```

Programs the non-volatile memory.

#### Parameters

|             |                             |
|-------------|-----------------------------|
| <i>addr</i> | Start address.              |
| <i>len</i>  | Length in bytes.            |
| <i>data</i> | Pointer to the data buffer. |

#### Returns

BLT\_TRUE if successful, BLT\_FALSE otherwise.

### 7.1117.2.12 NvmWriteHook()

```
blt_int8u NvmWriteHook (
 blt_addr addr,
 blt_int32u len,
 blt_int8u * data)
```

Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the data hasn't been written yet.

#### Parameters

|             |                             |
|-------------|-----------------------------|
| <i>addr</i> | Start address.              |
| <i>len</i>  | Length in bytes.            |
| <i>data</i> | Pointer to the data buffer. |

## Returns

BLT\_NVM\_OKAY if successful, BLT\_NVM\_NOT\_IN\_RANGE if the address is not within the supported memory range, or BLT\_NVM\_ERROR if the write operation failed.

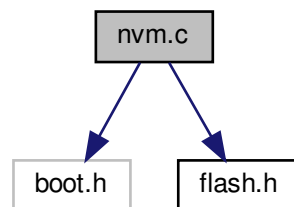
Referenced by NvmWrite().

## 7.1118 nvm.c File Reference

Bootloader non-volatile memory driver source file.

```
#include "boot.h"
#include "flash.h"
```

Include dependency graph for ARMCM4\_S32K14/nvm.c:



## Functions

- void [NvmInitHook](#) (void)  
*Callback that gets called at the start of the internal NVM driver initialization routine.*
- void [NvmReinitHook](#) (void)  
*Callback that gets called at the start of a firmware update to reinitialize the NVM driver.*
- [blt\\_int8u NvmWriteHook](#) ([blt\\_addr](#) addr, [blt\\_int32u](#) len, [blt\\_int8u](#) \*data)  
*Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the data hasn't been written yet.*
- [blt\\_int8u NvmEraseHook](#) ([blt\\_addr](#) addr, [blt\\_int32u](#) len)  
*Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the memory hasn't been erased yet.*
- [blt\\_bool NvmDoneHook](#) (void)  
*Callback that gets called at the end of the NVM programming session.*
- void [NvmInit](#) (void)  
*Initializes the NVM driver.*
- void [NvmReinit](#) (void)  
*Reinitializes the NVM driver. This function is called at the start of each firmware update as opposed to NvmInit, which is only called once during power on.*
- [blt\\_bool NvmWrite](#) ([blt\\_addr](#) addr, [blt\\_int32u](#) len, [blt\\_int8u](#) \*data)

- Programs the non-volatile memory.*
  - [blt\\_bool NvmErase](#) ([blt\\_addr](#) addr, [blt\\_int32u](#) len)
    - Erases the non-volatile memory.*
  - [blt\\_bool NvmVerifyChecksum](#) (void)
    - Verifies the checksum, which indicates that a valid user program is present and can be started.*
  - [blt\\_addr NvmGetUserProgBaseAddress](#) (void)
    - Obtains the base address of the non-volatile memory available to the user program. This is typically that start of the vector table.*
  - [blt\\_bool NvmDone](#) (void)
    - Once all erase and programming operations are completed, this function is called, so at the end of the programming session and right before a software reset is performed. It is used to calculate a checksum and program this into flash. This checksum is later used to determine if a valid user program is present in flash.*

### 7.1118.1 Detailed Description

Bootloader non-volatile memory driver source file.

### 7.1118.2 Function Documentation

#### 7.1118.2.1 NvmDone()

```
blt_bool NvmDone (
 void)
```

Once all erase and programming operations are completed, this function is called, so at the end of the programming session and right before a software reset is performed. It is used to calculate a checksum and program this into flash. This checksum is later used to determine if a valid user program is present in flash.

##### Returns

BLT\_TRUE if successful, BLT\_FALSE otherwise.

#### 7.1118.2.2 NvmDoneHook()

```
blt_bool NvmDoneHook (
 void)
```

Callback that gets called at the end of the NVM programming session.

##### Returns

BLT\_TRUE is successful, BLT\_FALSE otherwise.

Referenced by [NvmDone\(\)](#).

#### 7.1118.2.3 NvmErase()

```
blt_bool NvmErase (
 blt_addr addr,
 blt_int32u len)
```

Erases the non-volatile memory.

**Parameters**

|             |                  |
|-------------|------------------|
| <i>addr</i> | Start address.   |
| <i>len</i>  | Length in bytes. |

**Returns**

BLT\_TRUE if successful, BLT\_FALSE otherwise.

**7.1118.2.4 NvmEraseHook()**

```
blt_int8u NvmEraseHook (
 blt_addr addr,
 blt_int32u len)
```

Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the memory hasn't been erased yet.

**Parameters**

|             |                  |
|-------------|------------------|
| <i>addr</i> | Start address.   |
| <i>len</i>  | Length in bytes. |

**Returns**

BLT\_NVM\_OKAY if successful, BLT\_NVM\_NOT\_IN\_RANGE if the address is not within the supported memory range, or BLT\_NVM\_ERROR is the erase operation failed.

Referenced by NvmErase().

**7.1118.2.5 NvmGetUserProgBaseAddress()**

```
blt_addr NvmGetUserProgBaseAddress (
 void)
```

Obtains the base address of the non-volatile memory available to the user program. This is typically that start of the vector table.

**Returns**

Base address.



#### 7.1118.2.6 NvmInit()

```
void NvmInit (
 void)
```

Initializes the NVM driver.

##### Returns

none.

#### 7.1118.2.7 NvmInitHook()

```
void NvmInitHook (
 void)
```

Callback that gets called at the start of the internal NVM driver initialization routine.

##### Returns

none.

Referenced by NvmInit().

#### 7.1118.2.8 NvmReinit()

```
void NvmReinit (
 void)
```

Reinitializes the NVM driver. This function is called at the start of each firmware update as opposed to NvmInit, which is only called once during power on.

##### Returns

none.

#### 7.1118.2.9 NvmReinitHook()

```
void NvmReinitHook (
 void)
```

Callback that gets called at the start of a firmware update to reinitialize the NVM driver.

##### Returns

none.

Referenced by NvmReinit().

#### 7.1118.2.10 NvmVerifyChecksum()

```
blt_bool NvmVerifyChecksum (
 void)
```

Verifies the checksum, which indicates that a valid user program is present and can be started.

##### Returns

BLT\_TRUE if successful, BLT\_FALSE otherwise.

#### 7.1118.2.11 NvmWrite()

```
blt_bool NvmWrite (
 blt_addr addr,
 blt_int32u len,
 blt_int8u * data)
```

Programs the non-volatile memory.

##### Parameters

|             |                             |
|-------------|-----------------------------|
| <i>addr</i> | Start address.              |
| <i>len</i>  | Length in bytes.            |
| <i>data</i> | Pointer to the data buffer. |

##### Returns

BLT\_TRUE if successful, BLT\_FALSE otherwise.

#### 7.1118.2.12 NvmWriteHook()

```
blt_int8u NvmWriteHook (
 blt_addr addr,
 blt_int32u len,
 blt_int8u * data)
```

Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the data hasn't been written yet.

##### Parameters

|             |                             |
|-------------|-----------------------------|
| <i>addr</i> | Start address.              |
| <i>len</i>  | Length in bytes.            |
| <i>data</i> | Pointer to the data buffer. |

## Returns

BLT\_NVM\_OKAY if successful, BLT\_NVM\_NOT\_IN\_RANGE if the address is not within the supported memory range, or BLT\_NVM\_ERROR if the write operation failed.

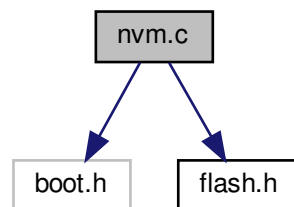
Referenced by NvmWrite().

## 7.1119 nvm.c File Reference

Bootloader non-volatile memory driver source file.

```
#include "boot.h"
#include "flash.h"
```

Include dependency graph for ARMCM4\_STM32F3/nvm.c:



## Functions

- void [NvmInitHook](#) (void)  
*Callback that gets called at the start of the internal NVM driver initialization routine.*
- void [NvmReinitHook](#) (void)  
*Callback that gets called at the start of a firmware update to reinitialize the NVM driver.*
- [blt\\_int8u NvmWriteHook](#) ([blt\\_addr](#) addr, [blt\\_int32u](#) len, [blt\\_int8u](#) \*data)  
*Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the data hasn't been written yet.*
- [blt\\_int8u NvmEraseHook](#) ([blt\\_addr](#) addr, [blt\\_int32u](#) len)  
*Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the memory hasn't been erased yet.*
- [blt\\_bool NvmDoneHook](#) (void)  
*Callback that gets called at the end of the NVM programming session.*
- void [NvmInit](#) (void)  
*Initializes the NVM driver.*
- void [NvmReinit](#) (void)  
*Reinitializes the NVM driver. This function is called at the start of each firmware update as opposed to NvmInit, which is only called once during power on.*
- [blt\\_bool NvmWrite](#) ([blt\\_addr](#) addr, [blt\\_int32u](#) len, [blt\\_int8u](#) \*data)

- Programs the non-volatile memory.*
- [blt\\_bool NvmErase](#) ([blt\\_addr](#) addr, [blt\\_int32u](#) len)
- Erases the non-volatile memory.*
- [blt\\_bool NvmVerifyChecksum](#) (void)
- Verifies the checksum, which indicates that a valid user program is present and can be started.*
- [blt\\_addr NvmGetUserProgBaseAddress](#) (void)
- Obtains the base address of the non-volatile memory available to the user program. This is typically that start of the vector table.*
- [blt\\_bool NvmDone](#) (void)
- Once all erase and programming operations are completed, this function is called, so at the end of the programming session and right before a software reset is performed. It is used to calculate a checksum and program this into flash. This checksum is later used to determine if a valid user program is present in flash.*

## 7.1119.1 Detailed Description

Bootloader non-volatile memory driver source file.

## 7.1119.2 Function Documentation

### 7.1119.2.1 NvmDone()

```
blt_bool NvmDone (
 void)
```

Once all erase and programming operations are completed, this function is called, so at the end of the programming session and right before a software reset is performed. It is used to calculate a checksum and program this into flash. This checksum is later used to determine if a valid user program is present in flash.

#### Returns

BLT\_TRUE if successful, BLT\_FALSE otherwise.

### 7.1119.2.2 NvmDoneHook()

```
blt_bool NvmDoneHook (
 void)
```

Callback that gets called at the end of the NVM programming session.

#### Returns

BLT\_TRUE is successful, BLT\_FALSE otherwise.

Referenced by [NvmDone\(\)](#).

### 7.1119.2.3 NvmErase()

```
blt_bool NvmErase (
 blt_addr addr,
 blt_int32u len)
```

Erases the non-volatile memory.

**Parameters**

|             |                  |
|-------------|------------------|
| <i>addr</i> | Start address.   |
| <i>len</i>  | Length in bytes. |

**Returns**

BLT\_TRUE if successful, BLT\_FALSE otherwise.

**7.1119.2.4 NvmEraseHook()**

```
blt_int8u NvmEraseHook (
 blt_addr addr,
 blt_int32u len)
```

Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the memory hasn't been erased yet.

**Parameters**

|             |                  |
|-------------|------------------|
| <i>addr</i> | Start address.   |
| <i>len</i>  | Length in bytes. |

**Returns**

BLT\_NVM\_OKAY if successful, BLT\_NVM\_NOT\_IN\_RANGE if the address is not within the supported memory range, or BLT\_NVM\_ERROR is the erase operation failed.

Referenced by NvmErase().

**7.1119.2.5 NvmGetUserProgBaseAddress()**

```
blt_addr NvmGetUserProgBaseAddress (
 void)
```

Obtains the base address of the non-volatile memory available to the user program. This is typically that start of the vector table.

**Returns**

Base address.

**7.1119.2.6 NvmInit()**

```
void NvmInit (
 void)
```

Initializes the NVM driver.

**Returns**

none.

**7.1119.2.7 NvmInitHook()**

```
void NvmInitHook (
 void)
```

Callback that gets called at the start of the internal NVM driver initialization routine.

**Returns**

none.

Referenced by NvmInit().

**7.1119.2.8 NvmReinit()**

```
void NvmReinit (
 void)
```

Reinitializes the NVM driver. This function is called at the start of each firmware update as opposed to NvmInit, which is only called once during power on.

**Returns**

none.

**7.1119.2.9 NvmReinitHook()**

```
void NvmReinitHook (
 void)
```

Callback that gets called at the start of a firmware update to reinitialize the NVM driver.

**Returns**

none.

Referenced by NvmReinit().

### 7.1119.2.10 NvmVerifyChecksum()

```
blt_bool NvmVerifyChecksum (
 void)
```

Verifies the checksum, which indicates that a valid user program is present and can be started.

#### Returns

BLT\_TRUE if successful, BLT\_FALSE otherwise.

### 7.1119.2.11 NvmWrite()

```
blt_bool NvmWrite (
 blt_addr addr,
 blt_int32u len,
 blt_int8u * data)
```

Programs the non-volatile memory.

#### Parameters

|             |                             |
|-------------|-----------------------------|
| <i>addr</i> | Start address.              |
| <i>len</i>  | Length in bytes.            |
| <i>data</i> | Pointer to the data buffer. |

#### Returns

BLT\_TRUE if successful, BLT\_FALSE otherwise.

### 7.1119.2.12 NvmWriteHook()

```
blt_int8u NvmWriteHook (
 blt_addr addr,
 blt_int32u len,
 blt_int8u * data)
```

Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the data hasn't been written yet.

#### Parameters

|             |                             |
|-------------|-----------------------------|
| <i>addr</i> | Start address.              |
| <i>len</i>  | Length in bytes.            |
| <i>data</i> | Pointer to the data buffer. |

### Returns

BLT\_NVM\_OKAY if successful, BLT\_NVM\_NOT\_IN\_RANGE if the address is not within the supported memory range, or BLT\_NVM\_ERROR if the write operation failed.

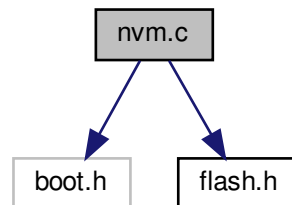
Referenced by NvmWrite().

## 7.1120 nvm.c File Reference

Bootloader non-volatile memory driver source file.

```
#include "boot.h"
#include "flash.h"
```

Include dependency graph for ARMCM4\_STM32F4/nvm.c:



### Functions

- void [NvmInitHook](#) (void)  
*Callback that gets called at the start of the internal NVM driver initialization routine.*
- void [NvmReinitHook](#) (void)  
*Callback that gets called at the start of a firmware update to reinitialize the NVM driver.*
- [blt\\_int8u NvmWriteHook](#) ([blt\\_addr](#) addr, [blt\\_int32u](#) len, [blt\\_int8u](#) \*data)  
*Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the data hasn't been written yet.*
- [blt\\_int8u NvmEraseHook](#) ([blt\\_addr](#) addr, [blt\\_int32u](#) len)  
*Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the memory hasn't been erased yet.*
- [blt\\_bool NvmDoneHook](#) (void)  
*Callback that gets called at the end of the NVM programming session.*
- void [NvmInit](#) (void)  
*Initializes the NVM driver.*
- void [NvmReinit](#) (void)  
*Reinitializes the NVM driver. This function is called at the start of each firmware update as opposed to NvmInit, which is only called once during power on.*
- [blt\\_bool NvmWrite](#) ([blt\\_addr](#) addr, [blt\\_int32u](#) len, [blt\\_int8u](#) \*data)



- Programs the non-volatile memory.*
- [blt\\_bool NvmErase](#) ([blt\\_addr](#) addr, [blt\\_int32u](#) len)
- Erases the non-volatile memory.*
- [blt\\_bool NvmVerifyChecksum](#) (void)
- Verifies the checksum, which indicates that a valid user program is present and can be started.*
- [blt\\_addr NvmGetUserProgBaseAddress](#) (void)
- Obtains the base address of the non-volatile memory available to the user program. This is typically that start of the vector table.*
- [blt\\_bool NvmDone](#) (void)
- Once all erase and programming operations are completed, this function is called, so at the end of the programming session and right before a software reset is performed. It is used to calculate a checksum and program this into flash. This checksum is later used to determine if a valid user program is present in flash.*

### 7.1120.1 Detailed Description

Bootloader non-volatile memory driver source file.

### 7.1120.2 Function Documentation

#### 7.1120.2.1 NvmDone()

```
blt_bool NvmDone (
 void)
```

Once all erase and programming operations are completed, this function is called, so at the end of the programming session and right before a software reset is performed. It is used to calculate a checksum and program this into flash. This checksum is later used to determine if a valid user program is present in flash.

##### Returns

BLT\_TRUE if successful, BLT\_FALSE otherwise.

#### 7.1120.2.2 NvmDoneHook()

```
blt_bool NvmDoneHook (
 void)
```

Callback that gets called at the end of the NVM programming session.

##### Returns

BLT\_TRUE is successful, BLT\_FALSE otherwise.

Referenced by [NvmDone\(\)](#).

#### 7.1120.2.3 NvmErase()

```
blt_bool NvmErase (
 blt_addr addr,
 blt_int32u len)
```

Erases the non-volatile memory.

**Parameters**

|             |                  |
|-------------|------------------|
| <i>addr</i> | Start address.   |
| <i>len</i>  | Length in bytes. |

**Returns**

BLT\_TRUE if successful, BLT\_FALSE otherwise.

**7.1120.2.4 NvmEraseHook()**

```
blt_int8u NvmEraseHook (
 blt_addr addr,
 blt_int32u len)
```

Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the memory hasn't been erased yet.

**Parameters**

|             |                  |
|-------------|------------------|
| <i>addr</i> | Start address.   |
| <i>len</i>  | Length in bytes. |

**Returns**

BLT\_NVM\_OKAY if successful, BLT\_NVM\_NOT\_IN\_RANGE if the address is not within the supported memory range, or BLT\_NVM\_ERROR is the erase operation failed.

Referenced by NvmErase().

**7.1120.2.5 NvmGetUserProgBaseAddress()**

```
blt_addr NvmGetUserProgBaseAddress (
 void)
```

Obtains the base address of the non-volatile memory available to the user program. This is typically that start of the vector table.

**Returns**

Base address.

#### 7.1120.2.6 NvmInit()

```
void NvmInit (
 void)
```

Initializes the NVM driver.

##### Returns

none.

#### 7.1120.2.7 NvmInitHook()

```
void NvmInitHook (
 void)
```

Callback that gets called at the start of the internal NVM driver initialization routine.

##### Returns

none.

Referenced by NvmInit().

#### 7.1120.2.8 NvmReinit()

```
void NvmReinit (
 void)
```

Reinitializes the NVM driver. This function is called at the start of each firmware update as opposed to NvmInit, which is only called once during power on.

##### Returns

none.

#### 7.1120.2.9 NvmReinitHook()

```
void NvmReinitHook (
 void)
```

Callback that gets called at the start of a firmware update to reinitialize the NVM driver.

##### Returns

none.

Referenced by NvmReinit().

#### 7.1120.2.10 NvmVerifyChecksum()

```
blt_bool NvmVerifyChecksum (
 void)
```

Verifies the checksum, which indicates that a valid user program is present and can be started.

##### Returns

BLT\_TRUE if successful, BLT\_FALSE otherwise.

#### 7.1120.2.11 NvmWrite()

```
blt_bool NvmWrite (
 blt_addr addr,
 blt_int32u len,
 blt_int8u * data)
```

Programs the non-volatile memory.

##### Parameters

|             |                             |
|-------------|-----------------------------|
| <i>addr</i> | Start address.              |
| <i>len</i>  | Length in bytes.            |
| <i>data</i> | Pointer to the data buffer. |

##### Returns

BLT\_TRUE if successful, BLT\_FALSE otherwise.

#### 7.1120.2.12 NvmWriteHook()

```
blt_int8u NvmWriteHook (
 blt_addr addr,
 blt_int32u len,
 blt_int8u * data)
```

Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the data hasn't been written yet.

##### Parameters

|             |                             |
|-------------|-----------------------------|
| <i>addr</i> | Start address.              |
| <i>len</i>  | Length in bytes.            |
| <i>data</i> | Pointer to the data buffer. |

## Returns

BLT\_NVM\_OKAY if successful, BLT\_NVM\_NOT\_IN\_RANGE if the address is not within the supported memory range, or BLT\_NVM\_ERROR if the write operation failed.

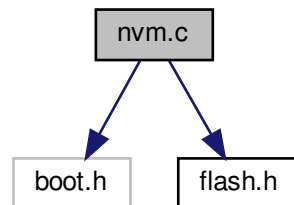
Referenced by NvmWrite().

## 7.1121 nvm.c File Reference

Bootloader non-volatile memory driver source file.

```
#include "boot.h"
#include "flash.h"
```

Include dependency graph for ARMCM4\_STM32L4/nvm.c:



## Functions

- void [NvmInitHook](#) (void)  
*Callback that gets called at the start of the internal NVM driver initialization routine.*
- void [NvmReinitHook](#) (void)  
*Callback that gets called at the start of a firmware update to reinitialize the NVM driver.*
- [blt\\_int8u NvmWriteHook](#) ([blt\\_addr](#) addr, [blt\\_int32u](#) len, [blt\\_int8u](#) \*data)  
*Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the data hasn't been written yet.*
- [blt\\_int8u NvmEraseHook](#) ([blt\\_addr](#) addr, [blt\\_int32u](#) len)  
*Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the memory hasn't been erased yet.*
- [blt\\_bool NvmDoneHook](#) (void)  
*Callback that gets called at the end of the NVM programming session.*
- void [NvmInit](#) (void)  
*Initializes the NVM driver.*
- void [NvmReinit](#) (void)  
*Reinitializes the NVM driver. This function is called at the start of each firmware update as opposed to NvmInit, which is only called once during power on.*
- [blt\\_bool NvmWrite](#) ([blt\\_addr](#) addr, [blt\\_int32u](#) len, [blt\\_int8u](#) \*data)

- Programs the non-volatile memory.*
  - `blt_bool NvmErase (blt_addr addr, blt_int32u len)`
    - Erases the non-volatile memory.*
  - `blt_bool NvmVerifyChecksum (void)`
    - Verifies the checksum, which indicates that a valid user program is present and can be started.*
  - `blt_addr NvmGetUserProgBaseAddress (void)`
    - Obtains the base address of the non-volatile memory available to the user program. This is typically that start of the vector table.*
  - `blt_bool NvmDone (void)`
    - Once all erase and programming operations are completed, this function is called, so at the end of the programming session and right before a software reset is performed. It is used to calculate a checksum and program this into flash. This checksum is later used to determine if a valid user program is present in flash.*

### 7.1121.1 Detailed Description

Bootloader non-volatile memory driver source file.

### 7.1121.2 Function Documentation

#### 7.1121.2.1 NvmDone()

```
blt_bool NvmDone (
 void)
```

Once all erase and programming operations are completed, this function is called, so at the end of the programming session and right before a software reset is performed. It is used to calculate a checksum and program this into flash. This checksum is later used to determine if a valid user program is present in flash.

##### Returns

BLT\_TRUE if successful, BLT\_FALSE otherwise.

#### 7.1121.2.2 NvmDoneHook()

```
blt_bool NvmDoneHook (
 void)
```

Callback that gets called at the end of the NVM programming session.

##### Returns

BLT\_TRUE is successful, BLT\_FALSE otherwise.

Referenced by NvmDone().

#### 7.1121.2.3 NvmErase()

```
blt_bool NvmErase (
 blt_addr addr,
 blt_int32u len)
```

Erases the non-volatile memory.

**Parameters**

|             |                  |
|-------------|------------------|
| <i>addr</i> | Start address.   |
| <i>len</i>  | Length in bytes. |

**Returns**

BLT\_TRUE if successful, BLT\_FALSE otherwise.

**7.1121.2.4 NvmEraseHook()**

```
blt_int8u NvmEraseHook (
 blt_addr addr,
 blt_int32u len)
```

Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the memory hasn't been erased yet.

**Parameters**

|             |                  |
|-------------|------------------|
| <i>addr</i> | Start address.   |
| <i>len</i>  | Length in bytes. |

**Returns**

BLT\_NVM\_OKAY if successful, BLT\_NVM\_NOT\_IN\_RANGE if the address is not within the supported memory range, or BLT\_NVM\_ERROR is the erase operation failed.

Referenced by NvmErase().

**7.1121.2.5 NvmGetUserProgBaseAddress()**

```
blt_addr NvmGetUserProgBaseAddress (
 void)
```

Obtains the base address of the non-volatile memory available to the user program. This is typically that start of the vector table.

**Returns**

Base address.

#### 7.1121.2.6 NvmInit()

```
void NvmInit (
 void)
```

Initializes the NVM driver.

##### Returns

none.

#### 7.1121.2.7 NvmInitHook()

```
void NvmInitHook (
 void)
```

Callback that gets called at the start of the internal NVM driver initialization routine.

##### Returns

none.

Referenced by NvmInit().

#### 7.1121.2.8 NvmReinit()

```
void NvmReinit (
 void)
```

Reinitializes the NVM driver. This function is called at the start of each firmware update as opposed to NvmInit, which is only called once during power on.

##### Returns

none.

#### 7.1121.2.9 NvmReinitHook()

```
void NvmReinitHook (
 void)
```

Callback that gets called at the start of a firmware update to reinitialize the NVM driver.

##### Returns

none.

Referenced by NvmReinit().



### 7.1121.2.10 NvmVerifyChecksum()

```
blt_bool NvmVerifyChecksum (
 void)
```

Verifies the checksum, which indicates that a valid user program is present and can be started.

#### Returns

BLT\_TRUE if successful, BLT\_FALSE otherwise.

### 7.1121.2.11 NvmWrite()

```
blt_bool NvmWrite (
 blt_addr addr,
 blt_int32u len,
 blt_int8u * data)
```

Programs the non-volatile memory.

#### Parameters

|             |                             |
|-------------|-----------------------------|
| <i>addr</i> | Start address.              |
| <i>len</i>  | Length in bytes.            |
| <i>data</i> | Pointer to the data buffer. |

#### Returns

BLT\_TRUE if successful, BLT\_FALSE otherwise.

### 7.1121.2.12 NvmWriteHook()

```
blt_int8u NvmWriteHook (
 blt_addr addr,
 blt_int32u len,
 blt_int8u * data)
```

Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the data hasn't been written yet.

#### Parameters

|             |                             |
|-------------|-----------------------------|
| <i>addr</i> | Start address.              |
| <i>len</i>  | Length in bytes.            |
| <i>data</i> | Pointer to the data buffer. |

## Returns

BLT\_NVM\_OKAY if successful, BLT\_NVM\_NOT\_IN\_RANGE if the address is not within the supported memory range, or BLT\_NVM\_ERROR if the write operation failed.

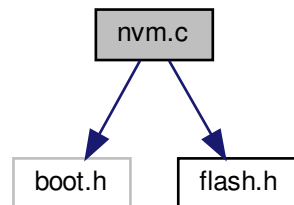
Referenced by NvmWrite().

## 7.1122 nvm.c File Reference

Bootloader non-volatile memory driver source file.

```
#include "boot.h"
#include "flash.h"
```

Include dependency graph for ARMCM4\_TM4C/nvm.c:



## Functions

- void [NvmInitHook](#) (void)  
*Callback that gets called at the start of the internal NVM driver initialization routine.*
- void [NvmReinitHook](#) (void)  
*Callback that gets called at the start of a firmware update to reinitialize the NVM driver.*
- [blt\\_int8u NvmWriteHook](#) ([blt\\_addr](#) addr, [blt\\_int32u](#) len, [blt\\_int8u](#) \*data)  
*Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the data hasn't been written yet.*
- [blt\\_int8u NvmEraseHook](#) ([blt\\_addr](#) addr, [blt\\_int32u](#) len)  
*Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the memory hasn't been erased yet.*
- [blt\\_bool NvmDoneHook](#) (void)  
*Callback that gets called at the end of the NVM programming session.*
- void [NvmInit](#) (void)  
*Initializes the NVM driver.*
- void [NvmReinit](#) (void)  
*Reinitializes the NVM driver. This function is called at the start of each firmware update as opposed to NvmInit, which is only called once during power on.*
- [blt\\_bool NvmWrite](#) ([blt\\_addr](#) addr, [blt\\_int32u](#) len, [blt\\_int8u](#) \*data)

- Programs the non-volatile memory.*
- [blt\\_bool NvmErase](#) ([blt\\_addr](#) addr, [blt\\_int32u](#) len)
- Erases the non-volatile memory.*
- [blt\\_bool NvmVerifyChecksum](#) (void)
- Verifies the checksum, which indicates that a valid user program is present and can be started.*
- [blt\\_addr NvmGetUserProgBaseAddress](#) (void)
- Obtains the base address of the non-volatile memory available to the user program. This is typically that start of the vector table.*
- [blt\\_bool NvmDone](#) (void)
- Once all erase and programming operations are completed, this function is called, so at the end of the programming session and right before a software reset is performed. It is used to calculate a checksum and program this into flash. This checksum is later used to determine if a valid user program is present in flash.*

### 7.1122.1 Detailed Description

Bootloader non-volatile memory driver source file.

### 7.1122.2 Function Documentation

#### 7.1122.2.1 NvmDone()

```
blt_bool NvmDone (
 void)
```

Once all erase and programming operations are completed, this function is called, so at the end of the programming session and right before a software reset is performed. It is used to calculate a checksum and program this into flash. This checksum is later used to determine if a valid user program is present in flash.

##### Returns

BLT\_TRUE if successful, BLT\_FALSE otherwise.

#### 7.1122.2.2 NvmDoneHook()

```
blt_bool NvmDoneHook (
 void)
```

Callback that gets called at the end of the NVM programming session.

##### Returns

BLT\_TRUE is successful, BLT\_FALSE otherwise.

Referenced by [NvmDone\(\)](#).

#### 7.1122.2.3 NvmErase()

```
blt_bool NvmErase (
 blt_addr addr,
 blt_int32u len)
```

Erases the non-volatile memory.

**Parameters**

|             |                  |
|-------------|------------------|
| <i>addr</i> | Start address.   |
| <i>len</i>  | Length in bytes. |

**Returns**

BLT\_TRUE if successful, BLT\_FALSE otherwise.

**7.1122.2.4 NvmEraseHook()**

```
blt_int8u NvmEraseHook (
 blt_addr addr,
 blt_int32u len)
```

Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the memory hasn't been erased yet.

**Parameters**

|             |                  |
|-------------|------------------|
| <i>addr</i> | Start address.   |
| <i>len</i>  | Length in bytes. |

**Returns**

BLT\_NVM\_OKAY if successful, BLT\_NVM\_NOT\_IN\_RANGE if the address is not within the supported memory range, or BLT\_NVM\_ERROR is the erase operation failed.

Referenced by NvmErase().

**7.1122.2.5 NvmGetUserProgBaseAddress()**

```
blt_addr NvmGetUserProgBaseAddress (
 void)
```

Obtains the base address of the non-volatile memory available to the user program. This is typically that start of the vector table.

**Returns**

Base address.

#### 7.1122.2.6 NvmInit()

```
void NvmInit (
 void)
```

Initializes the NVM driver.

##### Returns

none.

#### 7.1122.2.7 NvmInitHook()

```
void NvmInitHook (
 void)
```

Callback that gets called at the start of the internal NVM driver initialization routine.

##### Returns

none.

Referenced by NvmInit().

#### 7.1122.2.8 NvmReinit()

```
void NvmReinit (
 void)
```

Reinitializes the NVM driver. This function is called at the start of each firmware update as opposed to NvmInit, which is only called once during power on.

##### Returns

none.

#### 7.1122.2.9 NvmReinitHook()

```
void NvmReinitHook (
 void)
```

Callback that gets called at the start of a firmware update to reinitialize the NVM driver.

##### Returns

none.

Referenced by NvmReinit().

#### 7.1122.2.10 NvmVerifyChecksum()

```
blt_bool NvmVerifyChecksum (
 void)
```

Verifies the checksum, which indicates that a valid user program is present and can be started.

##### Returns

BLT\_TRUE if successful, BLT\_FALSE otherwise.

#### 7.1122.2.11 NvmWrite()

```
blt_bool NvmWrite (
 blt_addr addr,
 blt_int32u len,
 blt_int8u * data)
```

Programs the non-volatile memory.

##### Parameters

|             |                             |
|-------------|-----------------------------|
| <i>addr</i> | Start address.              |
| <i>len</i>  | Length in bytes.            |
| <i>data</i> | Pointer to the data buffer. |

##### Returns

BLT\_TRUE if successful, BLT\_FALSE otherwise.

#### 7.1122.2.12 NvmWriteHook()

```
blt_int8u NvmWriteHook (
 blt_addr addr,
 blt_int32u len,
 blt_int8u * data)
```

Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the data hasn't been written yet.

##### Parameters

|             |                             |
|-------------|-----------------------------|
| <i>addr</i> | Start address.              |
| <i>len</i>  | Length in bytes.            |
| <i>data</i> | Pointer to the data buffer. |

## Returns

BLT\_NVM\_OKAY if successful, BLT\_NVM\_NOT\_IN\_RANGE if the address is not within the supported memory range, or BLT\_NVM\_ERROR if the write operation failed.

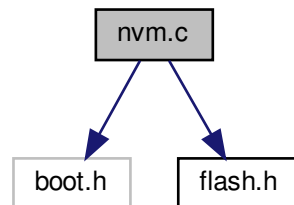
Referenced by NvmWrite().

## 7.1123 nvm.c File Reference

Bootloader non-volatile memory driver source file.

```
#include "boot.h"
#include "flash.h"
```

Include dependency graph for ARMCM4\_XMC4/nvm.c:



## Functions

- void [NvmInitHook](#) (void)  
*Callback that gets called at the start of the internal NVM driver initialization routine.*
- void [NvmReinitHook](#) (void)  
*Callback that gets called at the start of a firmware update to reinitialize the NVM driver.*
- [blt\\_int8u NvmWriteHook](#) ([blt\\_addr](#) addr, [blt\\_int32u](#) len, [blt\\_int8u](#) \*data)  
*Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the data hasn't been written yet.*
- [blt\\_int8u NvmEraseHook](#) ([blt\\_addr](#) addr, [blt\\_int32u](#) len)  
*Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the memory hasn't been erased yet.*
- [blt\\_bool NvmDoneHook](#) (void)  
*Callback that gets called at the end of the NVM programming session.*
- void [NvmInit](#) (void)  
*Initializes the NVM driver.*
- void [NvmReinit](#) (void)  
*Reinitializes the NVM driver. This function is called at the start of each firmware update as opposed to NvmInit, which is only called once during power on.*
- [blt\\_bool NvmWrite](#) ([blt\\_addr](#) addr, [blt\\_int32u](#) len, [blt\\_int8u](#) \*data)

- Programs the non-volatile memory.*
- [blt\\_bool NvmErase](#) ([blt\\_addr](#) addr, [blt\\_int32u](#) len)
- Erases the non-volatile memory.*
- [blt\\_bool NvmVerifyChecksum](#) (void)
- Verifies the checksum, which indicates that a valid user program is present and can be started.*
- [blt\\_addr NvmGetUserProgBaseAddress](#) (void)
- Obtains the base address of the non-volatile memory available to the user program. This is typically that start of the vector table.*
- [blt\\_bool NvmDone](#) (void)
- Once all erase and programming operations are completed, this function is called, so at the end of the programming session and right before a software reset is performed. It is used to calculate a checksum and program this into flash. This checksum is later used to determine if a valid user program is present in flash.*

### 7.1123.1 Detailed Description

Bootloader non-volatile memory driver source file.

### 7.1123.2 Function Documentation

#### 7.1123.2.1 NvmDone()

```
blt_bool NvmDone (
 void)
```

Once all erase and programming operations are completed, this function is called, so at the end of the programming session and right before a software reset is performed. It is used to calculate a checksum and program this into flash. This checksum is later used to determine if a valid user program is present in flash.

##### Returns

BLT\_TRUE if successful, BLT\_FALSE otherwise.

#### 7.1123.2.2 NvmDoneHook()

```
blt_bool NvmDoneHook (
 void)
```

Callback that gets called at the end of the NVM programming session.

##### Returns

BLT\_TRUE is successful, BLT\_FALSE otherwise.

Referenced by [NvmDone\(\)](#).

#### 7.1123.2.3 NvmErase()

```
blt_bool NvmErase (
 blt_addr addr,
 blt_int32u len)
```

Erases the non-volatile memory.



**Parameters**

|             |                  |
|-------------|------------------|
| <i>addr</i> | Start address.   |
| <i>len</i>  | Length in bytes. |

**Returns**

BLT\_TRUE if successful, BLT\_FALSE otherwise.

**7.1123.2.4 NvmEraseHook()**

```
blt_int8u NvmEraseHook (
 blt_addr addr,
 blt_int32u len)
```

Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the memory hasn't been erased yet.

**Parameters**

|             |                  |
|-------------|------------------|
| <i>addr</i> | Start address.   |
| <i>len</i>  | Length in bytes. |

**Returns**

BLT\_NVM\_OKAY if successful, BLT\_NVM\_NOT\_IN\_RANGE if the address is not within the supported memory range, or BLT\_NVM\_ERROR is the erase operation failed.

Referenced by NvmErase().

**7.1123.2.5 NvmGetUserProgBaseAddress()**

```
blt_addr NvmGetUserProgBaseAddress (
 void)
```

Obtains the base address of the non-volatile memory available to the user program. This is typically that start of the vector table.

**Returns**

Base address.

**7.1123.2.6 NvmInit()**

```
void NvmInit (
 void)
```

Initializes the NVM driver.

**Returns**

none.

**7.1123.2.7 NvmInitHook()**

```
void NvmInitHook (
 void)
```

Callback that gets called at the start of the internal NVM driver initialization routine.

**Returns**

none.

Referenced by NvmInit().

**7.1123.2.8 NvmReinit()**

```
void NvmReinit (
 void)
```

Reinitializes the NVM driver. This function is called at the start of each firmware update as opposed to NvmInit, which is only called once during power on.

**Returns**

none.

**7.1123.2.9 NvmReinitHook()**

```
void NvmReinitHook (
 void)
```

Callback that gets called at the start of a firmware update to reinitialize the NVM driver.

**Returns**

none.

Referenced by NvmReinit().

### 7.1123.2.10 NvmVerifyChecksum()

```
blt_bool NvmVerifyChecksum (
 void)
```

Verifies the checksum, which indicates that a valid user program is present and can be started.

#### Returns

BLT\_TRUE if successful, BLT\_FALSE otherwise.

### 7.1123.2.11 NvmWrite()

```
blt_bool NvmWrite (
 blt_addr addr,
 blt_int32u len,
 blt_int8u * data)
```

Programs the non-volatile memory.

#### Parameters

|             |                             |
|-------------|-----------------------------|
| <i>addr</i> | Start address.              |
| <i>len</i>  | Length in bytes.            |
| <i>data</i> | Pointer to the data buffer. |

#### Returns

BLT\_TRUE if successful, BLT\_FALSE otherwise.

### 7.1123.2.12 NvmWriteHook()

```
blt_int8u NvmWriteHook (
 blt_addr addr,
 blt_int32u len,
 blt_int8u * data)
```

Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the data hasn't been written yet.

#### Parameters

|             |                             |
|-------------|-----------------------------|
| <i>addr</i> | Start address.              |
| <i>len</i>  | Length in bytes.            |
| <i>data</i> | Pointer to the data buffer. |

## Returns

BLT\_NVM\_OKAY if successful, BLT\_NVM\_NOT\_IN\_RANGE if the address is not within the supported memory range, or BLT\_NVM\_ERROR if the write operation failed.

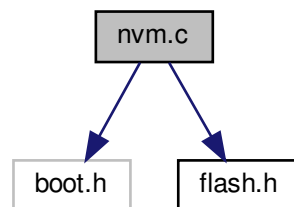
Referenced by NvmWrite().

## 7.1124 nvm.c File Reference

Bootloader non-volatile memory driver source file.

```
#include "boot.h"
#include "flash.h"
```

Include dependency graph for ARMCM7\_STM32F7/nvm.c:



## Functions

- void [NvmInitHook](#) (void)  
*Callback that gets called at the start of the internal NVM driver initialization routine.*
- void [NvmReinitHook](#) (void)  
*Callback that gets called at the start of a firmware update to reinitialize the NVM driver.*
- [blt\\_int8u NvmWriteHook](#) ([blt\\_addr](#) addr, [blt\\_int32u](#) len, [blt\\_int8u](#) \*data)  
*Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the data hasn't been written yet.*
- [blt\\_int8u NvmEraseHook](#) ([blt\\_addr](#) addr, [blt\\_int32u](#) len)  
*Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the memory hasn't been erased yet.*
- [blt\\_bool NvmDoneHook](#) (void)  
*Callback that gets called at the end of the NVM programming session.*
- void [NvmInit](#) (void)  
*Initializes the NVM driver.*
- void [NvmReinit](#) (void)  
*Reinitializes the NVM driver. This function is called at the start of each firmware update as opposed to NvmInit, which is only called once during power on.*
- [blt\\_bool NvmWrite](#) ([blt\\_addr](#) addr, [blt\\_int32u](#) len, [blt\\_int8u](#) \*data)

- Programs the non-volatile memory.*
- [blt\\_bool NvmErase](#) ([blt\\_addr](#) addr, [blt\\_int32u](#) len)
- Erases the non-volatile memory.*
- [blt\\_bool NvmVerifyChecksum](#) (void)
- Verifies the checksum, which indicates that a valid user program is present and can be started.*
- [blt\\_addr NvmGetUserProgBaseAddress](#) (void)
- Obtains the base address of the non-volatile memory available to the user program. This is typically that start of the vector table.*
- [blt\\_bool NvmDone](#) (void)
- Once all erase and programming operations are completed, this function is called, so at the end of the programming session and right before a software reset is performed. It is used to calculate a checksum and program this into flash. This checksum is later used to determine if a valid user program is present in flash.*

### 7.1124.1 Detailed Description

Bootloader non-volatile memory driver source file.

### 7.1124.2 Function Documentation

#### 7.1124.2.1 NvmDone()

```
blt_bool NvmDone (
 void)
```

Once all erase and programming operations are completed, this function is called, so at the end of the programming session and right before a software reset is performed. It is used to calculate a checksum and program this into flash. This checksum is later used to determine if a valid user program is present in flash.

##### Returns

BLT\_TRUE if successful, BLT\_FALSE otherwise.

#### 7.1124.2.2 NvmDoneHook()

```
blt_bool NvmDoneHook (
 void)
```

Callback that gets called at the end of the NVM programming session.

##### Returns

BLT\_TRUE is successful, BLT\_FALSE otherwise.

Referenced by [NvmDone\(\)](#).

#### 7.1124.2.3 NvmErase()

```
blt_bool NvmErase (
 blt_addr addr,
 blt_int32u len)
```

Erases the non-volatile memory.

**Parameters**

|             |                  |
|-------------|------------------|
| <i>addr</i> | Start address.   |
| <i>len</i>  | Length in bytes. |

**Returns**

BLT\_TRUE if successful, BLT\_FALSE otherwise.

**7.1124.2.4 NvmEraseHook()**

```
blt_int8u NvmEraseHook (
 blt_addr addr,
 blt_int32u len)
```

Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the memory hasn't been erased yet.

**Parameters**

|             |                  |
|-------------|------------------|
| <i>addr</i> | Start address.   |
| <i>len</i>  | Length in bytes. |

**Returns**

BLT\_NVM\_OKAY if successful, BLT\_NVM\_NOT\_IN\_RANGE if the address is not within the supported memory range, or BLT\_NVM\_ERROR is the erase operation failed.

Referenced by NvmErase().

**7.1124.2.5 NvmGetUserProgBaseAddress()**

```
blt_addr NvmGetUserProgBaseAddress (
 void)
```

Obtains the base address of the non-volatile memory available to the user program. This is typically that start of the vector table.

**Returns**

Base address.

#### 7.1124.2.6 NvmInit()

```
void NvmInit (
 void)
```

Initializes the NVM driver.

##### Returns

none.

#### 7.1124.2.7 NvmInitHook()

```
void NvmInitHook (
 void)
```

Callback that gets called at the start of the internal NVM driver initialization routine.

##### Returns

none.

Referenced by NvmInit().

#### 7.1124.2.8 NvmReinit()

```
void NvmReinit (
 void)
```

Reinitializes the NVM driver. This function is called at the start of each firmware update as opposed to NvmInit, which is only called once during power on.

##### Returns

none.

#### 7.1124.2.9 NvmReinitHook()

```
void NvmReinitHook (
 void)
```

Callback that gets called at the start of a firmware update to reinitialize the NVM driver.

##### Returns

none.

Referenced by NvmReinit().

#### 7.1124.2.10 NvmVerifyChecksum()

```
blt_bool NvmVerifyChecksum (
 void)
```

Verifies the checksum, which indicates that a valid user program is present and can be started.

##### Returns

BLT\_TRUE if successful, BLT\_FALSE otherwise.

#### 7.1124.2.11 NvmWrite()

```
blt_bool NvmWrite (
 blt_addr addr,
 blt_int32u len,
 blt_int8u * data)
```

Programs the non-volatile memory.

##### Parameters

|             |                             |
|-------------|-----------------------------|
| <i>addr</i> | Start address.              |
| <i>len</i>  | Length in bytes.            |
| <i>data</i> | Pointer to the data buffer. |

##### Returns

BLT\_TRUE if successful, BLT\_FALSE otherwise.

#### 7.1124.2.12 NvmWriteHook()

```
blt_int8u NvmWriteHook (
 blt_addr addr,
 blt_int32u len,
 blt_int8u * data)
```

Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the data hasn't been written yet.

##### Parameters

|             |                             |
|-------------|-----------------------------|
| <i>addr</i> | Start address.              |
| <i>len</i>  | Length in bytes.            |
| <i>data</i> | Pointer to the data buffer. |



**Returns**

BLT\_NVM\_OKAY if successful, BLT\_NVM\_NOT\_IN\_RANGE if the address is not within the supported memory range, or BLT\_NVM\_ERROR if the write operation failed.

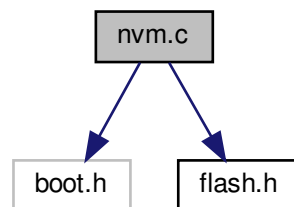
Referenced by NvmWrite().

**7.1125 nvm.c File Reference**

Bootloader non-volatile memory driver source file.

```
#include "boot.h"
#include "flash.h"
```

Include dependency graph for ARMCM7\_STM32H7/nvm.c:

**Functions**

- void [NvmInitHook](#) (void)  
*Callback that gets called at the start of the internal NVM driver initialization routine.*
- void [NvmReinitHook](#) (void)  
*Callback that gets called at the start of a firmware update to reinitialize the NVM driver.*
- [blt\\_int8u NvmWriteHook](#) ([blt\\_addr](#) addr, [blt\\_int32u](#) len, [blt\\_int8u](#) \*data)  
*Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the data hasn't been written yet.*
- [blt\\_int8u NvmEraseHook](#) ([blt\\_addr](#) addr, [blt\\_int32u](#) len)  
*Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the memory hasn't been erased yet.*
- [blt\\_bool NvmDoneHook](#) (void)  
*Callback that gets called at the end of the NVM programming session.*
- void [NvmInit](#) (void)  
*Initializes the NVM driver.*
- void [NvmReinit](#) (void)  
*Reinitializes the NVM driver. This function is called at the start of each firmware update as opposed to NvmInit, which is only called once during power on.*
- [blt\\_bool NvmWrite](#) ([blt\\_addr](#) addr, [blt\\_int32u](#) len, [blt\\_int8u](#) \*data)

- Programs the non-volatile memory.*
  - `blt_bool NvmErase (blt_addr addr, blt_int32u len)`
    - Erases the non-volatile memory.*
  - `blt_bool NvmVerifyChecksum (void)`
    - Verifies the checksum, which indicates that a valid user program is present and can be started.*
  - `blt_addr NvmGetUserProgBaseAddress (void)`
    - Obtains the base address of the non-volatile memory available to the user program. This is typically that start of the vector table.*
  - `blt_bool NvmDone (void)`
    - Once all erase and programming operations are completed, this function is called, so at the end of the programming session and right before a software reset is performed. It is used to calculate a checksum and program this into flash. This checksum is later used to determine if a valid user program is present in flash.*

### 7.1125.1 Detailed Description

Bootloader non-volatile memory driver source file.

### 7.1125.2 Function Documentation

#### 7.1125.2.1 NvmDone()

```
blt_bool NvmDone (
 void)
```

Once all erase and programming operations are completed, this function is called, so at the end of the programming session and right before a software reset is performed. It is used to calculate a checksum and program this into flash. This checksum is later used to determine if a valid user program is present in flash.

##### Returns

BLT\_TRUE if successful, BLT\_FALSE otherwise.

#### 7.1125.2.2 NvmDoneHook()

```
blt_bool NvmDoneHook (
 void)
```

Callback that gets called at the end of the NVM programming session.

##### Returns

BLT\_TRUE is successful, BLT\_FALSE otherwise.

Referenced by NvmDone().

#### 7.1125.2.3 NvmErase()

```
blt_bool NvmErase (
 blt_addr addr,
 blt_int32u len)
```

Erases the non-volatile memory.

**Parameters**

|             |                  |
|-------------|------------------|
| <i>addr</i> | Start address.   |
| <i>len</i>  | Length in bytes. |

**Returns**

BLT\_TRUE if successful, BLT\_FALSE otherwise.

**7.1125.2.4 NvmEraseHook()**

```
blt_int8u NvmEraseHook (
 blt_addr addr,
 blt_int32u len)
```

Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the memory hasn't been erased yet.

**Parameters**

|             |                  |
|-------------|------------------|
| <i>addr</i> | Start address.   |
| <i>len</i>  | Length in bytes. |

**Returns**

BLT\_NVM\_OKAY if successful, BLT\_NVM\_NOT\_IN\_RANGE if the address is not within the supported memory range, or BLT\_NVM\_ERROR is the erase operation failed.

Referenced by NvmErase().

**7.1125.2.5 NvmGetUserProgBaseAddress()**

```
blt_addr NvmGetUserProgBaseAddress (
 void)
```

Obtains the base address of the non-volatile memory available to the user program. This is typically that start of the vector table.

**Returns**

Base address.

**7.1125.2.6 NvmInit()**

```
void NvmInit (
 void)
```

Initializes the NVM driver.

**Returns**

none.

**7.1125.2.7 NvmInitHook()**

```
void NvmInitHook (
 void)
```

Callback that gets called at the start of the internal NVM driver initialization routine.

**Returns**

none.

Referenced by NvmInit().

**7.1125.2.8 NvmReinit()**

```
void NvmReinit (
 void)
```

Reinitializes the NVM driver. This function is called at the start of each firmware update as opposed to NvmInit, which is only called once during power on.

**Returns**

none.

**7.1125.2.9 NvmReinitHook()**

```
void NvmReinitHook (
 void)
```

Callback that gets called at the start of a firmware update to reinitialize the NVM driver.

**Returns**

none.

Referenced by NvmReinit().

### 7.1125.2.10 NvmVerifyChecksum()

```
blt_bool NvmVerifyChecksum (
 void)
```

Verifies the checksum, which indicates that a valid user program is present and can be started.

#### Returns

BLT\_TRUE if successful, BLT\_FALSE otherwise.

### 7.1125.2.11 NvmWrite()

```
blt_bool NvmWrite (
 blt_addr addr,
 blt_int32u len,
 blt_int8u * data)
```

Programs the non-volatile memory.

#### Parameters

|             |                             |
|-------------|-----------------------------|
| <i>addr</i> | Start address.              |
| <i>len</i>  | Length in bytes.            |
| <i>data</i> | Pointer to the data buffer. |

#### Returns

BLT\_TRUE if successful, BLT\_FALSE otherwise.

### 7.1125.2.12 NvmWriteHook()

```
blt_int8u NvmWriteHook (
 blt_addr addr,
 blt_int32u len,
 blt_int8u * data)
```

Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the data hasn't been written yet.

#### Parameters

|             |                             |
|-------------|-----------------------------|
| <i>addr</i> | Start address.              |
| <i>len</i>  | Length in bytes.            |
| <i>data</i> | Pointer to the data buffer. |

## Returns

BLT\_NVM\_OKAY if successful, BLT\_NVM\_NOT\_IN\_RANGE if the address is not within the supported memory range, or BLT\_NVM\_ERROR if the write operation failed.

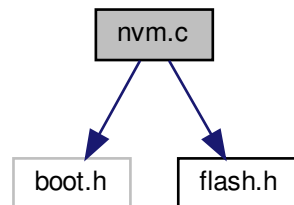
Referenced by NvmWrite().

## 7.1126 nvm.c File Reference

Bootloader non-volatile memory driver source file.

```
#include "boot.h"
#include "flash.h"
```

Include dependency graph for HCS12/nvm.c:



## Functions

- void [NvmInitHook](#) (void)  
*Callback that gets called at the start of the internal NVM driver initialization routine.*
- void [NvmReinitHook](#) (void)  
*Callback that gets called at the start of a firmware update to reinitialize the NVM driver.*
- [blt\\_int8u NvmWriteHook](#) ([blt\\_addr](#) addr, [blt\\_int32u](#) len, [blt\\_int8u](#) \*data)  
*Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the data hasn't been written yet.*
- [blt\\_int8u NvmEraseHook](#) ([blt\\_addr](#) addr, [blt\\_int32u](#) len)  
*Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the memory hasn't been erased yet.*
- [blt\\_bool NvmDoneHook](#) (void)  
*Callback that gets called at the end of the NVM programming session.*
- void [NvmInit](#) (void)  
*Initializes the NVM driver.*
- void [NvmReinit](#) (void)  
*Reinitializes the NVM driver. This function is called at the start of each firmware update as opposed to NvmInit, which is only called once during power on.*
- [blt\\_bool NvmWrite](#) ([blt\\_addr](#) addr, [blt\\_int32u](#) len, [blt\\_int8u](#) \*data)

- Programs the non-volatile memory.*
- [blt\\_bool NvmErase](#) ([blt\\_addr](#) addr, [blt\\_int32u](#) len)
- Erases the non-volatile memory.*
- [blt\\_bool NvmVerifyChecksum](#) (void)
- Verifies the checksum, which indicates that a valid user program is present and can be started.*
- [blt\\_addr NvmGetUserProgBaseAddress](#) (void)
- Obtains the base address of the non-volatile memory available to the user program. This is typically that start of the vector table.*
- [blt\\_bool NvmDone](#) (void)
- Once all erase and programming operations are completed, this function is called, so at the end of the programming session and right before a software reset is performed. It is used to calculate a checksum and program this into flash. This checksum is later used to determine if a valid user program is present in flash.*

## 7.1126.1 Detailed Description

Bootloader non-volatile memory driver source file.

## 7.1126.2 Function Documentation

### 7.1126.2.1 NvmDone()

```
blt_bool NvmDone (
 void)
```

Once all erase and programming operations are completed, this function is called, so at the end of the programming session and right before a software reset is performed. It is used to calculate a checksum and program this into flash. This checksum is later used to determine if a valid user program is present in flash.

#### Returns

BLT\_TRUE if successful, BLT\_FALSE otherwise.

### 7.1126.2.2 NvmDoneHook()

```
blt_bool NvmDoneHook (
 void)
```

Callback that gets called at the end of the NVM programming session.

#### Returns

BLT\_TRUE is successful, BLT\_FALSE otherwise.

Referenced by [NvmDone\(\)](#).

### 7.1126.2.3 NvmErase()

```
blt_bool NvmErase (
 blt_addr addr,
 blt_int32u len)
```

Erases the non-volatile memory.

**Parameters**

|             |                  |
|-------------|------------------|
| <i>addr</i> | Start address.   |
| <i>len</i>  | Length in bytes. |

**Returns**

BLT\_TRUE if successful, BLT\_FALSE otherwise.

**7.1126.2.4 NvmEraseHook()**

```
blt_int8u NvmEraseHook (
 blt_addr addr,
 blt_int32u len)
```

Callback that gets called at the start of the NVM driver erase routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the memory hasn't been erased yet.

**Parameters**

|             |                  |
|-------------|------------------|
| <i>addr</i> | Start address.   |
| <i>len</i>  | Length in bytes. |

**Returns**

BLT\_NVM\_OKAY if successful, BLT\_NVM\_NOT\_IN\_RANGE if the address is not within the supported memory range, or BLT\_NVM\_ERROR is the erase operation failed.

Referenced by NvmErase().

**7.1126.2.5 NvmGetUserProgBaseAddress()**

```
blt_addr NvmGetUserProgBaseAddress (
 void)
```

Obtains the base address of the non-volatile memory available to the user program. This is typically that start of the vector table.

**Returns**

Base address.



### 7.1126.2.6 NvmInit()

```
void NvmInit (
 void)
```

Initializes the NVM driver.

#### Returns

none.

### 7.1126.2.7 NvmInitHook()

```
void NvmInitHook (
 void)
```

Callback that gets called at the start of the internal NVM driver initialization routine.

#### Returns

none.

Referenced by NvmInit().

### 7.1126.2.8 NvmReinit()

```
void NvmReinit (
 void)
```

Reinitializes the NVM driver. This function is called at the start of each firmware update as opposed to NvmInit, which is only called once during power on.

#### Returns

none.

### 7.1126.2.9 NvmReinitHook()

```
void NvmReinitHook (
 void)
```

Callback that gets called at the start of a firmware update to reinitialize the NVM driver.

#### Returns

none.

Referenced by NvmReinit().

#### 7.1126.2.10 NvmVerifyChecksum()

```
blt_bool NvmVerifyChecksum (
 void)
```

Verifies the checksum, which indicates that a valid user program is present and can be started.

##### Returns

BLT\_TRUE if successful, BLT\_FALSE otherwise.

#### 7.1126.2.11 NvmWrite()

```
blt_bool NvmWrite (
 blt_addr addr,
 blt_int32u len,
 blt_int8u * data)
```

Programs the non-volatile memory.

##### Parameters

|             |                             |
|-------------|-----------------------------|
| <i>addr</i> | Start address.              |
| <i>len</i>  | Length in bytes.            |
| <i>data</i> | Pointer to the data buffer. |

##### Returns

BLT\_TRUE if successful, BLT\_FALSE otherwise.

#### 7.1126.2.12 NvmWriteHook()

```
blt_int8u NvmWriteHook (
 blt_addr addr,
 blt_int32u len,
 blt_int8u * data)
```

Callback that gets called at the start of the NVM driver write routine. It allows additional memory to be operated on. If the address is not within the range of the additional memory, then BLT\_NVM\_NOT\_IN\_RANGE must be returned to indicate that the data hasn't been written yet.

##### Parameters

|             |                             |
|-------------|-----------------------------|
| <i>addr</i> | Start address.              |
| <i>len</i>  | Length in bytes.            |
| <i>data</i> | Pointer to the data buffer. |

### Returns

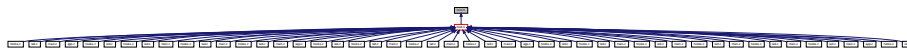
BLT\_NVM\_OKAY if successful, BLT\_NVM\_NOT\_IN\_RANGE if the address is not within the supported memory range, or BLT\_NVM\_ERROR if the write operation failed.

Referenced by NvmWrite().

## 7.1127 nvm.h File Reference

Bootloader non-volatile memory driver header file.

This graph shows which files directly or indirectly include this file:



### Macros

- `#define BLT_NVM_ERROR (0x00)`  
*Return code for success.*
- `#define BLT_NVM_OKAY (0x01)`  
*Return code for error.*
- `#define BLT_NVM_NOT_IN_RANGE (0x02)`  
*Return code for not in range.*

### Functions

- `void NvmInit (void)`  
*Initializes the NVM driver.*
- `void NvmReinit (void)`  
*Reinitializes the NVM driver. This function is called at the start of each firmware update as opposed to NvmInit, which is only called once during power on.*
- `blt_bool NvmWrite (blt_addr addr, blt_int32u len, blt_int8u *data)`  
*Programs the non-volatile memory.*
- `blt_bool NvmErase (blt_addr addr, blt_int32u len)`  
*Erases the non-volatile memory.*
- `blt_bool NvmVerifyChecksum (void)`  
*Verifies the checksum, which indicates that a valid user program is present and can be started.*
- `blt_addr NvmGetUserProgBaseAddress (void)`  
*Obtains the base address of the non-volatile memory available to the user program. This is typically that start of the vector table.*
- `blt_bool NvmDone (void)`  
*Once all erase and programming operations are completed, this function is called, so at the end of the programming session and right before a software reset is performed. It is used to calculate a checksum and program this into flash. This checksum is later used to determine if a valid user program is present in flash.*

### 7.1127.1 Detailed Description

Bootloader non-volatile memory driver header file.

## 7.1127.2 Function Documentation

### 7.1127.2.1 NvmDone()

```
blt_bool NvmDone (
 void)
```

Once all erase and programming operations are completed, this function is called, so at the end of the programming session and right before a software reset is performed. It is used to calculate a checksum and program this into flash. This checksum is later used to determine if a valid user program is present in flash.

#### Returns

BLT\_TRUE if successful, BLT\_FALSE otherwise.

Referenced by FileTask(), and XcpCmdProgram().

### 7.1127.2.2 NvmErase()

```
blt_bool NvmErase (
 blt_addr addr,
 blt_int32u len)
```

Erases the non-volatile memory.

#### Parameters

|             |                  |
|-------------|------------------|
| <i>addr</i> | Start address.   |
| <i>len</i>  | Length in bytes. |

#### Returns

BLT\_TRUE if successful, BLT\_FALSE otherwise.

Referenced by FileTask(), and XcpCmdProgramClear().

### 7.1127.2.3 NvmGetUserProgBaseAddress()

```
blt_addr NvmGetUserProgBaseAddress (
 void)
```

Obtains the base address of the non-volatile memory available to the user program. This is typically that start of the vector table.

**Returns**

Base address.

Referenced by CpuStartUserProgram().

**7.1127.2.4 NvmInit()**

```
void NvmInit (
 void)
```

Initializes the NVM driver.

**Returns**

none.

Referenced by BootInit(), FileTask(), and XcpCmdConnect().

**7.1127.2.5 NvmReinit()**

```
void NvmReinit (
 void)
```

Reinitializes the NVM driver. This function is called at the start of each firmware update as opposed to NvmInit, which is only called once during power on.

**Returns**

none.

**7.1127.2.6 NvmVerifyChecksum()**

```
blt_bool NvmVerifyChecksum (
 void)
```

Verifies the checksum, which indicates that a valid user program is present and can be started.

**Returns**

BLT\_TRUE if successful, BLT\_FALSE otherwise.

Referenced by CpuStartUserProgram().

**7.1127.2.7 NvmWrite()**

```
blt_bool NvmWrite (
 blt_addr addr,
 blt_int32u len,
 blt_int8u * data)
```

Programs the non-volatile memory.

**Parameters**

|             |                             |
|-------------|-----------------------------|
| <i>addr</i> | Start address.              |
| <i>len</i>  | Length in bytes.            |
| <i>data</i> | Pointer to the data buffer. |

**Returns**

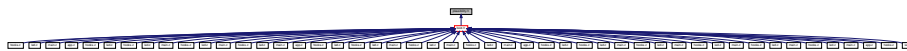
BLT\_TRUE if successful, BLT\_FALSE otherwise.

Referenced by FileTask(), XcpCmdProgram(), and XcpCmdProgramMax().

**7.1128 plausibility.h File Reference**

Bootloader plausibility check header file, for checking the configuration at compile time.

This graph shows which files directly or indirectly include this file:

**7.1128.1 Detailed Description**

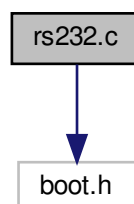
Bootloader plausibility check header file, for checking the configuration at compile time.

**7.1129 rs232.c File Reference**

Bootloader RS232 communication interface source file.

```
#include "boot.h"
```

Include dependency graph for \_template/rs232.c:



### 7.1129.1 Detailed Description

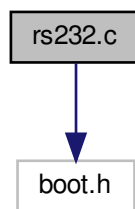
Bootloader RS232 communication interface source file.

## 7.1130 rs232.c File Reference

Bootloader RS232 communication interface source file.

```
#include "boot.h"
```

Include dependency graph for ARMCM0\_STM32F0/rs232.c:



### 7.1130.1 Detailed Description

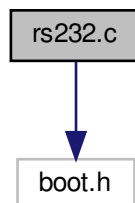
Bootloader RS232 communication interface source file.

## 7.1131 rs232.c File Reference

Bootloader RS232 communication interface source file.

```
#include "boot.h"
```

Include dependency graph for ARMCM0\_STM32G0/rs232.c:



### 7.1131.1 Detailed Description

Bootloader RS232 communication interface source file.

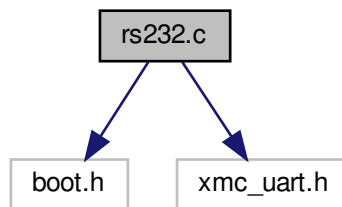
## 7.1132 rs232.c File Reference

Bootloader RS232 communication interface source file.

```
#include "boot.h"
```

```
#include "xmc_uart.h"
```

Include dependency graph for ARMCM0\_XMC1/rs232.c:



### 7.1132.1 Detailed Description

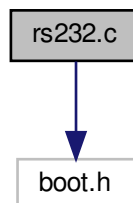
Bootloader RS232 communication interface source file.

## 7.1133 rs232.c File Reference

Bootloader RS232 communication interface source file.

```
#include "boot.h"
```

Include dependency graph for ARMCM33\_STM32L5/rs232.c:





### 7.1133.1 Detailed Description

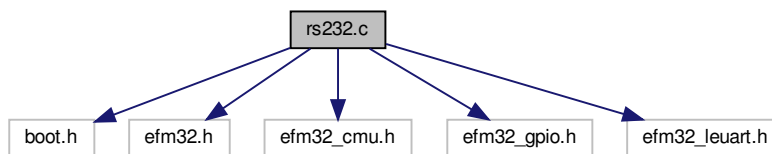
Bootloader RS232 communication interface source file.

## 7.1134 rs232.c File Reference

Bootloader RS232 communication interface source file.

```
#include "boot.h"
#include "efm32.h"
#include "efm32_cmu.h"
#include "efm32_gpio.h"
#include "efm32_leuart.h"
```

Include dependency graph for ARMCM3\_EFM32/rs232.c:



### 7.1134.1 Detailed Description

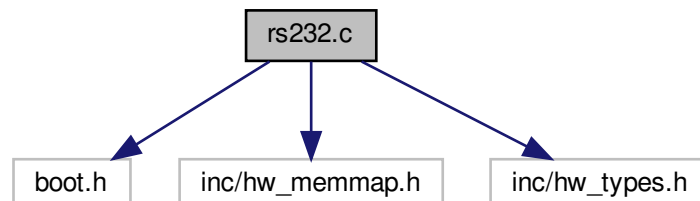
Bootloader RS232 communication interface source file.

## 7.1135 rs232.c File Reference

Bootloader RS232 communication interface source file.

```
#include "boot.h"
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
```

Include dependency graph for ARMCM3\_LM3S/rs232.c:



### 7.1135.1 Detailed Description

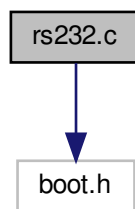
Bootloader RS232 communication interface source file.

## 7.1136 rs232.c File Reference

Bootloader RS232 communication interface source file.

```
#include "boot.h"
```

Include dependency graph for ARMCM3\_STM32F1/rs232.c:



### 7.1136.1 Detailed Description

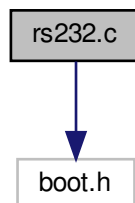
Bootloader RS232 communication interface source file.

## 7.1137 rs232.c File Reference

Bootloader RS232 communication interface source file.

```
#include "boot.h"
```

Include dependency graph for ARMCM3\_STM32F2/rs232.c:



### 7.1137.1 Detailed Description

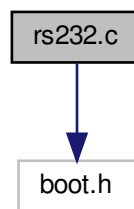
Bootloader RS232 communication interface source file.

## 7.1138 rs232.c File Reference

Bootloader RS232 communication interface source file.

```
#include "boot.h"
```

Include dependency graph for ARMCM4\_STM32F3/rs232.c:



### 7.1138.1 Detailed Description

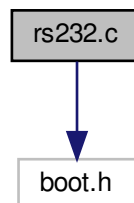
Bootloader RS232 communication interface source file.

## 7.1139 rs232.c File Reference

Bootloader RS232 communication interface source file.

```
#include "boot.h"
```

Include dependency graph for ARMCM4\_STM32F4/rs232.c:



### 7.1139.1 Detailed Description

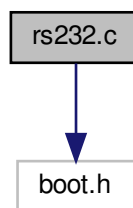
Bootloader RS232 communication interface source file.

### 7.1140 rs232.c File Reference

Bootloader RS232 communication interface source file.

```
#include "boot.h"
```

Include dependency graph for ARMC4\_STM32L4/rs232.c:



### 7.1140.1 Detailed Description

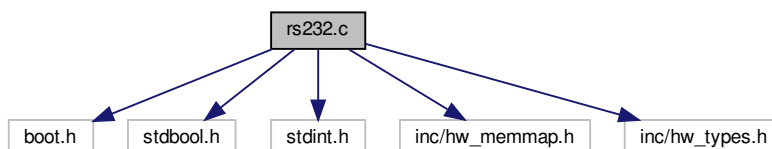
Bootloader RS232 communication interface source file.

### 7.1141 rs232.c File Reference

Bootloader RS232 communication interface source file.

```
#include "boot.h"
#include <stdbool.h>
#include <stdint.h>
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
```

Include dependency graph for ARMC4\_TM4C/rs232.c:



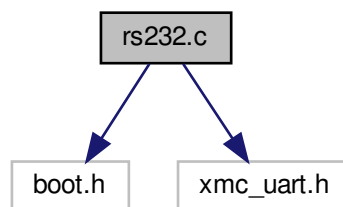
### 7.1141.1 Detailed Description

Bootloader RS232 communication interface source file.

## 7.1142 rs232.c File Reference

Bootloader RS232 communication interface source file.

```
#include "boot.h"
#include "xmc_uart.h"
Include dependency graph for ARMCM4_XMC4/rs232.c:
```



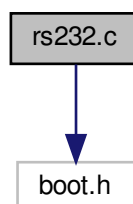
### 7.1142.1 Detailed Description

Bootloader RS232 communication interface source file.

## 7.1143 rs232.c File Reference

Bootloader RS232 communication interface source file.

```
#include "boot.h"
Include dependency graph for ARMCM7_STM32F7/rs232.c:
```



### 7.1143.1 Detailed Description

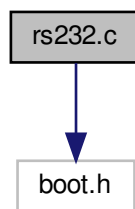
Bootloader RS232 communication interface source file.

## 7.1144 rs232.c File Reference

Bootloader RS232 communication interface source file.

```
#include "boot.h"
```

Include dependency graph for ARMCM7\_STM32H7/rs232.c:



### 7.1144.1 Detailed Description

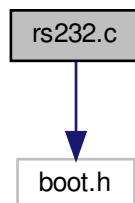
Bootloader RS232 communication interface source file.

## 7.1145 rs232.c File Reference

Bootloader RS232 communication interface source file.

```
#include "boot.h"
```

Include dependency graph for HCS12/rs232.c:



### 7.1145.1 Detailed Description

Bootloader RS232 communication interface source file.

## 7.1146 rs232.h File Reference

Bootloader RS232 communication interface header file.

### 7.1146.1 Detailed Description

Bootloader RS232 communication interface header file.

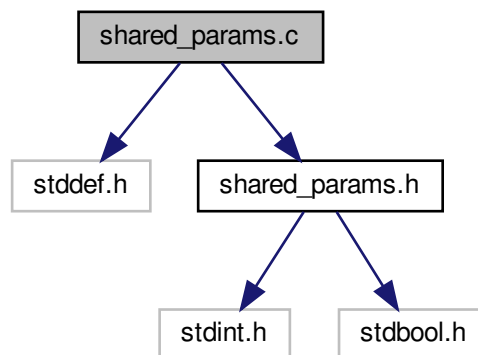
## 7.1147 shared\_params.c File Reference

Shared RAM parameters source file.

```
#include <stddef.h>
```

```
#include "shared_params.h"
```

Include dependency graph for ARMCM3\_LM3S\_EK\_LM3S6965\_GCC/Boot/shared\_params.c:



## Data Structures

- struct [tSharedParamsBuffer](#)

*Layout of the shared parameters RAM buffer.*

## Macros

- #define [SHARED\\_PARAMS\\_BUFFER\\_ID](#) (0xCE42E7A2u)

*Constant parameter buffer identifier. This value is always located as the start of the buffer to validate the the RAM contains valid shared parameters.*

## Functions

- static `tSharedParamsBuffer` `sharedParamsBuffer` `__attribute__` ((section(".shared"))))  
*Declaration of the actual parameter buffer that this module manages.*
- static bool `SharedParamsValidateBuffer` (void)  
*Validates the shared parameter buffer contents by looking at the table identifier and verifying its checksum.*
- static void `SharedParamsWriteChecksum` (void)  
*Calculates and writes the checksum into the buffer.*
- static bool `SharedParamsVerifyChecksum` (void)  
*Calculates and verifies the checksum that is currently present in the buffer.*
- static uint16\_t `SharedParamsCalculateChecksum` (void)  
*Calculates and returns the checksum value for the current contents in the buffer. The checksum is calculated by taking the sum of all bytes in the parameter buffer (excluding the checksum at the end) and then taking the two's complement value of it.*
- void `SharedParamsInit` (void)  
*Initializes the shared RAM parameters module.*
- bool `SharedParamsReadByIndex` (uint32\_t idx, uint8\_t \*value)  
*Reads a data byte from the shared parameter buffer at the specified index.*
- bool `SharedParamsWriteByIndex` (uint32\_t idx, uint8\_t value)  
*Writes a data byte to the shared parameter buffer at the specified index.*

### 7.1147.1 Detailed Description

Shared RAM parameters source file.

### 7.1147.2 Function Documentation

#### 7.1147.2.1 `__attribute__`()

```
static tSharedParamsBuffer sharedParamsBuffer __attribute__ (
 (section(".shared"))) [static]
```

Declaration of the actual parameter buffer that this module manages.

#### Warning

For the shared RAM parameters to work properly for sharing information between the bootloader and user program, it is important that this variable is linked to the exact same RAM address in both the bootloader and the user program. Additionally, it should be configured such that the C-startup code does NOT zero its contents during system initialization. This is the code that runs in the reset event handler, before function `main()` is called. For GCC based embedded toolchains, the solution is to assign this variable to a custom section, in this case called ".shared". Then in the linker script, add the following to the SECTIONS:

```
.shared (NOLOAD) : { . = ALIGN(4); _sshared = .; shared_start = _sshared; *(.shared) *(.shared.*) KE←
EP(*(.shared)) . = ALIGN(4); _eshared = .; shared_end = _eshared; } >SHARED
```

Next, add a new MEMORY entry for SHARED at the start of RAM and reduce the length of the remaining RAM:

```
SHARED (xrw) : ORIGIN = 0x200000C0, LENGTH = 64 RAM (xrw) : ORIGIN = 0x20000100, LENGTH = 32K - 192
- 64
```

Note that the previous example is for an STM32F0 microcontroller where the first 192 (0xC0) bytes in RAM are reserved for the user program vector table.



## Remarks

This same approach can be applied with other toolchains such as Keil MDK and IAR EWARM. Consult the compiler and linker user manuals of your toolchain to find out how to place a RAM variable at a fixed memory address and to prevent the C-startup code from zeroing its contents. Here are a few links to get you started:

- IAR EWARM: <https://www.iar.com/support/tech-notes/compiler/linker-error-for-absolute-located-variable/>
- Keil MDK: <http://www.keil.com/support/docs/3480.htm>

### 7.1147.2.2 SharedParamsCalculateChecksum()

```
static uint16_t SharedParamsCalculateChecksum (
 void) [static]
```

Calculates and returns the checksum value for the current contents in the buffer. The checksum is calculated by taking the sum of all bytes in the parameter buffer (excluding the checksum at the end) and then taking the two's complement value of it.

## Returns

The calculated checksum value.

Referenced by SharedParamsVerifyChecksum(), and SharedParamsWriteChecksum().

### 7.1147.2.3 SharedParamsInit()

```
void SharedParamsInit (
 void)
```

Initializes the shared RAM parameters module.

## Returns

none.

Referenced by AppInit(), and main().

### 7.1147.2.4 SharedParamsReadByIndex()

```
bool SharedParamsReadByIndex (
 uint32_t idx,
 uint8_t * value)
```

Reads a data byte from the shared parameter buffer at the specified index.

**Parameters**

|              |                                                                                                              |
|--------------|--------------------------------------------------------------------------------------------------------------|
| <i>idx</i>   | Index into the parameter data array. A valid value is between 0 and (SHARED_PARAMS_CFG_BUFFER_DATA_LEN - 1). |
| <i>value</i> | Pointer to where the read data value is stored.                                                              |

**Returns**

True if successful, false otherwise.

Referenced by ApplInit(), and main().

**7.1147.2.5 SharedParamsValidateBuffer()**

```
static bool SharedParamsValidateBuffer (
 void) [static]
```

Validates the shared parameter buffer contents by looking at the table identifier and verifying its checksum.

**Returns**

True if successful, false otherwise.

Referenced by SharedParamsInit(), SharedParamsReadByIndex(), and SharedParamsWriteByIndex().

**7.1147.2.6 SharedParamsVerifyChecksum()**

```
static bool SharedParamsVerifyChecksum (
 void) [static]
```

Calculates and verifies the checksum that is currently present in the buffer.

**Returns**

True is the checksum is correct, false otherwise.

Referenced by SharedParamsValidateBuffer().

**7.1147.2.7 SharedParamsWriteByIndex()**

```
bool SharedParamsWriteByIndex (
 uint32_t idx,
 uint8_t value)
```

Writes a data byte to the shared parameter buffer at the specified index.

### Parameters

|              |                                                                                                              |
|--------------|--------------------------------------------------------------------------------------------------------------|
| <i>idx</i>   | Index into the parameter data array. A valid value is between 0 and (SHARED_PARAMS_CFG_BUFFER_DATA_LEN - 1). |
| <i>value</i> | Value to write.                                                                                              |

### Returns

True if successful, false otherwise.

Referenced by ApplInit(), main(), and NetApp().

#### 7.1147.2.8 SharedParamsWriteChecksum()

```
static void SharedParamsWriteChecksum (
 void) [static]
```

Calculates and writes the checksum into the buffer.

### Returns

none.

Referenced by SharedParamsInit(), and SharedParamsWriteByIndex().

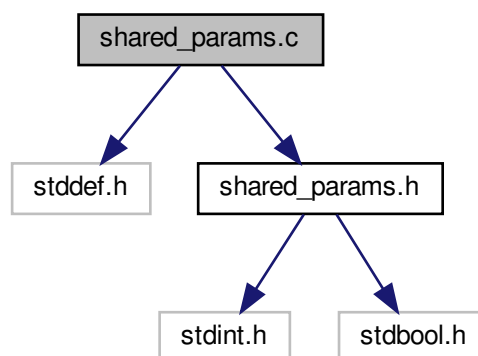
## 7.1148 shared\_params.c File Reference

Shared RAM parameters source file.

```
#include <stddef.h>
```

```
#include "shared_params.h"
```

Include dependency graph for ARMC3M3\_LM3S\_EK\_LM3S6965\_GCC/Prog/shared\_params.c:



## Data Structures

- struct [tSharedParamsBuffer](#)

*Layout of the shared parameters RAM buffer.*

## Macros

- #define [SHARED\\_PARAMS\\_BUFFER\\_ID](#) (0xCE42E7A2u)

*Constant parameter buffer identifier. This value is always located as the start of the buffer to validate the the RAM contains valid shared parameters.*

## Functions

- static [tSharedParamsBuffer](#) sharedParamsBuffer [\\_\\_attribute\\_\\_](#) ((section(".shared")))

*Declaration of the actual parameter buffer that this module manages.*

- static bool [SharedParamsValidateBuffer](#) (void)

*Validates the shared parameter buffer contents by looking at the table identifier and verifying its checksum.*

- static void [SharedParamsWriteChecksum](#) (void)

*Calculates and writes the checksum into the buffer.*

- static bool [SharedParamsVerifyChecksum](#) (void)

*Calculates and verifies the checksum that is currently present in the buffer.*

- static uint16\_t [SharedParamsCalculateChecksum](#) (void)

*Calculates and returns the checksum value for the current contents in the buffer. The checksum is calculated by taking the sum of all bytes in the parameter buffer (excluding the checksum at the end) and then taking the two's complement value of it.*

- void [SharedParamsInit](#) (void)

*Initializes the shared RAM parameters module.*

- bool [SharedParamsReadByIndex](#) (uint32\_t idx, uint8\_t \*value)

*Reads a data byte from the shared parameter buffer at the specified index.*

- bool [SharedParamsWriteByIndex](#) (uint32\_t idx, uint8\_t value)

*Writes a data byte to the shared parameter buffer at the specified index.*

### 7.1148.1 Detailed Description

Shared RAM parameters source file.

### 7.1148.2 Function Documentation

7.1148.2.1 `__attribute__()`

```
static tSharedParamsBuffer sharedParamsBuffer __attribute__ (
 (section(".shared"))) [static]
```

Declaration of the actual parameter buffer that this module manages.

## Warning

For the shared RAM parameters to work properly for sharing information between the bootloader and user program, it is important that this variable is linked to the exact same RAM address in both the bootloader and the user program. Additionally, it should be configured such that the C-startup code does NOT zero its contents during system initialization. This is the code that runs in the reset event handler, before function `main()` is called. For GCC based embedded toolchains, the solution is to assign this variable to a custom section, in this case called ".shared". Then in the linker script, add the following to the SECTIONS:

```
.shared (NOLOAD) : { . = ALIGN(4); _sshared = .; shared_start = _sshared; *(.shared) *(.shared.*) KE←
EP(*(.shared)) . = ALIGN(4); _eshared = .; shared_end = _eshared; } >SHARED
```

Next, add a new MEMORY entry for SHARED at the start of RAM and reduce the length of the remaining RAM:

```
SHARED (xrw) : ORIGIN = 0x200000C0, LENGTH = 64 RAM (xrw) : ORIGIN = 0x20000100, LENGTH = 32K - 192
- 64
```

Note that the previous example is for an STM32F0 microcontroller where the first 192 (0xC0) bytes in RAM are reserved for the user program vector table.

## Remarks

This same approach can be applied with other toolchains such as Keil MDK and IAR EWARM. Consult the compiler and linker user manuals of your toolchain to find out how to place a RAM variable at a fixed memory address and to prevent the C-startup code from zeroing its contents. Here are a few links to get you started:

- IAR EWARM: <https://www.iar.com/support/tech-notes/compiler/linker-error-for-absolute-located-variable/>
- Keil MDK: <http://www.keil.com/support/docs/3480.htm>

7.1148.2.2 `SharedParamsCalculateChecksum()`

```
static uint16_t SharedParamsCalculateChecksum (
 void) [static]
```

Calculates and returns the checksum value for the current contents in the buffer. The checksum is calculated by taking the sum of all bytes in the parameter buffer (excluding the checksum at the end) and then taking the two's complement value of it.

## Returns

The calculated checksum value.

Referenced by `SharedParamsVerifyChecksum()`, and `SharedParamsWriteChecksum()`.

### 7.1148.2.3 SharedParamsInit()

```
void SharedParamsInit (
 void)
```

Initializes the shared RAM parameters module.

#### Returns

none.

### 7.1148.2.4 SharedParamsReadByIndex()

```
bool SharedParamsReadByIndex (
 uint32_t idx,
 uint8_t * value)
```

Reads a data byte from the shared parameter buffer at the specified index.

#### Parameters

|              |                                                                                                              |
|--------------|--------------------------------------------------------------------------------------------------------------|
| <i>idx</i>   | Index into the parameter data array. A valid value is between 0 and (SHARED_PARAMS_CFG_BUFFER_DATA_LEN - 1). |
| <i>value</i> | Pointer to where the read data value is stored.                                                              |

#### Returns

True if successful, false otherwise.

### 7.1148.2.5 SharedParamsValidateBuffer()

```
static bool SharedParamsValidateBuffer (
 void) [static]
```

Validates the shared parameter buffer contents by looking at the table identifier and verifying its checksum.

#### Returns

True if successful, false otherwise.

Referenced by SharedParamsInit(), SharedParamsReadByIndex(), and SharedParamsWriteByIndex().

### 7.1148.2.6 SharedParamsVerifyChecksum()

```
static bool SharedParamsVerifyChecksum (
 void) [static]
```

Calculates and verifies the checksum that is currently present in the buffer.

#### Returns

True is the checksum is correct, false otherwise.

Referenced by SharedParamsValidateBuffer().

### 7.1148.2.7 SharedParamsWriteByIndex()

```
bool SharedParamsWriteByIndex (
 uint32_t idx,
 uint8_t value)
```

Writes a data byte to the shared parameter buffer at the specified index.

#### Parameters

|              |                                                                                                              |
|--------------|--------------------------------------------------------------------------------------------------------------|
| <i>idx</i>   | Index into the parameter data array. A valid value is between 0 and (SHARED_PARAMS_CFG_BUFFER_DATA_LEN - 1). |
| <i>value</i> | Value to write.                                                                                              |

#### Returns

True if successful, false otherwise.

### 7.1148.2.8 SharedParamsWriteChecksum()

```
static void SharedParamsWriteChecksum (
 void) [static]
```

Calculates and writes the checksum into the buffer.

#### Returns

none.

Referenced by SharedParamsInit(), and SharedParamsWriteByIndex().

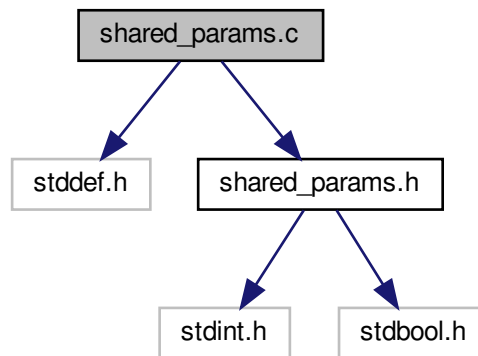
## 7.1149 shared\_params.c File Reference

Shared RAM parameters source file.

```
#include <stddef.h>
```

```
#include "shared_params.h"
```

Include dependency graph for ARMC3\_LM3S\_EK\_LM3S6965\_IAR/Boot/shared\_params.c:



### Data Structures

- struct [tSharedParamsBuffer](#)  
*Layout of the shared parameters RAM buffer.*

### Macros

- #define [SHARED\\_PARAMS\\_BUFFER\\_ID](#) (0xCE42E7A2u)  
*Constant parameter buffer identifier. This value is always located as the start of the buffer to validate the the RAM contains valid shared parameters.*

### Functions

- static bool [SharedParamsValidateBuffer](#) (void)  
*Validates the shared parameter buffer contents by looking at the table identifier and verifying its checksum.*
- static void [SharedParamsWriteChecksum](#) (void)  
*Calculates and writes the checksum into the buffer.*
- static bool [SharedParamsVerifyChecksum](#) (void)  
*Calculates and verifies the checksum that is currently present in the buffer.*
- static uint16\_t [SharedParamsCalculateChecksum](#) (void)  
*Calculates and returns the checksum value for the current contents in the buffer. The checksum is calculated by taking the sum of all bytes in the parameter buffer (excluding the checksum at the end) and then taking the two's complement value of it.*
- void [SharedParamsInit](#) (void)  
*Initializes the shared RAM parameters module.*
- bool [SharedParamsReadByIndex](#) (uint32\_t idx, uint8\_t \*value)  
*Reads a data byte from the shared parameter buffer at the specified index.*
- bool [SharedParamsWriteByIndex](#) (uint32\_t idx, uint8\_t value)  
*Writes a data byte to the shared parameter buffer at the specified index.*



## Variables

- static \_\_no\_init [tSharedParamsBuffer](#) sharedParamsBuffer [shared](#)  
*Declaration of the actual parameter buffer that this module manages.*

### 7.1149.1 Detailed Description

Shared RAM parameters source file.

### 7.1149.2 Function Documentation

#### 7.1149.2.1 SharedParamsCalculateChecksum()

```
static uint16_t SharedParamsCalculateChecksum (
 void) [static]
```

Calculates and returns the checksum value for the current contents in the buffer. The checksum is calculated by taking the sum of all bytes in the parameter buffer (excluding the checksum at the end) and then taking the two's complement value of it.

#### Returns

The calculated checksum value.

Referenced by SharedParamsVerifyChecksum(), and SharedParamsWriteChecksum().

#### 7.1149.2.2 SharedParamsInit()

```
void SharedParamsInit (
 void)
```

Initializes the shared RAM parameters module.

#### Returns

none.

#### 7.1149.2.3 SharedParamsReadByIndex()

```
bool SharedParamsReadByIndex (
 uint32_t idx,
 uint8_t * value)
```

Reads a data byte from the shared parameter buffer at the specified index.

**Parameters**

|              |                                                                                                              |
|--------------|--------------------------------------------------------------------------------------------------------------|
| <i>idx</i>   | Index into the parameter data array. A valid value is between 0 and (SHARED_PARAMS_CFG_BUFFER_DATA_LEN - 1). |
| <i>value</i> | Pointer to where the read data value is stored.                                                              |

**Returns**

True if successful, false otherwise.

**7.1149.2.4 SharedParamsValidateBuffer()**

```
static bool SharedParamsValidateBuffer (
 void) [static]
```

Validates the shared parameter buffer contents by looking at the table identifier and verifying its checksum.

**Returns**

True if successful, false otherwise.

Referenced by SharedParamsInit(), SharedParamsReadByIndex(), and SharedParamsWriteByIndex().

**7.1149.2.5 SharedParamsVerifyChecksum()**

```
static bool SharedParamsVerifyChecksum (
 void) [static]
```

Calculates and verifies the checksum that is currently present in the buffer.

**Returns**

True is the checksum is correct, false otherwise.

Referenced by SharedParamsValidateBuffer().

**7.1149.2.6 SharedParamsWriteByIndex()**

```
bool SharedParamsWriteByIndex (
 uint32_t idx,
 uint8_t value)
```

Writes a data byte to the shared parameter buffer at the specified index.

## Parameters

|              |                                                                                                              |
|--------------|--------------------------------------------------------------------------------------------------------------|
| <i>idx</i>   | Index into the parameter data array. A valid value is between 0 and (SHARED_PARAMS_CFG_BUFFER_DATA_LEN - 1). |
| <i>value</i> | Value to write.                                                                                              |

## Returns

True if successful, false otherwise.

## 7.1149.2.7 SharedParamsWriteChecksum()

```
static void SharedParamsWriteChecksum (
 void) [static]
```

Calculates and writes the checksum into the buffer.

## Returns

none.

Referenced by SharedParamsInit(), and SharedParamsWriteByIndex().

## 7.1149.3 Variable Documentation

## 7.1149.3.1 shared

```
__no_init tSharedParamsBuffer sharedParamsBuffer shared [static]
```

Declaration of the actual parameter buffer that this module manages.

## Warning

For the shared RAM parameters to work properly for sharing information between the bootloader and user program, it is important that this variable is linked to the exact same RAM address in both the bootloader and the user program. Additionally, it should be configured such that the C-startup code does NOT zero its contents during system initialization. This is the code that runs in the reset event handler, before function `main()` is called. For GCC based embedded toolchains, the solution is to assign this variable to a custom section, in this case called ".shared". Then in the linker script, add the following to the SECTIONS:

```
.shared (NOLOAD) : { . = ALIGN(4); _sshared = .; shared_start = _sshared; *(.shared) *(.shared.*) KE←
EP(*(.shared)) . = ALIGN(4); _eshared = .; shared_end = _eshared; } >SHARED
```

Next, add a new MEMORY entry for SHARED at the start of RAM and reduce the length of the remaining RAM:

```
SHARED (xrw) : ORIGIN = 0x200000C0, LENGTH = 64 RAM (xrw) : ORIGIN = 0x20000100, LENGTH = 32K - 192
- 64
```

Note that the previous example is for an STM32F0 microcontroller where the first 192 (0xC0) bytes in RAM are reserved for the user program vector table.

## Remarks

This same approach can be applied with other toolchains such as Keil MDK and IAR EWARM. Consult the compiler and linker user manuals of your toolchain to find out how to place a RAM variable at a fixed memory address and to prevent the C-startup code from zeroing its contents. Here are a few links to get you started:

- IAR EWARM: <https://www.iar.com/support/tech-notes/compiler/linker-error-for-absolute-located-variable/>
- Keil MDK: <http://www.keil.com/support/docs/3480.htm>

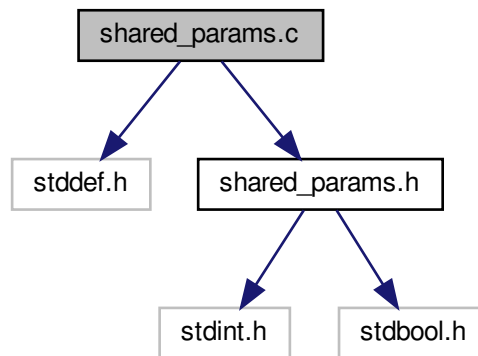
## 7.1150 shared\_params.c File Reference

Shared RAM parameters source file.

```
#include <stddef.h>
```

```
#include "shared_params.h"
```

Include dependency graph for ARMCM3\_LM3S\_EK\_LM3S6965\_IAR/Prog/shared\_params.c:



### Data Structures

- struct [tSharedParamsBuffer](#)  
*Layout of the shared parameters RAM buffer.*

### Macros

- `#define` [SHARED\\_PARAMS\\_BUFFER\\_ID](#) (0xCE42E7A2u)  
*Constant parameter buffer identifier. This value is always located as the start of the buffer to validate the the RAM contains valid shared parameters.*

### Functions

- static bool [SharedParamsValidateBuffer](#) (void)  
*Validates the shared parameter buffer contents by looking at the table identifier and verifying its checksum.*
- static void [SharedParamsWriteChecksum](#) (void)  
*Calculates and writes the checksum into the buffer.*
- static bool [SharedParamsVerifyChecksum](#) (void)  
*Calculates and verifies the checksum that is currently present in the buffer.*
- static uint16\_t [SharedParamsCalculateChecksum](#) (void)  
*Calculates and returns the checksum value for the current contents in the buffer. The checksum is calculated by taking the sum of all bytes in the parameter buffer (excluding the checksum at the end) and then taking the two's complement value of it.*
- void [SharedParamsInit](#) (void)  
*Initializes the shared RAM parameters module.*
- bool [SharedParamsReadByIndex](#) (uint32\_t idx, uint8\_t \*value)  
*Reads a data byte from the shared parameter buffer at the specified index.*
- bool [SharedParamsWriteByIndex](#) (uint32\_t idx, uint8\_t value)  
*Writes a data byte to the shared parameter buffer at the specified index.*

## Variables

- static \_\_no\_init [tSharedParamsBuffer](#) sharedParamsBuffer [shared](#)  
*Declaration of the actual parameter buffer that this module manages.*

### 7.1150.1 Detailed Description

Shared RAM parameters source file.

### 7.1150.2 Function Documentation

#### 7.1150.2.1 SharedParamsCalculateChecksum()

```
static uint16_t SharedParamsCalculateChecksum (
 void) [static]
```

Calculates and returns the checksum value for the current contents in the buffer. The checksum is calculated by taking the sum of all bytes in the parameter buffer (excluding the checksum at the end) and then taking the two's complement value of it.

#### Returns

The calculated checksum value.

Referenced by SharedParamsVerifyChecksum(), and SharedParamsWriteChecksum().

#### 7.1150.2.2 SharedParamsInit()

```
void SharedParamsInit (
 void)
```

Initializes the shared RAM parameters module.

#### Returns

none.

#### 7.1150.2.3 SharedParamsReadByIndex()

```
bool SharedParamsReadByIndex (
 uint32_t idx,
 uint8_t * value)
```

Reads a data byte from the shared parameter buffer at the specified index.

**Parameters**

|              |                                                                                                              |
|--------------|--------------------------------------------------------------------------------------------------------------|
| <i>idx</i>   | Index into the parameter data array. A valid value is between 0 and (SHARED_PARAMS_CFG_BUFFER_DATA_LEN - 1). |
| <i>value</i> | Pointer to where the read data value is stored.                                                              |

**Returns**

True if successful, false otherwise.

**7.1150.2.4 SharedParamsValidateBuffer()**

```
static bool SharedParamsValidateBuffer (
 void) [static]
```

Validates the shared parameter buffer contents by looking at the table identifier and verifying its checksum.

**Returns**

True if successful, false otherwise.

Referenced by SharedParamsInit(), SharedParamsReadByIndex(), and SharedParamsWriteByIndex().

**7.1150.2.5 SharedParamsVerifyChecksum()**

```
static bool SharedParamsVerifyChecksum (
 void) [static]
```

Calculates and verifies the checksum that is currently present in the buffer.

**Returns**

True is the checksum is correct, false otherwise.

Referenced by SharedParamsValidateBuffer().

**7.1150.2.6 SharedParamsWriteByIndex()**

```
bool SharedParamsWriteByIndex (
 uint32_t idx,
 uint8_t value)
```

Writes a data byte to the shared parameter buffer at the specified index.

## Parameters

|              |                                                                                                              |
|--------------|--------------------------------------------------------------------------------------------------------------|
| <i>idx</i>   | Index into the parameter data array. A valid value is between 0 and (SHARED_PARAMS_CFG_BUFFER_DATA_LEN - 1). |
| <i>value</i> | Value to write.                                                                                              |

## Returns

True if successful, false otherwise.

## 7.1150.2.7 SharedParamsWriteChecksum()

```
static void SharedParamsWriteChecksum (
 void) [static]
```

Calculates and writes the checksum into the buffer.

## Returns

none.

Referenced by SharedParamsInit(), and SharedParamsWriteByIndex().

## 7.1150.3 Variable Documentation

## 7.1150.3.1 shared

```
__no_init tSharedParamsBuffer sharedParamsBuffer shared [static]
```

Declaration of the actual parameter buffer that this module manages.

## Warning

For the shared RAM parameters to work properly for sharing information between the bootloader and user program, it is important that this variable is linked to the exact same RAM address in both the bootloader and the user program. Additionally, it should be configured such that the C-startup code does NOT zero its contents during system initialization. This is the code that runs in the reset event handler, before function `main()` is called. For GCC based embedded toolchains, the solution is to assign this variable to a custom section, in this case called ".shared". Then in the linker script, add the following to the SECTIONS:

```
.shared (NOLOAD) : { . = ALIGN(4); _sshared = .; shared_start = _sshared; *(.shared) *(.shared.*) KE←
EP(*(.shared)) . = ALIGN(4); _eshared = .; shared_end = _eshared; } >SHARED
```

Next, add a new MEMORY entry for SHARED at the start of RAM and reduce the length of the remaining RAM:

```
SHARED (xrw) : ORIGIN = 0x200000C0, LENGTH = 64 RAM (xrw) : ORIGIN = 0x20000100, LENGTH = 32K - 192
- 64
```

Note that the previous example is for an STM32F0 microcontroller where the first 192 (0xC0) bytes in RAM are reserved for the user program vector table.

## Remarks

This same approach can be applied with other toolchains such as Keil MDK and IAR EWARM. Consult the compiler and linker user manuals of your toolchain to find out how to place a RAM variable at a fixed memory address and to prevent the C-startup code from zeroing its contents. Here are a few links to get you started:

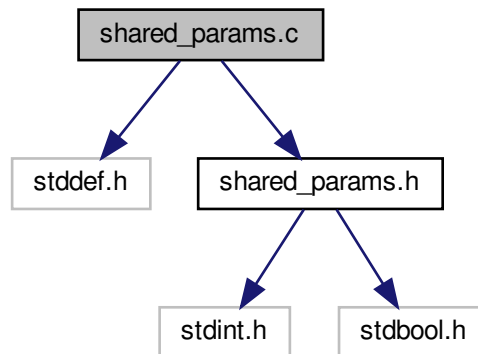
- IAR EWARM: <https://www.iar.com/support/tech-notes/compiler/linker-error-for-absolute-located-variable/>
- Keil MDK: <http://www.keil.com/support/docs/3480.htm>

## 7.1151 shared\_params.c File Reference

Shared RAM parameters source file.

```
#include <stddef.h>
#include "shared_params.h"
```

Include dependency graph for ARMCM4\_STM32F4\_Nucleo\_F429ZI\_CubeIDE/Boot/App/shared\_params.c:



### Data Structures

- struct [tSharedParamsBuffer](#)

*Layout of the shared parameters RAM buffer.*

### Macros

- `#define` [SHARED\\_PARAMS\\_BUFFER\\_ID](#) (0xCE42E7A2u)

*Constant parameter buffer identifier. This value is always located as the start of the buffer to validate the the RAM contains valid shared parameters.*

### Functions

- static [tSharedParamsBuffer](#) `sharedParamsBuffer` `__attribute__((section(".shared")))`  
*Declaration of the actual parameter buffer that this module manages.*
- static bool [SharedParamsValidateBuffer](#) (void)  
*Validates the shared parameter buffer contents by looking at the table identifier and verifying its checksum.*
- static void [SharedParamsWriteChecksum](#) (void)  
*Calculates and writes the checksum into the buffer.*
- static bool [SharedParamsVerifyChecksum](#) (void)  
*Calculates and verifies the checksum that is currently present in the buffer.*
- static uint16\_t [SharedParamsCalculateChecksum](#) (void)



*Calculates and returns the checksum value for the current contents in the buffer. The checksum is calculated by taking the sum of all bytes in the parameter buffer (excluding the checksum at the end) and then taking the two's complement value of it.*

- void [SharedParamsInit](#) (void)  
*Initializes the shared RAM parameters module.*
- bool [SharedParamsReadByIndex](#) (uint32\_t idx, uint8\_t \*value)  
*Reads a data byte from the shared parameter buffer at the specified index.*
- bool [SharedParamsWriteByIndex](#) (uint32\_t idx, uint8\_t value)  
*Writes a data byte to the shared parameter buffer at the specified index.*

### 7.1151.1 Detailed Description

Shared RAM parameters source file.

### 7.1151.2 Function Documentation

#### 7.1151.2.1 \_\_attribute\_\_()

```
static tSharedParamsBuffer sharedParamsBuffer __attribute__ (
 (section(".shared"))) [static]
```

Declaration of the actual parameter buffer that this module manages.

#### Warning

For the shared RAM parameters to work properly for sharing information between the bootloader and user program, it is important that this variable is linked to the exact same RAM address in both the bootloader and the user program. Additionally, it should be configured such that the C-startup code does NOT zero its contents during system initialization. This is the code that runs in the reset event handler, before function [main\(\)](#) is called. For GCC based embedded toolchains, the solution is to assign this variable to a custom section, in this case called ".shared". Then in the linker script, add the following to the SECTIONS:

```
.shared (NOLOAD) : { . = ALIGN(4); _sshared = .; shared_start = _sshared; *(.shared) *(.shared.*) KE↔
EP*(.shared)) . = ALIGN(4); _eshared = .; shared_end = _eshared; } >SHARED
```

Next, add a new MEMORY entry for SHARED at the start of RAM and reduce the length of the remaining RAM:

```
SHARED (xrw) : ORIGIN = 0x200000C0, LENGTH = 64 RAM (xrw) : ORIGIN = 0x20000100, LENGTH = 32K - 192
- 64
```

Note that the previous example is for an STM32F0 microcontroller where the first 192 (0xC0) bytes in RAM are reserved for the user program vector table.

#### Remarks

This same approach can be applied with other toolchains such as Keil MDK and IAR EWARM. Consult the compiler and linker user manuals of your toolchain to find out how to place a RAM variable at a fixed memory address and to prevent the C-startup code from zeroing its contents. Here are a few links to get you started:

- IAR EWARM: <https://www.iar.com/support/tech-notes/compiler/linker-error-for-absolute-located-variable/>
- Keil MDK: <http://www.keil.com/support/docs/3480.htm>

#### 7.1151.2.2 SharedParamsCalculateChecksum()

```
static uint16_t SharedParamsCalculateChecksum (
 void) [static]
```

Calculates and returns the checksum value for the current contents in the buffer. The checksum is calculated by taking the sum of all bytes in the parameter buffer (excluding the checksum at the end) and then taking the two's complement value of it.

##### Returns

The calculated checksum value.

Referenced by SharedParamsVerifyChecksum(), and SharedParamsWriteChecksum().

#### 7.1151.2.3 SharedParamsInit()

```
void SharedParamsInit (
 void)
```

Initializes the shared RAM parameters module.

##### Returns

none.

#### 7.1151.2.4 SharedParamsReadByIndex()

```
bool SharedParamsReadByIndex (
 uint32_t idx,
 uint8_t * value)
```

Reads a data byte from the shared parameter buffer at the specified index.

##### Parameters

|              |                                                                                                              |
|--------------|--------------------------------------------------------------------------------------------------------------|
| <i>idx</i>   | Index into the parameter data array. A valid value is between 0 and (SHARED_PARAMS_CFG_BUFFER_DATA_LEN - 1). |
| <i>value</i> | Pointer to where the read data value is stored.                                                              |

##### Returns

True if successful, false otherwise.

### 7.1151.2.5 SharedParamsValidateBuffer()

```
static bool SharedParamsValidateBuffer (
 void) [static]
```

Validates the shared parameter buffer contents by looking at the table identifier and verifying its checksum.

#### Returns

True if successful, false otherwise.

Referenced by SharedParamsInit(), SharedParamsReadByIndex(), and SharedParamsWriteByIndex().

### 7.1151.2.6 SharedParamsVerifyChecksum()

```
static bool SharedParamsVerifyChecksum (
 void) [static]
```

Calculates and verifies the checksum that is currently present in the buffer.

#### Returns

True is the checksum is correct, false otherwise.

Referenced by SharedParamsValidateBuffer().

### 7.1151.2.7 SharedParamsWriteByIndex()

```
bool SharedParamsWriteByIndex (
 uint32_t idx,
 uint8_t value)
```

Writes a data byte to the shared parameter buffer at the specified index.

#### Parameters

|              |                                                                                                              |
|--------------|--------------------------------------------------------------------------------------------------------------|
| <i>idx</i>   | Index into the parameter data array. A valid value is between 0 and (SHARED_PARAMS_CFG_BUFFER_DATA_LEN - 1). |
| <i>value</i> | Value to write.                                                                                              |

#### Returns

True if successful, false otherwise.

### 7.1151.2.8 SharedParamsWriteChecksum()

```
static void SharedParamsWriteChecksum (
 void) [static]
```

Calculates and writes the checksum into the buffer.

#### Returns

none.

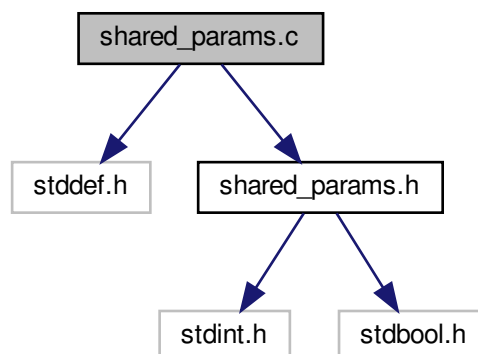
Referenced by SharedParamsInit(), and SharedParamsWriteByIndex().

## 7.1152 shared\_params.c File Reference

Shared RAM parameters source file.

```
#include <stddef.h>
#include "shared_params.h"
```

Include dependency graph for ARMCM4\_STM32F4\_Nucleo\_F429ZI\_CubeIDE/Prog/App/shared\_params.c:



## Data Structures

- struct [tSharedParamsBuffer](#)

*Layout of the shared parameters RAM buffer.*

## Macros

- #define [SHARED\\_PARAMS\\_BUFFER\\_ID](#) (0xCE42E7A2u)

*Constant parameter buffer identifier. This value is always located as the start of the buffer to validate the the RAM contains valid shared parameters.*

## Functions

- static `tSharedParamsBuffer` sharedParamsBuffer `__attribute__ ((section(".shared")))`  
*Declaration of the actual parameter buffer that this module manages.*
- static bool `SharedParamsValidateBuffer` (void)  
*Validates the shared parameter buffer contents by looking at the table identifier and verifying its checksum.*
- static void `SharedParamsWriteChecksum` (void)  
*Calculates and writes the checksum into the buffer.*
- static bool `SharedParamsVerifyChecksum` (void)  
*Calculates and verifies the checksum that is currently present in the buffer.*
- static uint16\_t `SharedParamsCalculateChecksum` (void)  
*Calculates and returns the checksum value for the current contents in the buffer. The checksum is calculated by taking the sum of all bytes in the parameter buffer (excluding the checksum at the end) and then taking the two's complement value of it.*
- void `SharedParamsInit` (void)  
*Initializes the shared RAM parameters module.*
- bool `SharedParamsReadByIndex` (uint32\_t idx, uint8\_t \*value)  
*Reads a data byte from the shared parameter buffer at the specified index.*
- bool `SharedParamsWriteByIndex` (uint32\_t idx, uint8\_t value)  
*Writes a data byte to the shared parameter buffer at the specified index.*

### 7.1152.1 Detailed Description

Shared RAM parameters source file.

### 7.1152.2 Function Documentation

#### 7.1152.2.1 `__attribute__()`

```
static tSharedParamsBuffer sharedParamsBuffer __attribute__ (
 (section(".shared"))) [static]
```

Declaration of the actual parameter buffer that this module manages.

#### Warning

For the shared RAM parameters to work properly for sharing information between the bootloader and user program, it is important that this variable is linked to the exact same RAM address in both the bootloader and the user program. Additionally, it should be configured such that the C-startup code does NOT zero its contents during system initialization. This is the code that runs in the reset event handler, before function `main()` is called. For GCC based embedded toolchains, the solution is to assign this variable to a custom section, in this case called ".shared". Then in the linker script, add the following to the SECTIONS:

```
.shared (NOLOAD) : { . = ALIGN(4); _sshared = .; shared_start = _sshared; *(.shared) *(.shared.*) KE←
EP(*(.shared)) . = ALIGN(4); _eshared = .; shared_end = _eshared; } >SHARED
```

Next, add a new MEMORY entry for SHARED at the start of RAM and reduce the length of the remaining RAM:

```
SHARED (xrw) : ORIGIN = 0x200000C0, LENGTH = 64 RAM (xrw) : ORIGIN = 0x20000100, LENGTH = 32K - 192
- 64
```

Note that the previous example is for an STM32F0 microcontroller where the first 192 (0xC0) bytes in RAM are reserved for the user program vector table.

## Remarks

This same approach can be applied with other toolchains such as Keil MDK and IAR EWARM. Consult the compiler and linker user manuals of your toolchain to find out how to place a RAM variable at a fixed memory address and to prevent the C-startup code from zeroing its contents. Here are a few links to get you started:

- IAR EWARM: <https://www.iar.com/support/tech-notes/compiler/linker-error-for-absolute-located-variable/>
- Keil MDK: <http://www.keil.com/support/docs/3480.htm>

### 7.1152.2.2 SharedParamsCalculateChecksum()

```
static uint16_t SharedParamsCalculateChecksum (
 void) [static]
```

Calculates and returns the checksum value for the current contents in the buffer. The checksum is calculated by taking the sum of all bytes in the parameter buffer (excluding the checksum at the end) and then taking the two's complement value of it.

## Returns

The calculated checksum value.

Referenced by SharedParamsVerifyChecksum(), and SharedParamsWriteChecksum().

### 7.1152.2.3 SharedParamsInit()

```
void SharedParamsInit (
 void)
```

Initializes the shared RAM parameters module.

## Returns

none.

### 7.1152.2.4 SharedParamsReadByIndex()

```
bool SharedParamsReadByIndex (
 uint32_t idx,
 uint8_t * value)
```

Reads a data byte from the shared parameter buffer at the specified index.

**Parameters**

|              |                                                                                                              |
|--------------|--------------------------------------------------------------------------------------------------------------|
| <i>idx</i>   | Index into the parameter data array. A valid value is between 0 and (SHARED_PARAMS_CFG_BUFFER_DATA_LEN - 1). |
| <i>value</i> | Pointer to where the read data value is stored.                                                              |

**Returns**

True if successful, false otherwise.

**7.1152.2.5 SharedParamsValidateBuffer()**

```
static bool SharedParamsValidateBuffer (
 void) [static]
```

Validates the shared parameter buffer contents by looking at the table identifier and verifying its checksum.

**Returns**

True if successful, false otherwise.

Referenced by SharedParamsInit(), SharedParamsReadByIndex(), and SharedParamsWriteByIndex().

**7.1152.2.6 SharedParamsVerifyChecksum()**

```
static bool SharedParamsVerifyChecksum (
 void) [static]
```

Calculates and verifies the checksum that is currently present in the buffer.

**Returns**

True is the checksum is correct, false otherwise.

Referenced by SharedParamsValidateBuffer().

**7.1152.2.7 SharedParamsWriteByIndex()**

```
bool SharedParamsWriteByIndex (
 uint32_t idx,
 uint8_t value)
```

Writes a data byte to the shared parameter buffer at the specified index.

### Parameters

|              |                                                                                                              |
|--------------|--------------------------------------------------------------------------------------------------------------|
| <i>idx</i>   | Index into the parameter data array. A valid value is between 0 and (SHARED_PARAMS_CFG_BUFFER_DATA_LEN - 1). |
| <i>value</i> | Value to write.                                                                                              |

### Returns

True if successful, false otherwise.

#### 7.1152.2.8 SharedParamsWriteChecksum()

```
static void SharedParamsWriteChecksum (
 void) [static]
```

Calculates and writes the checksum into the buffer.

### Returns

none.

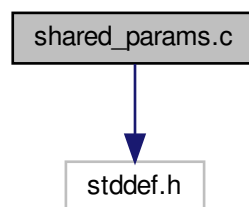
Referenced by SharedParamsInit(), and SharedParamsWriteByIndex().

## 7.1153 shared\_params.c File Reference

Shared RAM parameters source file.

```
#include <stddef.h>
#include "shared_params.h"
```

Include dependency graph for ARMCM4\_STM32F4\_Nucleo\_F429ZI\_GCC/Boot/shared\_params.c:



### Data Structures

- struct [tSharedParamsBuffer](#)

*Layout of the shared parameters RAM buffer.*



## Macros

- `#define SHARED_PARAMS_BUFFER_ID (0xCE42E7A2u)`  
*Constant parameter buffer identifier. This value is always located as the start of the buffer to validate the the RAM contains valid shared parameters.*

## Functions

- `static tSharedParamsBuffer sharedParamsBuffer __attribute__((section(".shared")))`  
*Declaration of the actual parameter buffer that this module manages.*
- `static bool SharedParamsValidateBuffer (void)`  
*Validates the shared parameter buffer contents by looking at the table identifier and verifying its checksum.*
- `static void SharedParamsWriteChecksum (void)`  
*Calculates and writes the checksum into the buffer.*
- `static bool SharedParamsVerifyChecksum (void)`  
*Calculates and verifies the checksum that is currently present in the buffer.*
- `static uint16_t SharedParamsCalculateChecksum (void)`  
*Calculates and returns the checksum value for the current contents in the buffer. The checksum is calculated by taking the sum of all bytes in the parameter buffer (excluding the checksum at the end) and then taking the two's complement value of it.*
- `void SharedParamsInit (void)`  
*Initializes the shared RAM parameters module.*
- `bool SharedParamsReadByIndex (uint32_t idx, uint8_t *value)`  
*Reads a data byte from the shared parameter buffer at the specified index.*
- `bool SharedParamsWriteByIndex (uint32_t idx, uint8_t value)`  
*Writes a data byte to the shared parameter buffer at the specified index.*

### 7.1153.1 Detailed Description

Shared RAM parameters source file.

### 7.1153.2 Function Documentation

#### 7.1153.2.1 \_\_attribute\_\_()

```
static tSharedParamsBuffer sharedParamsBuffer __attribute__((section(".shared"))) [static]
```

Declaration of the actual parameter buffer that this module manages.

### Warning

For the shared RAM parameters to work properly for sharing information between the bootloader and user program, it is important that this variable is linked to the exact same RAM address in both the bootloader and the user program. Additionally, it should be configured such that the C-startup code does NOT zero its contents during system initialization. This is the code that runs in the reset event handler, before function `main()` is called. For GCC based embedded toolchains, the solution is to assign this variable to a custom section, in this case called ".shared". Then in the linker script, add the following to the SECTIONS:

```
.shared (NOLOAD) : { . = ALIGN(4); _sshared = .; shared_start = _sshared; *(.shared) *(.shared.*) KE←
EP(*(.shared)) . = ALIGN(4); _eshared = .; shared_end = _eshared; } >SHARED
```

Next, add a new MEMORY entry for SHARED at the start of RAM and reduce the length of the remaining RAM:

```
SHARED (xrw) : ORIGIN = 0x200000C0, LENGTH = 64 RAM (xrw) : ORIGIN = 0x20000100, LENGTH = 32K - 192
- 64
```

Note that the previous example is for an STM32F0 microcontroller where the first 192 (0xC0) bytes in RAM are reserved for the user program vector table.

### Remarks

This same approach can be applied with other toolchains such as Keil MDK and IAR EWARM. Consult the compiler and linker user manuals of your toolchain to find out how to place a RAM variable at a fixed memory address and to prevent the C-startup code from zeroing its contents. Here are a few links to get you started:

- IAR EWARM: <https://www.iar.com/support/tech-notes/compiler/linker-error-for-absolute-located-variable/>
- Keil MDK: <http://www.keil.com/support/docs/3480.htm>

#### 7.1153.2.2 SharedParamsCalculateChecksum()

```
static uint16_t SharedParamsCalculateChecksum (
 void) [static]
```

Calculates and returns the checksum value for the current contents in the buffer. The checksum is calculated by taking the sum of all bytes in the parameter buffer (excluding the checksum at the end) and then taking the two's complement value of it.

### Returns

The calculated checksum value.

Referenced by SharedParamsVerifyChecksum(), and SharedParamsWriteChecksum().

### 7.1153.2.3 SharedParamsInit()

```
void SharedParamsInit (
 void)
```

Initializes the shared RAM parameters module.

#### Returns

none.

### 7.1153.2.4 SharedParamsReadByIndex()

```
bool SharedParamsReadByIndex (
 uint32_t idx,
 uint8_t * value)
```

Reads a data byte from the shared parameter buffer at the specified index.

#### Parameters

|              |                                                                                                              |
|--------------|--------------------------------------------------------------------------------------------------------------|
| <i>idx</i>   | Index into the parameter data array. A valid value is between 0 and (SHARED_PARAMS_CFG_BUFFER_DATA_LEN - 1). |
| <i>value</i> | Pointer to where the read data value is stored.                                                              |

#### Returns

True if successful, false otherwise.

### 7.1153.2.5 SharedParamsValidateBuffer()

```
static bool SharedParamsValidateBuffer (
 void) [static]
```

Validates the shared parameter buffer contents by looking at the table identifier and verifying its checksum.

#### Returns

True if successful, false otherwise.

Referenced by SharedParamsInit(), SharedParamsReadByIndex(), and SharedParamsWriteByIndex().

#### 7.1153.2.6 SharedParamsVerifyChecksum()

```
static bool SharedParamsVerifyChecksum (
 void) [static]
```

Calculates and verifies the checksum that is currently present in the buffer.

##### Returns

True is the checksum is correct, false otherwise.

Referenced by SharedParamsValidateBuffer().

#### 7.1153.2.7 SharedParamsWriteByIndex()

```
bool SharedParamsWriteByIndex (
 uint32_t idx,
 uint8_t value)
```

Writes a data byte to the shared parameter buffer at the specified index.

##### Parameters

|              |                                                                                                              |
|--------------|--------------------------------------------------------------------------------------------------------------|
| <i>idx</i>   | Index into the parameter data array. A valid value is between 0 and (SHARED_PARAMS_CFG_BUFFER_DATA_LEN - 1). |
| <i>value</i> | Value to write.                                                                                              |

##### Returns

True if successful, false otherwise.

#### 7.1153.2.8 SharedParamsWriteChecksum()

```
static void SharedParamsWriteChecksum (
 void) [static]
```

Calculates and writes the checksum into the buffer.

##### Returns

none.

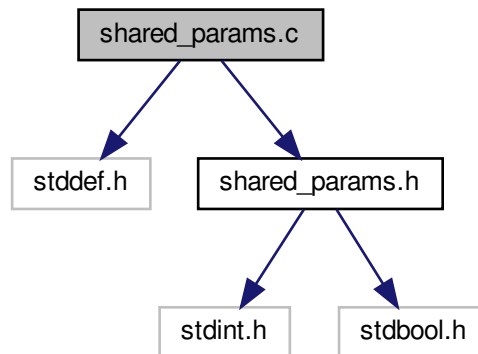
Referenced by SharedParamsInit(), and SharedParamsWriteByIndex().

## 7.1154 shared\_params.c File Reference

Shared RAM parameters source file.

```
#include <stddef.h>
#include "shared_params.h"
```

Include dependency graph for ARMCM4\_STM32F4\_Nucleo\_F429ZI\_GCC/Prog/shared\_params.c:



### Data Structures

- struct [tSharedParamsBuffer](#)

*Layout of the shared parameters RAM buffer.*

### Macros

- `#define` [SHARED\\_PARAMS\\_BUFFER\\_ID](#) (0xCE42E7A2u)

*Constant parameter buffer identifier. This value is always located as the start of the buffer to validate the the RAM contains valid shared parameters.*

### Functions

- static [tSharedParamsBuffer](#) `sharedParamsBuffer` `__attribute__((section(".shared")))`  
*Declaration of the actual parameter buffer that this module manages.*
- static bool [SharedParamsValidateBuffer](#) (void)  
*Validates the shared parameter buffer contents by looking at the table identifier and verifying its checksum.*
- static void [SharedParamsWriteChecksum](#) (void)  
*Calculates and writes the checksum into the buffer.*
- static bool [SharedParamsVerifyChecksum](#) (void)  
*Calculates and verifies the checksum that is currently present in the buffer.*
- static uint16\_t [SharedParamsCalculateChecksum](#) (void)

*Calculates and returns the checksum value for the current contents in the buffer. The checksum is calculated by taking the sum of all bytes in the parameter buffer (excluding the checksum at the end) and then taking the two's complement value of it.*

- void [SharedParamsInit](#) (void)  
*Initializes the shared RAM parameters module.*
- bool [SharedParamsReadByIndex](#) (uint32\_t idx, uint8\_t \*value)  
*Reads a data byte from the shared parameter buffer at the specified index.*
- bool [SharedParamsWriteByIndex](#) (uint32\_t idx, uint8\_t value)  
*Writes a data byte to the shared parameter buffer at the specified index.*

### 7.1154.1 Detailed Description

Shared RAM parameters source file.

### 7.1154.2 Function Documentation

#### 7.1154.2.1 `__attribute__()`

```
static tSharedParamsBuffer sharedParamsBuffer __attribute__ (
 (section(".shared"))) [static]
```

Declaration of the actual parameter buffer that this module manages.

#### Warning

For the shared RAM parameters to work properly for sharing information between the bootloader and user program, it is important that this variable is linked to the exact same RAM address in both the bootloader and the user program. Additionally, it should be configured such that the C-startup code does NOT zero its contents during system initialization. This is the code that runs in the reset event handler, before function [main\(\)](#) is called. For GCC based embedded toolchains, the solution is to assign this variable to a custom section, in this case called ".shared". Then in the linker script, add the following to the SECTIONS:

```
.shared (NOLOAD) : { . = ALIGN(4); _sshared = .; shared_start = _sshared; *(.shared) *(.shared.*) KE↔
EP*(.shared)) . = ALIGN(4); _eshared = .; shared_end = _eshared; } >SHARED
```

Next, add a new MEMORY entry for SHARED at the start of RAM and reduce the length of the remaining RAM:

```
SHARED (xrw) : ORIGIN = 0x200000C0, LENGTH = 64 RAM (xrw) : ORIGIN = 0x20000100, LENGTH = 32K - 192
- 64
```

Note that the previous example is for an STM32F0 microcontroller where the first 192 (0xC0) bytes in RAM are reserved for the user program vector table.

#### Remarks

This same approach can be applied with other toolchains such as Keil MDK and IAR EWARM. Consult the compiler and linker user manuals of your toolchain to find out how to place a RAM variable at a fixed memory address and to prevent the C-startup code from zeroing its contents. Here are a few links to get you started:

- IAR EWARM: <https://www.iar.com/support/tech-notes/compiler/linker-error-for-absolute-located-variable/>
- Keil MDK: <http://www.keil.com/support/docs/3480.htm>

### 7.1154.2.2 SharedParamsCalculateChecksum()

```
static uint16_t SharedParamsCalculateChecksum (
 void) [static]
```

Calculates and returns the checksum value for the current contents in the buffer. The checksum is calculated by taking the sum of all bytes in the parameter buffer (excluding the checksum at the end) and then taking the two's complement value of it.

#### Returns

The calculated checksum value.

Referenced by SharedParamsVerifyChecksum(), and SharedParamsWriteChecksum().

### 7.1154.2.3 SharedParamsInit()

```
void SharedParamsInit (
 void)
```

Initializes the shared RAM parameters module.

#### Returns

none.

### 7.1154.2.4 SharedParamsReadByIndex()

```
bool SharedParamsReadByIndex (
 uint32_t idx,
 uint8_t * value)
```

Reads a data byte from the shared parameter buffer at the specified index.

#### Parameters

|              |                                                                                                              |
|--------------|--------------------------------------------------------------------------------------------------------------|
| <i>idx</i>   | Index into the parameter data array. A valid value is between 0 and (SHARED_PARAMS_CFG_BUFFER_DATA_LEN - 1). |
| <i>value</i> | Pointer to where the read data value is stored.                                                              |

#### Returns

True if successful, false otherwise.

#### 7.1154.2.5 SharedParamsValidateBuffer()

```
static bool SharedParamsValidateBuffer (
 void) [static]
```

Validates the shared parameter buffer contents by looking at the table identifier and verifying its checksum.

##### Returns

True if successful, false otherwise.

Referenced by SharedParamsInit(), SharedParamsReadByIndex(), and SharedParamsWriteByIndex().

#### 7.1154.2.6 SharedParamsVerifyChecksum()

```
static bool SharedParamsVerifyChecksum (
 void) [static]
```

Calculates and verifies the checksum that is currently present in the buffer.

##### Returns

True is the checksum is correct, false otherwise.

Referenced by SharedParamsValidateBuffer().

#### 7.1154.2.7 SharedParamsWriteByIndex()

```
bool SharedParamsWriteByIndex (
 uint32_t idx,
 uint8_t value)
```

Writes a data byte to the shared parameter buffer at the specified index.

##### Parameters

|              |                                                                                                              |
|--------------|--------------------------------------------------------------------------------------------------------------|
| <i>idx</i>   | Index into the parameter data array. A valid value is between 0 and (SHARED_PARAMS_CFG_BUFFER_DATA_LEN - 1). |
| <i>value</i> | Value to write.                                                                                              |

##### Returns

True if successful, false otherwise.



### 7.1154.2.8 SharedParamsWriteChecksum()

```
static void SharedParamsWriteChecksum (
 void) [static]
```

Calculates and writes the checksum into the buffer.

#### Returns

none.

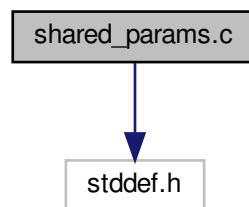
Referenced by SharedParamsInit(), and SharedParamsWriteByIndex().

## 7.1155 shared\_params.c File Reference

Shared RAM parameters source file.

```
#include <stddef.h>
#include "shared_params.h"
```

Include dependency graph for ARMCM4\_STM32F4\_Nucleo\_F429ZI\_IAR/Boot/shared\_params.c:



### Data Structures

- struct [tSharedParamsBuffer](#)  
*Layout of the shared parameters RAM buffer.*

### Macros

- #define [SHARED\\_PARAMS\\_BUFFER\\_ID](#) (0xCE42E7A2u)  
*Constant parameter buffer identifier. This value is always located as the start of the buffer to validate the the RAM contains valid shared parameters.*

## Functions

- static bool [SharedParamsValidateBuffer](#) (void)  
*Validates the shared parameter buffer contents by looking at the table identifier and verifying its checksum.*
- static void [SharedParamsWriteChecksum](#) (void)  
*Calculates and writes the checksum into the buffer.*
- static bool [SharedParamsVerifyChecksum](#) (void)  
*Calculates and verifies the checksum that is currently present in the buffer.*
- static uint16\_t [SharedParamsCalculateChecksum](#) (void)  
*Calculates and returns the checksum value for the current contents in the buffer. The checksum is calculated by taking the sum of all bytes in the parameter buffer (excluding the checksum at the end) and then taking the two's complement value of it.*
- void [SharedParamsInit](#) (void)  
*Initializes the shared RAM parameters module.*
- bool [SharedParamsReadByIndex](#) (uint32\_t idx, uint8\_t \*value)  
*Reads a data byte from the shared parameter buffer at the specified index.*
- bool [SharedParamsWriteByIndex](#) (uint32\_t idx, uint8\_t value)  
*Writes a data byte to the shared parameter buffer at the specified index.*

## Variables

- static \_\_no\_init [tSharedParamsBuffer](#) sharedParamsBuffer [shared](#)  
*Declaration of the actual parameter buffer that this module manages.*

### 7.1155.1 Detailed Description

Shared RAM parameters source file.

### 7.1155.2 Function Documentation

#### 7.1155.2.1 SharedParamsCalculateChecksum()

```
static uint16_t SharedParamsCalculateChecksum (
 void) [static]
```

Calculates and returns the checksum value for the current contents in the buffer. The checksum is calculated by taking the sum of all bytes in the parameter buffer (excluding the checksum at the end) and then taking the two's complement value of it.

#### Returns

The calculated checksum value.

Referenced by [SharedParamsVerifyChecksum\(\)](#), and [SharedParamsWriteChecksum\(\)](#).

### 7.1155.2.2 SharedParamsInit()

```
void SharedParamsInit (
 void)
```

Initializes the shared RAM parameters module.

#### Returns

none.

### 7.1155.2.3 SharedParamsReadByIndex()

```
bool SharedParamsReadByIndex (
 uint32_t idx,
 uint8_t * value)
```

Reads a data byte from the shared parameter buffer at the specified index.

#### Parameters

|              |                                                                                                              |
|--------------|--------------------------------------------------------------------------------------------------------------|
| <i>idx</i>   | Index into the parameter data array. A valid value is between 0 and (SHARED_PARAMS_CFG_BUFFER_DATA_LEN - 1). |
| <i>value</i> | Pointer to where the read data value is stored.                                                              |

#### Returns

True if successful, false otherwise.

### 7.1155.2.4 SharedParamsValidateBuffer()

```
static bool SharedParamsValidateBuffer (
 void) [static]
```

Validates the shared parameter buffer contents by looking at the table identifier and verifying its checksum.

#### Returns

True if successful, false otherwise.

Referenced by SharedParamsInit(), SharedParamsReadByIndex(), and SharedParamsWriteByIndex().

#### 7.1155.2.5 SharedParamsVerifyChecksum()

```
static bool SharedParamsVerifyChecksum (
 void) [static]
```

Calculates and verifies the checksum that is currently present in the buffer.

##### Returns

True is the checksum is correct, false otherwise.

Referenced by SharedParamsValidateBuffer().

#### 7.1155.2.6 SharedParamsWriteByIndex()

```
bool SharedParamsWriteByIndex (
 uint32_t idx,
 uint8_t value)
```

Writes a data byte to the shared parameter buffer at the specified index.

##### Parameters

|              |                                                                                                              |
|--------------|--------------------------------------------------------------------------------------------------------------|
| <i>idx</i>   | Index into the parameter data array. A valid value is between 0 and (SHARED_PARAMS_CFG_BUFFER_DATA_LEN - 1). |
| <i>value</i> | Value to write.                                                                                              |

##### Returns

True if successful, false otherwise.

#### 7.1155.2.7 SharedParamsWriteChecksum()

```
static void SharedParamsWriteChecksum (
 void) [static]
```

Calculates and writes the checksum into the buffer.

##### Returns

none.

Referenced by SharedParamsInit(), and SharedParamsWriteByIndex().

### 7.1155.3 Variable Documentation

#### 7.1155.3.1 shared

```
__no_init tSharedParamsBuffer sharedParamsBuffer shared [static]
```

Declaration of the actual parameter buffer that this module manages.

#### Warning

For the shared RAM parameters to work properly for sharing information between the bootloader and user program, it is important that this variable is linked to the exact same RAM address in both the bootloader and the user program. Additionally, it should be configured such that the C-startup code does NOT zero its contents during system initialization. This is the code that runs in the reset event handler, before function `main()` is called. For GCC based embedded toolchains, the solution is to assign this variable to a custom section, in this case called ".shared". Then in the linker script, add the following to the SECTIONS:

```
.shared (NOLOAD) : { . = ALIGN(4); _sshared = .; shared_start = _sshared; *(.shared) *(.shared.*) KE←
EP(*(.shared)) . = ALIGN(4); _eshared = .; shared_end = _eshared; } >SHARED
```

Next, add a new MEMORY entry for SHARED at the start of RAM and reduce the length of the remaining RAM:

```
SHARED (xrw) : ORIGIN = 0x200000C0, LENGTH = 64 RAM (xrw) : ORIGIN = 0x20000100, LENGTH = 32K - 192
- 64
```

Note that the previous example is for an STM32F0 microcontroller where the first 192 (0xC0) bytes in RAM are reserved for the user program vector table.

#### Remarks

This same approach can be applied with other toolchains such as Keil MDK and IAR EWARM. Consult the compiler and linker user manuals of your toolchain to find out how to place a RAM variable at a fixed memory address and to prevent the C-startup code from zeroing its contents. Here are a few links to get you started:

- IAR EWARM: <https://www.iar.com/support/tech-notes/compiler/linker-error-for-absolute-located-variable/>
- Keil MDK: <http://www.keil.com/support/docs/3480.htm>

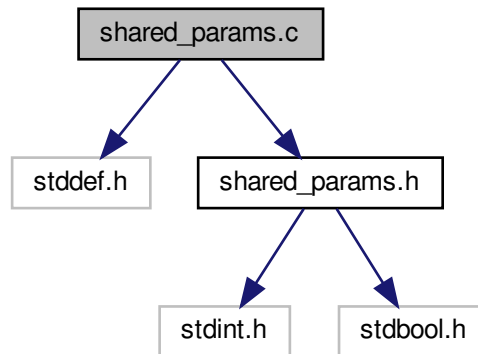
## 7.1156 shared\_params.c File Reference

Shared RAM parameters source file.

```
#include <stddef.h>
```

```
#include "shared_params.h"
```

Include dependency graph for ARMCM4\_STM32F4\_Nucleo\_F429ZI\_IAR/Prog/shared\_params.c:



## Data Structures

- struct [tSharedParamsBuffer](#)

*Layout of the shared parameters RAM buffer.*

## Macros

- `#define` [SHARED\\_PARAMS\\_BUFFER\\_ID](#) (0xCE42E7A2u)

*Constant parameter buffer identifier. This value is always located as the start of the buffer to validate the the RAM contains valid shared parameters.*

## Functions

- static bool [SharedParamsValidateBuffer](#) (void)  
*Validates the shared parameter buffer contents by looking at the table identifier and verifying its checksum.*
- static void [SharedParamsWriteChecksum](#) (void)  
*Calculates and writes the checksum into the buffer.*
- static bool [SharedParamsVerifyChecksum](#) (void)  
*Calculates and verifies the checksum that is currently present in the buffer.*
- static uint16\_t [SharedParamsCalculateChecksum](#) (void)  
*Calculates and returns the checksum value for the current contents in the buffer. The checksum is calculated by taking the sum of all bytes in the parameter buffer (excluding the checksum at the end) and then taking the two's complement value of it.*
- void [SharedParamsInit](#) (void)  
*Initializes the shared RAM parameters module.*
- bool [SharedParamsReadByIndex](#) (uint32\_t idx, uint8\_t \*value)  
*Reads a data byte from the shared parameter buffer at the specified index.*
- bool [SharedParamsWriteByIndex](#) (uint32\_t idx, uint8\_t value)  
*Writes a data byte to the shared parameter buffer at the specified index.*

## Variables

- static \_\_no\_init [tSharedParamsBuffer](#) sharedParamsBuffer [shared](#)  
*Declaration of the actual parameter buffer that this module manages.*

### 7.1156.1 Detailed Description

Shared RAM parameters source file.

### 7.1156.2 Function Documentation

#### 7.1156.2.1 SharedParamsCalculateChecksum()

```
static uint16_t SharedParamsCalculateChecksum (
 void) [static]
```

Calculates and returns the checksum value for the current contents in the buffer. The checksum is calculated by taking the sum of all bytes in the parameter buffer (excluding the checksum at the end) and then taking the two's complement value of it.

#### Returns

The calculated checksum value.

Referenced by SharedParamsVerifyChecksum(), and SharedParamsWriteChecksum().

#### 7.1156.2.2 SharedParamsInit()

```
void SharedParamsInit (
 void)
```

Initializes the shared RAM parameters module.

#### Returns

none.

#### 7.1156.2.3 SharedParamsReadByIndex()

```
bool SharedParamsReadByIndex (
 uint32_t idx,
 uint8_t * value)
```

Reads a data byte from the shared parameter buffer at the specified index.

**Parameters**

|              |                                                                                                              |
|--------------|--------------------------------------------------------------------------------------------------------------|
| <i>idx</i>   | Index into the parameter data array. A valid value is between 0 and (SHARED_PARAMS_CFG_BUFFER_DATA_LEN - 1). |
| <i>value</i> | Pointer to where the read data value is stored.                                                              |

**Returns**

True if successful, false otherwise.

**7.1156.2.4 SharedParamsValidateBuffer()**

```
static bool SharedParamsValidateBuffer (
 void) [static]
```

Validates the shared parameter buffer contents by looking at the table identifier and verifying its checksum.

**Returns**

True if successful, false otherwise.

Referenced by SharedParamsInit(), SharedParamsReadByIndex(), and SharedParamsWriteByIndex().

**7.1156.2.5 SharedParamsVerifyChecksum()**

```
static bool SharedParamsVerifyChecksum (
 void) [static]
```

Calculates and verifies the checksum that is currently present in the buffer.

**Returns**

True is the checksum is correct, false otherwise.

Referenced by SharedParamsValidateBuffer().

**7.1156.2.6 SharedParamsWriteByIndex()**

```
bool SharedParamsWriteByIndex (
 uint32_t idx,
 uint8_t value)
```

Writes a data byte to the shared parameter buffer at the specified index.



## Parameters

|              |                                                                                                              |
|--------------|--------------------------------------------------------------------------------------------------------------|
| <i>idx</i>   | Index into the parameter data array. A valid value is between 0 and (SHARED_PARAMS_CFG_BUFFER_DATA_LEN - 1). |
| <i>value</i> | Value to write.                                                                                              |

## Returns

True if successful, false otherwise.

## 7.1156.2.7 SharedParamsWriteChecksum()

```
static void SharedParamsWriteChecksum (
 void) [static]
```

Calculates and writes the checksum into the buffer.

## Returns

none.

Referenced by SharedParamsInit(), and SharedParamsWriteByIndex().

## 7.1156.3 Variable Documentation

## 7.1156.3.1 shared

```
__no_init tSharedParamsBuffer sharedParamsBuffer shared [static]
```

Declaration of the actual parameter buffer that this module manages.

## Warning

For the shared RAM parameters to work properly for sharing information between the bootloader and user program, it is important that this variable is linked to the exact same RAM address in both the bootloader and the user program. Additionally, it should be configured such that the C-startup code does NOT zero its contents during system initialization. This is the code that runs in the reset event handler, before function `main()` is called. For GCC based embedded toolchains, the solution is to assign this variable to a custom section, in this case called ".shared". Then in the linker script, add the following to the SECTIONS:

```
.shared (NOLOAD) : { . = ALIGN(4); _sshared = .; shared_start = _sshared; *(.shared) *(.shared.*) KE←
EP(*(.shared)) . = ALIGN(4); _eshared = .; shared_end = _eshared; } >SHARED
```

Next, add a new MEMORY entry for SHARED at the start of RAM and reduce the length of the remaining RAM:

```
SHARED (xrw) : ORIGIN = 0x200000C0, LENGTH = 64 RAM (xrw) : ORIGIN = 0x20000100, LENGTH = 32K - 192
- 64
```

Note that the previous example is for an STM32F0 microcontroller where the first 192 (0xC0) bytes in RAM are reserved for the user program vector table.

## Remarks

This same approach can be applied with other toolchains such as Keil MDK and IAR EWARM. Consult the compiler and linker user manuals of your toolchain to find out how to place a RAM variable at a fixed memory address and to prevent the C-startup code from zeroing its contents. Here are a few links to get you started:

- IAR EWARM: <https://www.iar.com/support/tech-notes/compiler/linker-error-for-absolute-located-variable/>
- Keil MDK: <http://www.keil.com/support/docs/3480.htm>

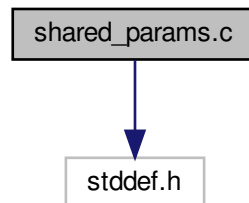
## 7.1157 shared\_params.c File Reference

Shared RAM parameters source file.

```
#include <stddef.h>
```

```
#include "shared_params.h"
```

Include dependency graph for ARMCM4\_STM32F4\_Nucleo\_F429ZI\_Keil/Boot/shared\_params.c:



### Data Structures

- struct [tSharedParamsBuffer](#)  
*Layout of the shared parameters RAM buffer.*

### Macros

- #define [SHARED\\_PARAMS\\_BUFFER\\_ID](#) (0xCE42E7A2u)  
*Constant parameter buffer identifier. This value is always located as the start of the buffer to validate the the RAM contains valid shared parameters.*

### Functions

- static [tSharedParamsBuffer](#) sharedParamsBuffer [\\_\\_attribute\\_\\_](#)((section("shared"), zero\_init))  
*Declaration of the actual parameter buffer that this module manages.*
- static bool [SharedParamsValidateBuffer](#) (void)  
*Validates the shared parameter buffer contents by looking at the table identifier and verifying its checksum.*
- static void [SharedParamsWriteChecksum](#) (void)  
*Calculates and writes the checksum into the buffer.*
- static bool [SharedParamsVerifyChecksum](#) (void)  
*Calculates and verifies the checksum that is currently present in the buffer.*
- static uint16\_t [SharedParamsCalculateChecksum](#) (void)  
*Calculates and returns the checksum value for the current contents in the buffer. The checksum is calculated by taking the sum of all bytes in the parameter buffer (excluding the checksum at the end) and then taking the two's complement value of it.*
- void [SharedParamsInit](#) (void)  
*Initializes the shared RAM parameters module.*
- bool [SharedParamsReadByIndex](#) (uint32\_t idx, uint8\_t \*value)  
*Reads a data byte from the shared parameter buffer at the specified index.*
- bool [SharedParamsWriteByIndex](#) (uint32\_t idx, uint8\_t value)  
*Writes a data byte to the shared parameter buffer at the specified index.*

### 7.1157.1 Detailed Description

Shared RAM parameters source file.

### 7.1157.2 Function Documentation

#### 7.1157.2.1 \_\_attribute\_\_((section("shared"), zero\_init)) static

```
static tSharedParamsBuffer sharedParamsBuffer __attribute__((
 (section("shared"), zero_init))) [static]
```

Declaration of the actual parameter buffer that this module manages.

#### Warning

For the shared RAM parameters to work properly for sharing information between the bootloader and user program, it is important that this variable is linked to the exact same RAM address in both the bootloader and the user program. Additionally, it should be configured such that the C-startup code does NOT zero its contents during system initialization. This is the code that runs in the reset event handler, before function `main()` is called. For GCC based embedded toolchains, the solution is to assign this variable to a custom section, in this case called ".shared". Then in the linker script, add the following to the SECTIONS:

```
.shared (NOLOAD) : { . = ALIGN(4); _sshared = .; shared_start = _sshared; *(.shared) *(.shared.*) KE←
EP(*(.shared)) . = ALIGN(4); _eshared = .; shared_end = _eshared; } >SHARED
```

Next, add a new MEMORY entry for SHARED at the start of RAM and reduce the length of the remaining RAM:

```
SHARED (xrw) : ORIGIN = 0x200000C0, LENGTH = 64 RAM (xrw) : ORIGIN = 0x20000100, LENGTH = 32K - 192
- 64
```

Note that the previous example is for an STM32F0 microcontroller where the first 192 (0xC0) bytes in RAM are reserved for the user program vector table.

#### Remarks

This same approach can be applied with other toolchains such as Keil MDK and IAR EWARM. Consult the compiler and linker user manuals of your toolchain to find out how to place a RAM variable at a fixed memory address and to prevent the C-startup code from zeroing its contents. Here are a few links to get you started:

- IAR EWARM: <https://www.iar.com/support/tech-notes/compiler/linker-error-for-absolute-located-variable/>
- Keil MDK: <http://www.keil.com/support/docs/3480.htm>

### 7.1157.2.2 SharedParamsCalculateChecksum()

```
static uint16_t SharedParamsCalculateChecksum (
 void) [static]
```

Calculates and returns the checksum value for the current contents in the buffer. The checksum is calculated by taking the sum of all bytes in the parameter buffer (excluding the checksum at the end) and then taking the two's complement value of it.

#### Returns

The calculated checksum value.

Referenced by SharedParamsVerifyChecksum(), and SharedParamsWriteChecksum().

### 7.1157.2.3 SharedParamsInit()

```
void SharedParamsInit (
 void)
```

Initializes the shared RAM parameters module.

#### Returns

none.

### 7.1157.2.4 SharedParamsReadByIndex()

```
bool SharedParamsReadByIndex (
 uint32_t idx,
 uint8_t * value)
```

Reads a data byte from the shared parameter buffer at the specified index.

#### Parameters

|              |                                                                                                              |
|--------------|--------------------------------------------------------------------------------------------------------------|
| <i>idx</i>   | Index into the parameter data array. A valid value is between 0 and (SHARED_PARAMS_CFG_BUFFER_DATA_LEN - 1). |
| <i>value</i> | Pointer to where the read data value is stored.                                                              |

#### Returns

True if successful, false otherwise.

### 7.1157.2.5 SharedParamsValidateBuffer()

```
static bool SharedParamsValidateBuffer (
 void) [static]
```

Validates the shared parameter buffer contents by looking at the table identifier and verifying its checksum.

#### Returns

True if successful, false otherwise.

Referenced by SharedParamsInit(), SharedParamsReadByIndex(), and SharedParamsWriteByIndex().

### 7.1157.2.6 SharedParamsVerifyChecksum()

```
static bool SharedParamsVerifyChecksum (
 void) [static]
```

Calculates and verifies the checksum that is currently present in the buffer.

#### Returns

True is the checksum is correct, false otherwise.

Referenced by SharedParamsValidateBuffer().

### 7.1157.2.7 SharedParamsWriteByIndex()

```
bool SharedParamsWriteByIndex (
 uint32_t idx,
 uint8_t value)
```

Writes a data byte to the shared parameter buffer at the specified index.

#### Parameters

|              |                                                                                                              |
|--------------|--------------------------------------------------------------------------------------------------------------|
| <i>idx</i>   | Index into the parameter data array. A valid value is between 0 and (SHARED_PARAMS_CFG_BUFFER_DATA_LEN - 1). |
| <i>value</i> | Value to write.                                                                                              |

#### Returns

True if successful, false otherwise.

### 7.1157.2.8 SharedParamsWriteChecksum()

```
static void SharedParamsWriteChecksum (
 void) [static]
```

Calculates and writes the checksum into the buffer.

#### Returns

none.

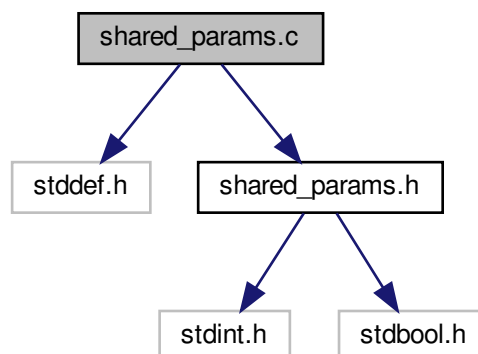
Referenced by SharedParamsInit(), and SharedParamsWriteByIndex().

## 7.1158 shared\_params.c File Reference

Shared RAM parameters source file.

```
#include <stddef.h>
#include "shared_params.h"
```

Include dependency graph for ARMCM4\_STM32F4\_Nucleo\_F429ZI\_Keil/Prog/shared\_params.c:



## Data Structures

- struct [tSharedParamsBuffer](#)

*Layout of the shared parameters RAM buffer.*

## Macros

- #define [SHARED\\_PARAMS\\_BUFFER\\_ID](#) (0xCE42E7A2u)

*Constant parameter buffer identifier. This value is always located as the start of the buffer to validate the the RAM contains valid shared parameters.*

## Functions

- static `tSharedParamsBuffer` sharedParamsBuffer `__attribute__` ((section("shared"), zero\_init))  
*Declaration of the actual parameter buffer that this module manages.*
- static bool `SharedParamsValidateBuffer` (void)  
*Validates the shared parameter buffer contents by looking at the table identifier and verifying its checksum.*
- static void `SharedParamsWriteChecksum` (void)  
*Calculates and writes the checksum into the buffer.*
- static bool `SharedParamsVerifyChecksum` (void)  
*Calculates and verifies the checksum that is currently present in the buffer.*
- static uint16\_t `SharedParamsCalculateChecksum` (void)  
*Calculates and returns the checksum value for the current contents in the buffer. The checksum is calculated by taking the sum of all bytes in the parameter buffer (excluding the checksum at the end) and then taking the two's complement value of it.*
- void `SharedParamsInit` (void)  
*Initializes the shared RAM parameters module.*
- bool `SharedParamsReadByIndex` (uint32\_t idx, uint8\_t \*value)  
*Reads a data byte from the shared parameter buffer at the specified index.*
- bool `SharedParamsWriteByIndex` (uint32\_t idx, uint8\_t value)  
*Writes a data byte to the shared parameter buffer at the specified index.*

### 7.1158.1 Detailed Description

Shared RAM parameters source file.

### 7.1158.2 Function Documentation

#### 7.1158.2.1 `__attribute__`()

```
static tSharedParamsBuffer sharedParamsBuffer __attribute__ (
 (section("shared"), zero_init)) [static]
```

Declaration of the actual parameter buffer that this module manages.

#### Warning

For the shared RAM parameters to work properly for sharing information between the bootloader and user program, it is important that this variable is linked to the exact same RAM address in both the bootloader and the user program. Additionally, it should be configured such that the C-startup code does NOT zero its contents during system initialization. This is the code that runs in the reset event handler, before function `main()` is called. For GCC based embedded toolchains, the solution is to assign this variable to a custom section, in this case called ".shared". Then in the linker script, add the following to the SECTIONS:

```
.shared (NOLOAD) : { . = ALIGN(4); _sshared = .; shared_start = _sshared; *(.shared) *(.shared.*) KE←
EP(*(.shared)) . = ALIGN(4); _eshared = .; shared_end = _eshared; } >SHARED
```

Next, add a new MEMORY entry for SHARED at the start of RAM and reduce the length of the remaining RAM:

```
SHARED (xrw) : ORIGIN = 0x200000C0, LENGTH = 64 RAM (xrw) : ORIGIN = 0x20000100, LENGTH = 32K - 192
- 64
```

Note that the previous example is for an STM32F0 microcontroller where the first 192 (0xC0) bytes in RAM are reserved for the user program vector table.

## Remarks

This same approach can be applied with other toolchains such as Keil MDK and IAR EWARM. Consult the compiler and linker user manuals of your toolchain to find out how to place a RAM variable at a fixed memory address and to prevent the C-startup code from zeroing its contents. Here are a few links to get you started:

- IAR EWARM: <https://www.iar.com/support/tech-notes/compiler/linker-error-for-absolute-located-variable/>
- Keil MDK: <http://www.keil.com/support/docs/3480.htm>

### 7.1158.2.2 SharedParamsCalculateChecksum()

```
static uint16_t SharedParamsCalculateChecksum (
 void) [static]
```

Calculates and returns the checksum value for the current contents in the buffer. The checksum is calculated by taking the sum of all bytes in the parameter buffer (excluding the checksum at the end) and then taking the two's complement value of it.

## Returns

The calculated checksum value.

Referenced by SharedParamsVerifyChecksum(), and SharedParamsWriteChecksum().

### 7.1158.2.3 SharedParamsInit()

```
void SharedParamsInit (
 void)
```

Initializes the shared RAM parameters module.

## Returns

none.

### 7.1158.2.4 SharedParamsReadByIndex()

```
bool SharedParamsReadByIndex (
 uint32_t idx,
 uint8_t * value)
```

Reads a data byte from the shared parameter buffer at the specified index.



**Parameters**

|              |                                                                                                              |
|--------------|--------------------------------------------------------------------------------------------------------------|
| <i>idx</i>   | Index into the parameter data array. A valid value is between 0 and (SHARED_PARAMS_CFG_BUFFER_DATA_LEN - 1). |
| <i>value</i> | Pointer to where the read data value is stored.                                                              |

**Returns**

True if successful, false otherwise.

**7.1158.2.5 SharedParamsValidateBuffer()**

```
static bool SharedParamsValidateBuffer (
 void) [static]
```

Validates the shared parameter buffer contents by looking at the table identifier and verifying its checksum.

**Returns**

True if successful, false otherwise.

Referenced by SharedParamsInit(), SharedParamsReadByIndex(), and SharedParamsWriteByIndex().

**7.1158.2.6 SharedParamsVerifyChecksum()**

```
static bool SharedParamsVerifyChecksum (
 void) [static]
```

Calculates and verifies the checksum that is currently present in the buffer.

**Returns**

True is the checksum is correct, false otherwise.

Referenced by SharedParamsValidateBuffer().

**7.1158.2.7 SharedParamsWriteByIndex()**

```
bool SharedParamsWriteByIndex (
 uint32_t idx,
 uint8_t value)
```

Writes a data byte to the shared parameter buffer at the specified index.

**Parameters**

|              |                                                                                                              |
|--------------|--------------------------------------------------------------------------------------------------------------|
| <i>idx</i>   | Index into the parameter data array. A valid value is between 0 and (SHARED_PARAMS_CFG_BUFFER_DATA_LEN - 1). |
| <i>value</i> | Value to write.                                                                                              |

**Returns**

True if successful, false otherwise.

**7.1158.2.8 SharedParamsWriteChecksum()**

```
static void SharedParamsWriteChecksum (
 void) [static]
```

Calculates and writes the checksum into the buffer.

**Returns**

none.

Referenced by SharedParamsInit(), and SharedParamsWriteByIndex().

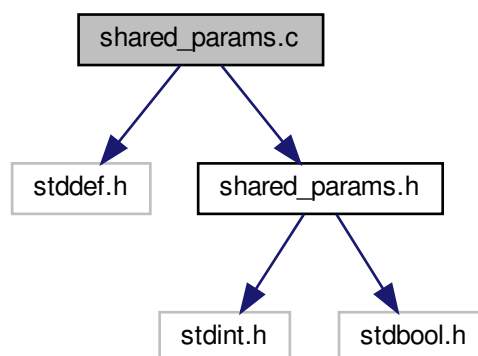
**7.1159 shared\_params.c File Reference**

Shared RAM parameters source file.

```
#include <stddef.h>
```

```
#include "shared_params.h"
```

Include dependency graph for ARMC4\_XMC4700\_Relax\_Kit\_GCC/Boot/shared\_params.c:



## Data Structures

- struct [tSharedParamsBuffer](#)

*Layout of the shared parameters RAM buffer.*

## Macros

- #define [SHARED\\_PARAMS\\_BUFFER\\_ID](#) (0xCE42E7A2u)

*Constant parameter buffer identifier. This value is always located as the start of the buffer to validate the the RAM contains valid shared parameters.*

## Functions

- static [tSharedParamsBuffer](#) sharedParamsBuffer [\\_\\_attribute\\_\\_](#) ((section(".shared")))

*Declaration of the actual parameter buffer that this module manages.*

- static bool [SharedParamsValidateBuffer](#) (void)

*Validates the shared parameter buffer contents by looking at the table identifier and verifying its checksum.*

- static void [SharedParamsWriteChecksum](#) (void)

*Calculates and writes the checksum into the buffer.*

- static bool [SharedParamsVerifyChecksum](#) (void)

*Calculates and verifies the checksum that is currently present in the buffer.*

- static uint16\_t [SharedParamsCalculateChecksum](#) (void)

*Calculates and returns the checksum value for the current contents in the buffer. The checksum is calculated by taking the sum of all bytes in the parameter buffer (excluding the checksum at the end) and then taking the two's complement value of it.*

- void [SharedParamsInit](#) (void)

*Initializes the shared RAM parameters module.*

- bool [SharedParamsReadByIndex](#) (uint32\_t idx, uint8\_t \*value)

*Reads a data byte from the shared parameter buffer at the specified index.*

- bool [SharedParamsWriteByIndex](#) (uint32\_t idx, uint8\_t value)

*Writes a data byte to the shared parameter buffer at the specified index.*

### 7.1159.1 Detailed Description

Shared RAM parameters source file.

### 7.1159.2 Function Documentation

### 7.1159.2.1 \_\_attribute\_\_((section(\".shared\")))

```
static tSharedParamsBuffer sharedParamsBuffer __attribute__((section(\".shared\"))) [static]
```

Declaration of the actual parameter buffer that this module manages.

#### Warning

For the shared RAM parameters to work properly for sharing information between the bootloader and user program, it is important that this variable is linked to the exact same RAM address in both the bootloader and the user program. Additionally, it should be configured such that the C-startup code does NOT zero its contents during system initialization. This is the code that runs in the reset event handler, before function `main()` is called. For GCC based embedded toolchains, the solution is to assign this variable to a custom section, in this case called `\".shared\"`. Then in the linker script, add the following to the SECTIONS:

```
\".shared (NOLOAD) : { . = ALIGN(4); _sshared = .; shared_start = _sshared; *(.shared) *(.shared.*) KE←
EP(*(.shared)) . = ALIGN(4); _eshared = .; shared_end = _eshared; } >SHARED
```

Next, add a new MEMORY entry for SHARED at the start of RAM and reduce the length of the remaining RAM:

```
SHARED (xrw) : ORIGIN = 0x200000C0, LENGTH = 64 RAM (xrw) : ORIGIN = 0x20000100, LENGTH = 32K - 192
- 64
```

Note that the previous example is for an STM32F0 microcontroller where the first 192 (0xC0) bytes in RAM are reserved for the user program vector table.

#### Remarks

This same approach can be applied with other toolchains such as Keil MDK and IAR EWARM. Consult the compiler and linker user manuals of your toolchain to find out how to place a RAM variable at a fixed memory address and to prevent the C-startup code from zeroing its contents. Here are a few links to get you started:

- IAR EWARM: <https://www.iar.com/support/tech-notes/compiler/linker-error-for-absolute-located-variable/>
- Keil MDK: <http://www.keil.com/support/docs/3480.htm>

### 7.1159.2.2 SharedParamsCalculateChecksum()

```
static uint16_t SharedParamsCalculateChecksum (
 void) [static]
```

Calculates and returns the checksum value for the current contents in the buffer. The checksum is calculated by taking the sum of all bytes in the parameter buffer (excluding the checksum at the end) and then taking the two's complement value of it.

#### Returns

The calculated checksum value.

Referenced by `SharedParamsVerifyChecksum()`, and `SharedParamsWriteChecksum()`.

### 7.1159.2.3 SharedParamsInit()

```
void SharedParamsInit (
 void)
```

Initializes the shared RAM parameters module.

#### Returns

none.

### 7.1159.2.4 SharedParamsReadByIndex()

```
bool SharedParamsReadByIndex (
 uint32_t idx,
 uint8_t * value)
```

Reads a data byte from the shared parameter buffer at the specified index.

#### Parameters

|              |                                                                                                              |
|--------------|--------------------------------------------------------------------------------------------------------------|
| <i>idx</i>   | Index into the parameter data array. A valid value is between 0 and (SHARED_PARAMS_CFG_BUFFER_DATA_LEN - 1). |
| <i>value</i> | Pointer to where the read data value is stored.                                                              |

#### Returns

True if successful, false otherwise.

### 7.1159.2.5 SharedParamsValidateBuffer()

```
static bool SharedParamsValidateBuffer (
 void) [static]
```

Validates the shared parameter buffer contents by looking at the table identifier and verifying its checksum.

#### Returns

True if successful, false otherwise.

Referenced by SharedParamsInit(), SharedParamsReadByIndex(), and SharedParamsWriteByIndex().

### 7.1159.2.6 SharedParamsVerifyChecksum()

```
static bool SharedParamsVerifyChecksum (
 void) [static]
```

Calculates and verifies the checksum that is currently present in the buffer.

#### Returns

True is the checksum is correct, false otherwise.

Referenced by SharedParamsValidateBuffer().

### 7.1159.2.7 SharedParamsWriteByIndex()

```
bool SharedParamsWriteByIndex (
 uint32_t idx,
 uint8_t value)
```

Writes a data byte to the shared parameter buffer at the specified index.

#### Parameters

|              |                                                                                                              |
|--------------|--------------------------------------------------------------------------------------------------------------|
| <i>idx</i>   | Index into the parameter data array. A valid value is between 0 and (SHARED_PARAMS_CFG_BUFFER_DATA_LEN - 1). |
| <i>value</i> | Value to write.                                                                                              |

#### Returns

True if successful, false otherwise.

### 7.1159.2.8 SharedParamsWriteChecksum()

```
static void SharedParamsWriteChecksum (
 void) [static]
```

Calculates and writes the checksum into the buffer.

#### Returns

none.

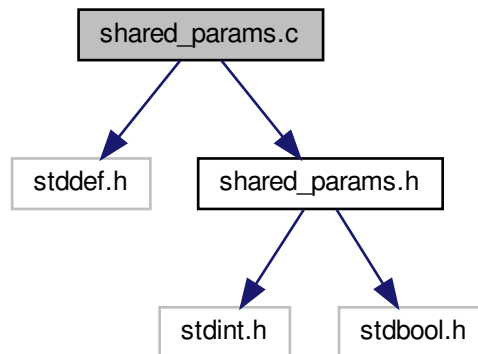
Referenced by SharedParamsInit(), and SharedParamsWriteByIndex().

## 7.1160 shared\_params.c File Reference

Shared RAM parameters source file.

```
#include <stddef.h>
#include "shared_params.h"
```

Include dependency graph for ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_GCC/Prog/shared\_params.c:



### Data Structures

- struct [tSharedParamsBuffer](#)

*Layout of the shared parameters RAM buffer.*

### Macros

- `#define` [SHARED\\_PARAMS\\_BUFFER\\_ID](#) (0xCE42E7A2u)

*Constant parameter buffer identifier. This value is always located as the start of the buffer to validate the the RAM contains valid shared parameters.*

### Functions

- static [tSharedParamsBuffer](#) `sharedParamsBuffer` `__attribute__((section(".shared")))`  
*Declaration of the actual parameter buffer that this module manages.*
- static bool [SharedParamsValidateBuffer](#) (void)  
*Validates the shared parameter buffer contents by looking at the table identifier and verifying its checksum.*
- static void [SharedParamsWriteChecksum](#) (void)  
*Calculates and writes the checksum into the buffer.*
- static bool [SharedParamsVerifyChecksum](#) (void)  
*Calculates and verifies the checksum that is currently present in the buffer.*
- static uint16\_t [SharedParamsCalculateChecksum](#) (void)

*Calculates and returns the checksum value for the current contents in the buffer. The checksum is calculated by taking the sum of all bytes in the parameter buffer (excluding the checksum at the end) and then taking the two's complement value of it.*

- void [SharedParamsInit](#) (void)  
*Initializes the shared RAM parameters module.*
- bool [SharedParamsReadByIndex](#) (uint32\_t idx, uint8\_t \*value)  
*Reads a data byte from the shared parameter buffer at the specified index.*
- bool [SharedParamsWriteByIndex](#) (uint32\_t idx, uint8\_t value)  
*Writes a data byte to the shared parameter buffer at the specified index.*

### 7.1160.1 Detailed Description

Shared RAM parameters source file.

### 7.1160.2 Function Documentation

#### 7.1160.2.1 `__attribute__()`

```
static tSharedParamsBuffer sharedParamsBuffer __attribute__ (
 (section(".shared"))) [static]
```

Declaration of the actual parameter buffer that this module manages.

#### Warning

For the shared RAM parameters to work properly for sharing information between the bootloader and user program, it is important that this variable is linked to the exact same RAM address in both the bootloader and the user program. Additionally, it should be configured such that the C-startup code does NOT zero its contents during system initialization. This is the code that runs in the reset event handler, before function [main\(\)](#) is called. For GCC based embedded toolchains, the solution is to assign this variable to a custom section, in this case called ".shared". Then in the linker script, add the following to the SECTIONS:

```
.shared (NOLOAD) : { . = ALIGN(4); _sshared = .; shared_start = _sshared; *(.shared) *(.shared.*) KE↔
EP(*(.shared)) . = ALIGN(4); _eshared = .; shared_end = _eshared; } >SHARED
```

Next, add a new MEMORY entry for SHARED at the start of RAM and reduce the length of the remaining RAM:

```
SHARED (xrw) : ORIGIN = 0x200000C0, LENGTH = 64 RAM (xrw) : ORIGIN = 0x20000100, LENGTH = 32K - 192
- 64
```

Note that the previous example is for an STM32F0 microcontroller where the first 192 (0xC0) bytes in RAM are reserved for the user program vector table.

#### Remarks

This same approach can be applied with other toolchains such as Keil MDK and IAR EWARM. Consult the compiler and linker user manuals of your toolchain to find out how to place a RAM variable at a fixed memory address and to prevent the C-startup code from zeroing its contents. Here are a few links to get you started:

- IAR EWARM: <https://www.iar.com/support/tech-notes/compiler/linker-error-for-absolute-located-variable/>
- Keil MDK: <http://www.keil.com/support/docs/3480.htm>



### 7.1160.2.2 SharedParamsCalculateChecksum()

```
static uint16_t SharedParamsCalculateChecksum (
 void) [static]
```

Calculates and returns the checksum value for the current contents in the buffer. The checksum is calculated by taking the sum of all bytes in the parameter buffer (excluding the checksum at the end) and then taking the two's complement value of it.

#### Returns

The calculated checksum value.

Referenced by SharedParamsVerifyChecksum(), and SharedParamsWriteChecksum().

### 7.1160.2.3 SharedParamsInit()

```
void SharedParamsInit (
 void)
```

Initializes the shared RAM parameters module.

#### Returns

none.

### 7.1160.2.4 SharedParamsReadByIndex()

```
bool SharedParamsReadByIndex (
 uint32_t idx,
 uint8_t * value)
```

Reads a data byte from the shared parameter buffer at the specified index.

#### Parameters

|              |                                                                                                              |
|--------------|--------------------------------------------------------------------------------------------------------------|
| <i>idx</i>   | Index into the parameter data array. A valid value is between 0 and (SHARED_PARAMS_CFG_BUFFER_DATA_LEN - 1). |
| <i>value</i> | Pointer to where the read data value is stored.                                                              |

#### Returns

True if successful, false otherwise.

#### 7.1160.2.5 SharedParamsValidateBuffer()

```
static bool SharedParamsValidateBuffer (
 void) [static]
```

Validates the shared parameter buffer contents by looking at the table identifier and verifying its checksum.

##### Returns

True if successful, false otherwise.

Referenced by SharedParamsInit(), SharedParamsReadByIndex(), and SharedParamsWriteByIndex().

#### 7.1160.2.6 SharedParamsVerifyChecksum()

```
static bool SharedParamsVerifyChecksum (
 void) [static]
```

Calculates and verifies the checksum that is currently present in the buffer.

##### Returns

True is the checksum is correct, false otherwise.

Referenced by SharedParamsValidateBuffer().

#### 7.1160.2.7 SharedParamsWriteByIndex()

```
bool SharedParamsWriteByIndex (
 uint32_t idx,
 uint8_t value)
```

Writes a data byte to the shared parameter buffer at the specified index.

##### Parameters

|              |                                                                                                              |
|--------------|--------------------------------------------------------------------------------------------------------------|
| <i>idx</i>   | Index into the parameter data array. A valid value is between 0 and (SHARED_PARAMS_CFG_BUFFER_DATA_LEN - 1). |
| <i>value</i> | Value to write.                                                                                              |

##### Returns

True if successful, false otherwise.

## 7.1160.2.8 SharedParamsWriteChecksum()

```
static void SharedParamsWriteChecksum (
 void) [static]
```

Calculates and writes the checksum into the buffer.

## Returns

none.

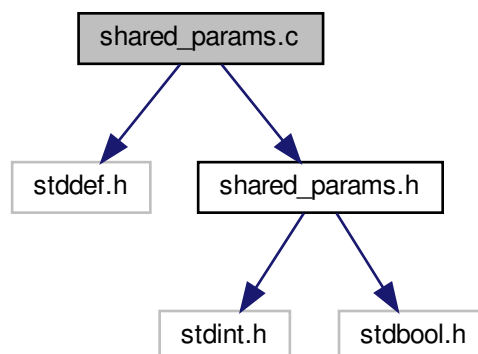
Referenced by SharedParamsInit(), and SharedParamsWriteByIndex().

## 7.1161 shared\_params.c File Reference

Shared RAM parameters source file.

```
#include <stddef.h>
#include "shared_params.h"
```

Include dependency graph for ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_IAR/Boot/shared\_params.c:



## Data Structures

- struct [tSharedParamsBuffer](#)

*Layout of the shared parameters RAM buffer.*

## Macros

- #define [SHARED\\_PARAMS\\_BUFFER\\_ID](#) (0xCE42E7A2u)

*Constant parameter buffer identifier. This value is always located as the start of the buffer to validate the the RAM contains valid shared parameters.*

## Functions

- static bool [SharedParamsValidateBuffer](#) (void)  
*Validates the shared parameter buffer contents by looking at the table identifier and verifying its checksum.*
- static void [SharedParamsWriteChecksum](#) (void)  
*Calculates and writes the checksum into the buffer.*
- static bool [SharedParamsVerifyChecksum](#) (void)  
*Calculates and verifies the checksum that is currently present in the buffer.*
- static uint16\_t [SharedParamsCalculateChecksum](#) (void)  
*Calculates and returns the checksum value for the current contents in the buffer. The checksum is calculated by taking the sum of all bytes in the parameter buffer (excluding the checksum at the end) and then taking the two's complement value of it.*
- void [SharedParamsInit](#) (void)  
*Initializes the shared RAM parameters module.*
- bool [SharedParamsReadByIndex](#) (uint32\_t idx, uint8\_t \*value)  
*Reads a data byte from the shared parameter buffer at the specified index.*
- bool [SharedParamsWriteByIndex](#) (uint32\_t idx, uint8\_t value)  
*Writes a data byte to the shared parameter buffer at the specified index.*

## Variables

- static \_\_no\_init [tSharedParamsBuffer](#) sharedParamsBuffer [shared](#)  
*Declaration of the actual parameter buffer that this module manages.*

### 7.1161.1 Detailed Description

Shared RAM parameters source file.

### 7.1161.2 Function Documentation

#### 7.1161.2.1 SharedParamsCalculateChecksum()

```
static uint16_t SharedParamsCalculateChecksum (
 void) [static]
```

Calculates and returns the checksum value for the current contents in the buffer. The checksum is calculated by taking the sum of all bytes in the parameter buffer (excluding the checksum at the end) and then taking the two's complement value of it.

#### Returns

The calculated checksum value.

Referenced by [SharedParamsVerifyChecksum\(\)](#), and [SharedParamsWriteChecksum\(\)](#).

### 7.1161.2.2 SharedParamsInit()

```
void SharedParamsInit (
 void)
```

Initializes the shared RAM parameters module.

#### Returns

none.

### 7.1161.2.3 SharedParamsReadByIndex()

```
bool SharedParamsReadByIndex (
 uint32_t idx,
 uint8_t * value)
```

Reads a data byte from the shared parameter buffer at the specified index.

#### Parameters

|              |                                                                                                              |
|--------------|--------------------------------------------------------------------------------------------------------------|
| <i>idx</i>   | Index into the parameter data array. A valid value is between 0 and (SHARED_PARAMS_CFG_BUFFER_DATA_LEN - 1). |
| <i>value</i> | Pointer to where the read data value is stored.                                                              |

#### Returns

True if successful, false otherwise.

### 7.1161.2.4 SharedParamsValidateBuffer()

```
static bool SharedParamsValidateBuffer (
 void) [static]
```

Validates the shared parameter buffer contents by looking at the table identifier and verifying its checksum.

#### Returns

True if successful, false otherwise.

Referenced by SharedParamsInit(), SharedParamsReadByIndex(), and SharedParamsWriteByIndex().

#### 7.1161.2.5 SharedParamsVerifyChecksum()

```
static bool SharedParamsVerifyChecksum (
 void) [static]
```

Calculates and verifies the checksum that is currently present in the buffer.

##### Returns

True is the checksum is correct, false otherwise.

Referenced by SharedParamsValidateBuffer().

#### 7.1161.2.6 SharedParamsWriteByIndex()

```
bool SharedParamsWriteByIndex (
 uint32_t idx,
 uint8_t value)
```

Writes a data byte to the shared parameter buffer at the specified index.

##### Parameters

|              |                                                                                                              |
|--------------|--------------------------------------------------------------------------------------------------------------|
| <i>idx</i>   | Index into the parameter data array. A valid value is between 0 and (SHARED_PARAMS_CFG_BUFFER_DATA_LEN - 1). |
| <i>value</i> | Value to write.                                                                                              |

##### Returns

True if successful, false otherwise.

#### 7.1161.2.7 SharedParamsWriteChecksum()

```
static void SharedParamsWriteChecksum (
 void) [static]
```

Calculates and writes the checksum into the buffer.

##### Returns

none.

Referenced by SharedParamsInit(), and SharedParamsWriteByIndex().

### 7.1161.3 Variable Documentation

#### 7.1161.3.1 shared

```
__no_init tSharedParamsBuffer sharedParamsBuffer shared [static]
```

Declaration of the actual parameter buffer that this module manages.

#### Warning

For the shared RAM parameters to work properly for sharing information between the bootloader and user program, it is important that this variable is linked to the exact same RAM address in both the bootloader and the user program. Additionally, it should be configured such that the C-startup code does NOT zero its contents during system initialization. This is the code that runs in the reset event handler, before function `main()` is called. For GCC based embedded toolchains, the solution is to assign this variable to a custom section, in this case called ".shared". Then in the linker script, add the following to the SECTIONS:

```
.shared (NOLOAD) : { . = ALIGN(4); _sshared = .; shared_start = _sshared; *(.shared) *(.shared.*) KE←
EP(*(.(shared))) . = ALIGN(4); _eshared = .; shared_end = _eshared; } >SHARED
```

Next, add a new MEMORY entry for SHARED at the start of RAM and reduce the length of the remaining RAM:

```
SHARED (xrw) : ORIGIN = 0x200000C0, LENGTH = 64 RAM (xrw) : ORIGIN = 0x20000100, LENGTH = 32K - 192
- 64
```

Note that the previous example is for an STM32F0 microcontroller where the first 192 (0xC0) bytes in RAM are reserved for the user program vector table.

#### Remarks

This same approach can be applied with other toolchains such as Keil MDK and IAR EWARM. Consult the compiler and linker user manuals of your toolchain to find out how to place a RAM variable at a fixed memory address and to prevent the C-startup code from zeroing its contents. Here are a few links to get you started:

- IAR EWARM: <https://www.iar.com/support/tech-notes/compiler/linker-error-for-absolute-located-variable/>
- Keil MDK: <http://www.keil.com/support/docs/3480.htm>

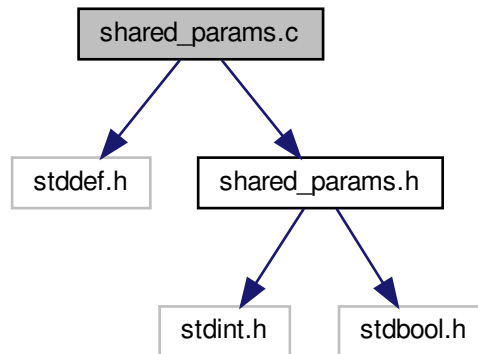
## 7.1162 shared\_params.c File Reference

Shared RAM parameters source file.

```
#include <stddef.h>
```

```
#include "shared_params.h"
```

Include dependency graph for ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_IAR/Prog/shared\_params.c:



## Data Structures

- struct [tSharedParamsBuffer](#)

*Layout of the shared parameters RAM buffer.*

## Macros

- `#define` [SHARED\\_PARAMS\\_BUFFER\\_ID](#) (0xCE42E7A2u)

*Constant parameter buffer identifier. This value is always located as the start of the buffer to validate the the RAM contains valid shared parameters.*

## Functions

- static bool [SharedParamsValidateBuffer](#) (void)  
*Validates the shared parameter buffer contents by looking at the table identifier and verifying its checksum.*
- static void [SharedParamsWriteChecksum](#) (void)  
*Calculates and writes the checksum into the buffer.*
- static bool [SharedParamsVerifyChecksum](#) (void)  
*Calculates and verifies the checksum that is currently present in the buffer.*
- static uint16\_t [SharedParamsCalculateChecksum](#) (void)  
*Calculates and returns the checksum value for the current contents in the buffer. The checksum is calculated by taking the sum of all bytes in the parameter buffer (excluding the checksum at the end) and then taking the two's complement value of it.*
- void [SharedParamsInit](#) (void)  
*Initializes the shared RAM parameters module.*
- bool [SharedParamsReadByIndex](#) (uint32\_t idx, uint8\_t \*value)  
*Reads a data byte from the shared parameter buffer at the specified index.*
- bool [SharedParamsWriteByIndex](#) (uint32\_t idx, uint8\_t value)  
*Writes a data byte to the shared parameter buffer at the specified index.*



## Variables

- static \_\_no\_init [tSharedParamsBuffer](#) sharedParamsBuffer [shared](#)  
*Declaration of the actual parameter buffer that this module manages.*

### 7.1162.1 Detailed Description

Shared RAM parameters source file.

### 7.1162.2 Function Documentation

#### 7.1162.2.1 SharedParamsCalculateChecksum()

```
static uint16_t SharedParamsCalculateChecksum (
 void) [static]
```

Calculates and returns the checksum value for the current contents in the buffer. The checksum is calculated by taking the sum of all bytes in the parameter buffer (excluding the checksum at the end) and then taking the two's complement value of it.

#### Returns

The calculated checksum value.

Referenced by SharedParamsVerifyChecksum(), and SharedParamsWriteChecksum().

#### 7.1162.2.2 SharedParamsInit()

```
void SharedParamsInit (
 void)
```

Initializes the shared RAM parameters module.

#### Returns

none.

#### 7.1162.2.3 SharedParamsReadByIndex()

```
bool SharedParamsReadByIndex (
 uint32_t idx,
 uint8_t * value)
```

Reads a data byte from the shared parameter buffer at the specified index.

**Parameters**

|              |                                                                                                              |
|--------------|--------------------------------------------------------------------------------------------------------------|
| <i>idx</i>   | Index into the parameter data array. A valid value is between 0 and (SHARED_PARAMS_CFG_BUFFER_DATA_LEN - 1). |
| <i>value</i> | Pointer to where the read data value is stored.                                                              |

**Returns**

True if successful, false otherwise.

**7.1162.2.4 SharedParamsValidateBuffer()**

```
static bool SharedParamsValidateBuffer (
 void) [static]
```

Validates the shared parameter buffer contents by looking at the table identifier and verifying its checksum.

**Returns**

True if successful, false otherwise.

Referenced by SharedParamsInit(), SharedParamsReadByIndex(), and SharedParamsWriteByIndex().

**7.1162.2.5 SharedParamsVerifyChecksum()**

```
static bool SharedParamsVerifyChecksum (
 void) [static]
```

Calculates and verifies the checksum that is currently present in the buffer.

**Returns**

True is the checksum is correct, false otherwise.

Referenced by SharedParamsValidateBuffer().

**7.1162.2.6 SharedParamsWriteByIndex()**

```
bool SharedParamsWriteByIndex (
 uint32_t idx,
 uint8_t value)
```

Writes a data byte to the shared parameter buffer at the specified index.

## Parameters

|              |                                                                                                              |
|--------------|--------------------------------------------------------------------------------------------------------------|
| <i>idx</i>   | Index into the parameter data array. A valid value is between 0 and (SHARED_PARAMS_CFG_BUFFER_DATA_LEN - 1). |
| <i>value</i> | Value to write.                                                                                              |

## Returns

True if successful, false otherwise.

## 7.1162.2.7 SharedParamsWriteChecksum()

```
static void SharedParamsWriteChecksum (
 void) [static]
```

Calculates and writes the checksum into the buffer.

## Returns

none.

Referenced by SharedParamsInit(), and SharedParamsWriteByIndex().

## 7.1162.3 Variable Documentation

## 7.1162.3.1 shared

```
__no_init tSharedParamsBuffer sharedParamsBuffer shared [static]
```

Declaration of the actual parameter buffer that this module manages.

## Warning

For the shared RAM parameters to work properly for sharing information between the bootloader and user program, it is important that this variable is linked to the exact same RAM address in both the bootloader and the user program. Additionally, it should be configured such that the C-startup code does NOT zero its contents during system initialization. This is the code that runs in the reset event handler, before function `main()` is called. For GCC based embedded toolchains, the solution is to assign this variable to a custom section, in this case called ".shared". Then in the linker script, add the following to the SECTIONS:

```
.shared (NOLOAD) : { . = ALIGN(4); _sshared = .; shared_start = _sshared; *(.shared) *(.shared.*) KE←
EP(*(.shared)) . = ALIGN(4); _eshared = .; shared_end = _eshared; } >SHARED
```

Next, add a new MEMORY entry for SHARED at the start of RAM and reduce the length of the remaining RAM:

```
SHARED (xrw) : ORIGIN = 0x200000C0, LENGTH = 64 RAM (xrw) : ORIGIN = 0x20000100, LENGTH = 32K - 192
- 64
```

Note that the previous example is for an STM32F0 microcontroller where the first 192 (0xC0) bytes in RAM are reserved for the user program vector table.

## Remarks

This same approach can be applied with other toolchains such as Keil MDK and IAR EWARM. Consult the compiler and linker user manuals of your toolchain to find out how to place a RAM variable at a fixed memory address and to prevent the C-startup code from zeroing its contents. Here are a few links to get you started:

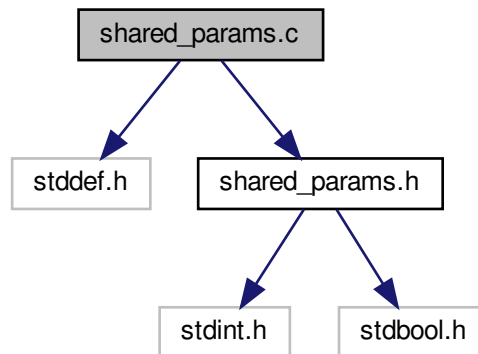
- IAR EWARM: <https://www.iar.com/support/tech-notes/compiler/linker-error-for-absolute-located-variable/>
- Keil MDK: <http://www.keil.com/support/docs/3480.htm>

## 7.1163 shared\_params.c File Reference

Shared RAM parameters source file.

```
#include <stddef.h>
#include "shared_params.h"
```

Include dependency graph for ARMCM7\_STM32F7\_Nucleo\_F767ZI\_CubeIDE/Boot/App/shared\_params.c:



### Data Structures

- struct [tSharedParamsBuffer](#)

*Layout of the shared parameters RAM buffer.*

### Macros

- `#define` [SHARED\\_PARAMS\\_BUFFER\\_ID](#) (0xCE42E7A2u)

*Constant parameter buffer identifier. This value is always located as the start of the buffer to validate the the RAM contains valid shared parameters.*

### Functions

- static [tSharedParamsBuffer](#) `sharedParamsBuffer` `__attribute__((section(".shared")))`  
*Declaration of the actual parameter buffer that this module manages.*
- static bool [SharedParamsValidateBuffer](#) (void)  
*Validates the shared parameter buffer contents by looking at the table identifier and verifying its checksum.*
- static void [SharedParamsWriteChecksum](#) (void)  
*Calculates and writes the checksum into the buffer.*
- static bool [SharedParamsVerifyChecksum](#) (void)  
*Calculates and verifies the checksum that is currently present in the buffer.*
- static uint16\_t [SharedParamsCalculateChecksum](#) (void)

*Calculates and returns the checksum value for the current contents in the buffer. The checksum is calculated by taking the sum of all bytes in the parameter buffer (excluding the checksum at the end) and then taking the two's complement value of it.*

- void [SharedParamsInit](#) (void)  
*Initializes the shared RAM parameters module.*
- bool [SharedParamsReadByIndex](#) (uint32\_t idx, uint8\_t \*value)  
*Reads a data byte from the shared parameter buffer at the specified index.*
- bool [SharedParamsWriteByIndex](#) (uint32\_t idx, uint8\_t value)  
*Writes a data byte to the shared parameter buffer at the specified index.*

### 7.1163.1 Detailed Description

Shared RAM parameters source file.

### 7.1163.2 Function Documentation

#### 7.1163.2.1 \_\_attribute\_\_()

```
static tSharedParamsBuffer sharedParamsBuffer __attribute__ (
 (section(".shared"))) [static]
```

Declaration of the actual parameter buffer that this module manages.

#### Warning

For the shared RAM parameters to work properly for sharing information between the bootloader and user program, it is important that this variable is linked to the exact same RAM address in both the bootloader and the user program. Additionally, it should be configured such that the C-startup code does NOT zero its contents during system initialization. This is the code that runs in the reset event handler, before function [main\(\)](#) is called. For GCC based embedded toolchains, the solution is to assign this variable to a custom section, in this case called ".shared". Then in the linker script, add the following to the SECTIONS:

```
.shared (NOLOAD) : { . = ALIGN(4); _sshared = .; shared_start = _sshared; *(.shared) *(.shared.*) KE↔
EP*(.shared)) . = ALIGN(4); _eshared = .; shared_end = _eshared; } >SHARED
```

Next, add a new MEMORY entry for SHARED at the start of RAM and reduce the length of the remaining RAM:

```
SHARED (xrw) : ORIGIN = 0x200000C0, LENGTH = 64 RAM (xrw) : ORIGIN = 0x20000100, LENGTH = 32K - 192
- 64
```

Note that the previous example is for an STM32F0 microcontroller where the first 192 (0xC0) bytes in RAM are reserved for the user program vector table.

#### Remarks

This same approach can be applied with other toolchains such as Keil MDK and IAR EWARM. Consult the compiler and linker user manuals of your toolchain to find out how to place a RAM variable at a fixed memory address and to prevent the C-startup code from zeroing its contents. Here are a few links to get you started:

- IAR EWARM: <https://www.iar.com/support/tech-notes/compiler/linker-error-for-absolute-located-variable/>
- Keil MDK: <http://www.keil.com/support/docs/3480.htm>

#### 7.1163.2.2 SharedParamsCalculateChecksum()

```
static uint16_t SharedParamsCalculateChecksum (
 void) [static]
```

Calculates and returns the checksum value for the current contents in the buffer. The checksum is calculated by taking the sum of all bytes in the parameter buffer (excluding the checksum at the end) and then taking the two's complement value of it.

##### Returns

The calculated checksum value.

Referenced by SharedParamsVerifyChecksum(), and SharedParamsWriteChecksum().

#### 7.1163.2.3 SharedParamsInit()

```
void SharedParamsInit (
 void)
```

Initializes the shared RAM parameters module.

##### Returns

none.

#### 7.1163.2.4 SharedParamsReadByIndex()

```
bool SharedParamsReadByIndex (
 uint32_t idx,
 uint8_t * value)
```

Reads a data byte from the shared parameter buffer at the specified index.

##### Parameters

|              |                                                                                                              |
|--------------|--------------------------------------------------------------------------------------------------------------|
| <i>idx</i>   | Index into the parameter data array. A valid value is between 0 and (SHARED_PARAMS_CFG_BUFFER_DATA_LEN - 1). |
| <i>value</i> | Pointer to where the read data value is stored.                                                              |

##### Returns

True if successful, false otherwise.

### 7.1163.2.5 SharedParamsValidateBuffer()

```
static bool SharedParamsValidateBuffer (
 void) [static]
```

Validates the shared parameter buffer contents by looking at the table identifier and verifying its checksum.

#### Returns

True if successful, false otherwise.

Referenced by SharedParamsInit(), SharedParamsReadByIndex(), and SharedParamsWriteByIndex().

### 7.1163.2.6 SharedParamsVerifyChecksum()

```
static bool SharedParamsVerifyChecksum (
 void) [static]
```

Calculates and verifies the checksum that is currently present in the buffer.

#### Returns

True is the checksum is correct, false otherwise.

Referenced by SharedParamsValidateBuffer().

### 7.1163.2.7 SharedParamsWriteByIndex()

```
bool SharedParamsWriteByIndex (
 uint32_t idx,
 uint8_t value)
```

Writes a data byte to the shared parameter buffer at the specified index.

#### Parameters

|              |                                                                                                              |
|--------------|--------------------------------------------------------------------------------------------------------------|
| <i>idx</i>   | Index into the parameter data array. A valid value is between 0 and (SHARED_PARAMS_CFG_BUFFER_DATA_LEN - 1). |
| <i>value</i> | Value to write.                                                                                              |

#### Returns

True if successful, false otherwise.

### 7.1163.2.8 SharedParamsWriteChecksum()

```
static void SharedParamsWriteChecksum (
 void) [static]
```

Calculates and writes the checksum into the buffer.

#### Returns

none.

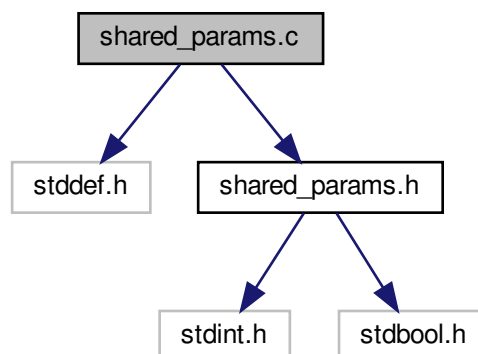
Referenced by SharedParamsInit(), and SharedParamsWriteByIndex().

## 7.1164 shared\_params.c File Reference

Shared RAM parameters source file.

```
#include <stddef.h>
#include "shared_params.h"
```

Include dependency graph for ARMCM7\_STM32F7\_Nucleo\_F767ZI\_CubeIDE/Prog/App/shared\_params.c:



## Data Structures

- struct [tSharedParamsBuffer](#)

*Layout of the shared parameters RAM buffer.*

## Macros

- #define [SHARED\\_PARAMS\\_BUFFER\\_ID](#) (0xCE42E7A2u)

*Constant parameter buffer identifier. This value is always located as the start of the buffer to validate the the RAM contains valid shared parameters.*



## Functions

- static `tSharedParamsBuffer` sharedParamsBuffer `__attribute__ ((section(".shared")))`  
*Declaration of the actual parameter buffer that this module manages.*
- static bool `SharedParamsValidateBuffer` (void)  
*Validates the shared parameter buffer contents by looking at the table identifier and verifying its checksum.*
- static void `SharedParamsWriteChecksum` (void)  
*Calculates and writes the checksum into the buffer.*
- static bool `SharedParamsVerifyChecksum` (void)  
*Calculates and verifies the checksum that is currently present in the buffer.*
- static uint16\_t `SharedParamsCalculateChecksum` (void)  
*Calculates and returns the checksum value for the current contents in the buffer. The checksum is calculated by taking the sum of all bytes in the parameter buffer (excluding the checksum at the end) and then taking the two's complement value of it.*
- void `SharedParamsInit` (void)  
*Initializes the shared RAM parameters module.*
- bool `SharedParamsReadByIndex` (uint32\_t idx, uint8\_t \*value)  
*Reads a data byte from the shared parameter buffer at the specified index.*
- bool `SharedParamsWriteByIndex` (uint32\_t idx, uint8\_t value)  
*Writes a data byte to the shared parameter buffer at the specified index.*

### 7.1164.1 Detailed Description

Shared RAM parameters source file.

### 7.1164.2 Function Documentation

#### 7.1164.2.1 `__attribute__()`

```
static tSharedParamsBuffer sharedParamsBuffer __attribute__ (
 (section(".shared"))) [static]
```

Declaration of the actual parameter buffer that this module manages.

#### Warning

For the shared RAM parameters to work properly for sharing information between the bootloader and user program, it is important that this variable is linked to the exact same RAM address in both the bootloader and the user program. Additionally, it should be configured such that the C-startup code does NOT zero its contents during system initialization. This is the code that runs in the reset event handler, before function `main()` is called. For GCC based embedded toolchains, the solution is to assign this variable to a custom section, in this case called ".shared". Then in the linker script, add the following to the SECTIONS:

```
.shared (NOLOAD) : { . = ALIGN(4); _sshared = .; shared_start = _sshared; *(.shared) *(.shared.*) KE←
EP(*(.shared)) . = ALIGN(4); _eshared = .; shared_end = _eshared; } >SHARED
```

Next, add a new MEMORY entry for SHARED at the start of RAM and reduce the length of the remaining RAM:

```
SHARED (xrw) : ORIGIN = 0x200000C0, LENGTH = 64 RAM (xrw) : ORIGIN = 0x20000100, LENGTH = 32K - 192
- 64
```

Note that the previous example is for an STM32F0 microcontroller where the first 192 (0xC0) bytes in RAM are reserved for the user program vector table.

## Remarks

This same approach can be applied with other toolchains such as Keil MDK and IAR EWARM. Consult the compiler and linker user manuals of your toolchain to find out how to place a RAM variable at a fixed memory address and to prevent the C-startup code from zeroing its contents. Here are a few links to get you started:

- IAR EWARM: <https://www.iar.com/support/tech-notes/compiler/linker-error-for-absolute-located-variable/>
- Keil MDK: <http://www.keil.com/support/docs/3480.htm>

### 7.1164.2.2 SharedParamsCalculateChecksum()

```
static uint16_t SharedParamsCalculateChecksum (
 void) [static]
```

Calculates and returns the checksum value for the current contents in the buffer. The checksum is calculated by taking the sum of all bytes in the parameter buffer (excluding the checksum at the end) and then taking the two's complement value of it.

## Returns

The calculated checksum value.

Referenced by SharedParamsVerifyChecksum(), and SharedParamsWriteChecksum().

### 7.1164.2.3 SharedParamsInit()

```
void SharedParamsInit (
 void)
```

Initializes the shared RAM parameters module.

## Returns

none.

### 7.1164.2.4 SharedParamsReadByIndex()

```
bool SharedParamsReadByIndex (
 uint32_t idx,
 uint8_t * value)
```

Reads a data byte from the shared parameter buffer at the specified index.

**Parameters**

|              |                                                                                                              |
|--------------|--------------------------------------------------------------------------------------------------------------|
| <i>idx</i>   | Index into the parameter data array. A valid value is between 0 and (SHARED_PARAMS_CFG_BUFFER_DATA_LEN - 1). |
| <i>value</i> | Pointer to where the read data value is stored.                                                              |

**Returns**

True if successful, false otherwise.

**7.1164.2.5 SharedParamsValidateBuffer()**

```
static bool SharedParamsValidateBuffer (
 void) [static]
```

Validates the shared parameter buffer contents by looking at the table identifier and verifying its checksum.

**Returns**

True if successful, false otherwise.

Referenced by SharedParamsInit(), SharedParamsReadByIndex(), and SharedParamsWriteByIndex().

**7.1164.2.6 SharedParamsVerifyChecksum()**

```
static bool SharedParamsVerifyChecksum (
 void) [static]
```

Calculates and verifies the checksum that is currently present in the buffer.

**Returns**

True is the checksum is correct, false otherwise.

Referenced by SharedParamsValidateBuffer().

**7.1164.2.7 SharedParamsWriteByIndex()**

```
bool SharedParamsWriteByIndex (
 uint32_t idx,
 uint8_t value)
```

Writes a data byte to the shared parameter buffer at the specified index.

**Parameters**

|              |                                                                                                              |
|--------------|--------------------------------------------------------------------------------------------------------------|
| <i>idx</i>   | Index into the parameter data array. A valid value is between 0 and (SHARED_PARAMS_CFG_BUFFER_DATA_LEN - 1). |
| <i>value</i> | Value to write.                                                                                              |

**Returns**

True if successful, false otherwise.

**7.1164.2.8 SharedParamsWriteChecksum()**

```
static void SharedParamsWriteChecksum (
 void) [static]
```

Calculates and writes the checksum into the buffer.

**Returns**

none.

Referenced by SharedParamsInit(), and SharedParamsWriteByIndex().

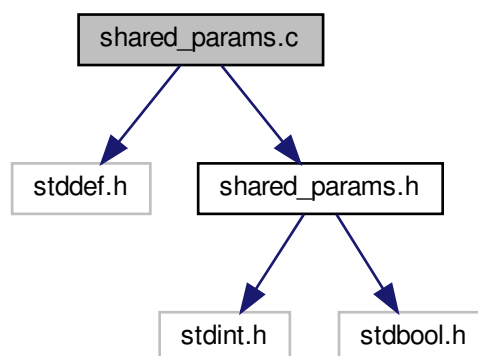
**7.1165 shared\_params.c File Reference**

Shared RAM parameters source file.

```
#include <stddef.h>
```

```
#include "shared_params.h"
```

Include dependency graph for ARMC77\_STM32F7\_Nucleo\_F767ZI\_GCC/Boot/shared\_params.c:



## Data Structures

- struct [tSharedParamsBuffer](#)

*Layout of the shared parameters RAM buffer.*

## Macros

- #define [SHARED\\_PARAMS\\_BUFFER\\_ID](#) (0xCE42E7A2u)

*Constant parameter buffer identifier. This value is always located as the start of the buffer to validate the the RAM contains valid shared parameters.*

## Functions

- static [tSharedParamsBuffer](#) sharedParamsBuffer [\\_\\_attribute\\_\\_](#) ((section(".shared")))

*Declaration of the actual parameter buffer that this module manages.*

- static bool [SharedParamsValidateBuffer](#) (void)

*Validates the shared parameter buffer contents by looking at the table identifier and verifying its checksum.*

- static void [SharedParamsWriteChecksum](#) (void)

*Calculates and writes the checksum into the buffer.*

- static bool [SharedParamsVerifyChecksum](#) (void)

*Calculates and verifies the checksum that is currently present in the buffer.*

- static uint16\_t [SharedParamsCalculateChecksum](#) (void)

*Calculates and returns the checksum value for the current contents in the buffer. The checksum is calculated by taking the sum of all bytes in the parameter buffer (excluding the checksum at the end) and then taking the two's complement value of it.*

- void [SharedParamsInit](#) (void)

*Initializes the shared RAM parameters module.*

- bool [SharedParamsReadByIndex](#) (uint32\_t idx, uint8\_t \*value)

*Reads a data byte from the shared parameter buffer at the specified index.*

- bool [SharedParamsWriteByIndex](#) (uint32\_t idx, uint8\_t value)

*Writes a data byte to the shared parameter buffer at the specified index.*

### 7.1165.1 Detailed Description

Shared RAM parameters source file.

### 7.1165.2 Function Documentation

### 7.1165.2.1 \_\_attribute\_\_((section(\".shared\")))

```
static tSharedParamsBuffer sharedParamsBuffer __attribute__((section(\".shared\"))) [static]
```

Declaration of the actual parameter buffer that this module manages.

#### Warning

For the shared RAM parameters to work properly for sharing information between the bootloader and user program, it is important that this variable is linked to the exact same RAM address in both the bootloader and the user program. Additionally, it should be configured such that the C-startup code does NOT zero its contents during system initialization. This is the code that runs in the reset event handler, before function `main()` is called. For GCC based embedded toolchains, the solution is to assign this variable to a custom section, in this case called `\".shared\"`. Then in the linker script, add the following to the SECTIONS:

```
\".shared (NOLOAD) : { . = ALIGN(4); _sshared = .; shared_start = _sshared; *(.shared) *(.shared.*) KE←
EP(*(.shared)) . = ALIGN(4); _eshared = .; shared_end = _eshared; } >SHARED
```

Next, add a new MEMORY entry for SHARED at the start of RAM and reduce the length of the remaining RAM:

```
SHARED (xrw) : ORIGIN = 0x200000C0, LENGTH = 64 RAM (xrw) : ORIGIN = 0x20000100, LENGTH = 32K - 192
- 64
```

Note that the previous example is for an STM32F0 microcontroller where the first 192 (0xC0) bytes in RAM are reserved for the user program vector table.

#### Remarks

This same approach can be applied with other toolchains such as Keil MDK and IAR EWARM. Consult the compiler and linker user manuals of your toolchain to find out how to place a RAM variable at a fixed memory address and to prevent the C-startup code from zeroing its contents. Here are a few links to get you started:

- IAR EWARM: <https://www.iar.com/support/tech-notes/compiler/linker-error-for-absolute-located-variable/>
- Keil MDK: <http://www.keil.com/support/docs/3480.htm>

### 7.1165.2.2 SharedParamsCalculateChecksum()

```
static uint16_t SharedParamsCalculateChecksum (
 void) [static]
```

Calculates and returns the checksum value for the current contents in the buffer. The checksum is calculated by taking the sum of all bytes in the parameter buffer (excluding the checksum at the end) and then taking the two's complement value of it.

#### Returns

The calculated checksum value.

Referenced by `SharedParamsVerifyChecksum()`, and `SharedParamsWriteChecksum()`.

### 7.1165.2.3 SharedParamsInit()

```
void SharedParamsInit (
 void)
```

Initializes the shared RAM parameters module.

#### Returns

none.

### 7.1165.2.4 SharedParamsReadByIndex()

```
bool SharedParamsReadByIndex (
 uint32_t idx,
 uint8_t * value)
```

Reads a data byte from the shared parameter buffer at the specified index.

#### Parameters

|              |                                                                                                              |
|--------------|--------------------------------------------------------------------------------------------------------------|
| <i>idx</i>   | Index into the parameter data array. A valid value is between 0 and (SHARED_PARAMS_CFG_BUFFER_DATA_LEN - 1). |
| <i>value</i> | Pointer to where the read data value is stored.                                                              |

#### Returns

True if successful, false otherwise.

### 7.1165.2.5 SharedParamsValidateBuffer()

```
static bool SharedParamsValidateBuffer (
 void) [static]
```

Validates the shared parameter buffer contents by looking at the table identifier and verifying its checksum.

#### Returns

True if successful, false otherwise.

Referenced by SharedParamsInit(), SharedParamsReadByIndex(), and SharedParamsWriteByIndex().

#### 7.1165.2.6 SharedParamsVerifyChecksum()

```
static bool SharedParamsVerifyChecksum (
 void) [static]
```

Calculates and verifies the checksum that is currently present in the buffer.

##### Returns

True is the checksum is correct, false otherwise.

Referenced by SharedParamsValidateBuffer().

#### 7.1165.2.7 SharedParamsWriteByIndex()

```
bool SharedParamsWriteByIndex (
 uint32_t idx,
 uint8_t value)
```

Writes a data byte to the shared parameter buffer at the specified index.

##### Parameters

|              |                                                                                                              |
|--------------|--------------------------------------------------------------------------------------------------------------|
| <i>idx</i>   | Index into the parameter data array. A valid value is between 0 and (SHARED_PARAMS_CFG_BUFFER_DATA_LEN - 1). |
| <i>value</i> | Value to write.                                                                                              |

##### Returns

True if successful, false otherwise.

#### 7.1165.2.8 SharedParamsWriteChecksum()

```
static void SharedParamsWriteChecksum (
 void) [static]
```

Calculates and writes the checksum into the buffer.

##### Returns

none.

Referenced by SharedParamsInit(), and SharedParamsWriteByIndex().

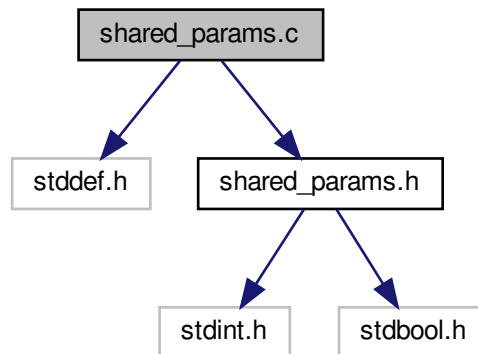


## 7.1166 shared\_params.c File Reference

Shared RAM parameters source file.

```
#include <stddef.h>
#include "shared_params.h"
```

Include dependency graph for ARMCM7\_STM32F7\_Nucleo\_F767ZI\_GCC/Prog/shared\_params.c:



### Data Structures

- struct [tSharedParamsBuffer](#)

*Layout of the shared parameters RAM buffer.*

### Macros

- `#define` [SHARED\\_PARAMS\\_BUFFER\\_ID](#) (0xCE42E7A2u)

*Constant parameter buffer identifier. This value is always located as the start of the buffer to validate the the RAM contains valid shared parameters.*

### Functions

- static [tSharedParamsBuffer](#) `sharedParamsBuffer` `__attribute__((section(".shared")))`  
*Declaration of the actual parameter buffer that this module manages.*
- static bool [SharedParamsValidateBuffer](#) (void)  
*Validates the shared parameter buffer contents by looking at the table identifier and verifying its checksum.*
- static void [SharedParamsWriteChecksum](#) (void)  
*Calculates and writes the checksum into the buffer.*
- static bool [SharedParamsVerifyChecksum](#) (void)  
*Calculates and verifies the checksum that is currently present in the buffer.*
- static uint16\_t [SharedParamsCalculateChecksum](#) (void)

*Calculates and returns the checksum value for the current contents in the buffer. The checksum is calculated by taking the sum of all bytes in the parameter buffer (excluding the checksum at the end) and then taking the two's complement value of it.*

- void [SharedParamsInit](#) (void)  
*Initializes the shared RAM parameters module.*
- bool [SharedParamsReadByIndex](#) (uint32\_t idx, uint8\_t \*value)  
*Reads a data byte from the shared parameter buffer at the specified index.*
- bool [SharedParamsWriteByIndex](#) (uint32\_t idx, uint8\_t value)  
*Writes a data byte to the shared parameter buffer at the specified index.*

### 7.1166.1 Detailed Description

Shared RAM parameters source file.

### 7.1166.2 Function Documentation

#### 7.1166.2.1 `__attribute__()`

```
static tSharedParamsBuffer sharedParamsBuffer __attribute__ (
 (section(".shared"))) [static]
```

Declaration of the actual parameter buffer that this module manages.

#### Warning

For the shared RAM parameters to work properly for sharing information between the bootloader and user program, it is important that this variable is linked to the exact same RAM address in both the bootloader and the user program. Additionally, it should be configured such that the C-startup code does NOT zero its contents during system initialization. This is the code that runs in the reset event handler, before function [main\(\)](#) is called. For GCC based embedded toolchains, the solution is to assign this variable to a custom section, in this case called ".shared". Then in the linker script, add the following to the SECTIONS:

```
.shared (NOLOAD) : { . = ALIGN(4); _sshared = .; shared_start = _sshared; *(.shared) *(.shared.*) KE↔
EP(*(.shared)) . = ALIGN(4); _eshared = .; shared_end = _eshared; } >SHARED
```

Next, add a new MEMORY entry for SHARED at the start of RAM and reduce the length of the remaining RAM:

```
SHARED (xrw) : ORIGIN = 0x200000C0, LENGTH = 64 RAM (xrw) : ORIGIN = 0x20000100, LENGTH = 32K - 192
- 64
```

Note that the previous example is for an STM32F0 microcontroller where the first 192 (0xC0) bytes in RAM are reserved for the user program vector table.

#### Remarks

This same approach can be applied with other toolchains such as Keil MDK and IAR EWARM. Consult the compiler and linker user manuals of your toolchain to find out how to place a RAM variable at a fixed memory address and to prevent the C-startup code from zeroing its contents. Here are a few links to get you started:

- IAR EWARM: <https://www.iar.com/support/tech-notes/compiler/linker-error-for-absolute-located-variable/>
- Keil MDK: <http://www.keil.com/support/docs/3480.htm>

### 7.1166.2.2 SharedParamsCalculateChecksum()

```
static uint16_t SharedParamsCalculateChecksum (
 void) [static]
```

Calculates and returns the checksum value for the current contents in the buffer. The checksum is calculated by taking the sum of all bytes in the parameter buffer (excluding the checksum at the end) and then taking the two's complement value of it.

#### Returns

The calculated checksum value.

Referenced by SharedParamsVerifyChecksum(), and SharedParamsWriteChecksum().

### 7.1166.2.3 SharedParamsInit()

```
void SharedParamsInit (
 void)
```

Initializes the shared RAM parameters module.

#### Returns

none.

### 7.1166.2.4 SharedParamsReadByIndex()

```
bool SharedParamsReadByIndex (
 uint32_t idx,
 uint8_t * value)
```

Reads a data byte from the shared parameter buffer at the specified index.

#### Parameters

|              |                                                                                                              |
|--------------|--------------------------------------------------------------------------------------------------------------|
| <i>idx</i>   | Index into the parameter data array. A valid value is between 0 and (SHARED_PARAMS_CFG_BUFFER_DATA_LEN - 1). |
| <i>value</i> | Pointer to where the read data value is stored.                                                              |

#### Returns

True if successful, false otherwise.

#### 7.1166.2.5 SharedParamsValidateBuffer()

```
static bool SharedParamsValidateBuffer (
 void) [static]
```

Validates the shared parameter buffer contents by looking at the table identifier and verifying its checksum.

##### Returns

True if successful, false otherwise.

Referenced by SharedParamsInit(), SharedParamsReadByIndex(), and SharedParamsWriteByIndex().

#### 7.1166.2.6 SharedParamsVerifyChecksum()

```
static bool SharedParamsVerifyChecksum (
 void) [static]
```

Calculates and verifies the checksum that is currently present in the buffer.

##### Returns

True is the checksum is correct, false otherwise.

Referenced by SharedParamsValidateBuffer().

#### 7.1166.2.7 SharedParamsWriteByIndex()

```
bool SharedParamsWriteByIndex (
 uint32_t idx,
 uint8_t value)
```

Writes a data byte to the shared parameter buffer at the specified index.

##### Parameters

|              |                                                                                                              |
|--------------|--------------------------------------------------------------------------------------------------------------|
| <i>idx</i>   | Index into the parameter data array. A valid value is between 0 and (SHARED_PARAMS_CFG_BUFFER_DATA_LEN - 1). |
| <i>value</i> | Value to write.                                                                                              |

##### Returns

True if successful, false otherwise.

### 7.1166.2.8 SharedParamsWriteChecksum()

```
static void SharedParamsWriteChecksum (
 void) [static]
```

Calculates and writes the checksum into the buffer.

#### Returns

none.

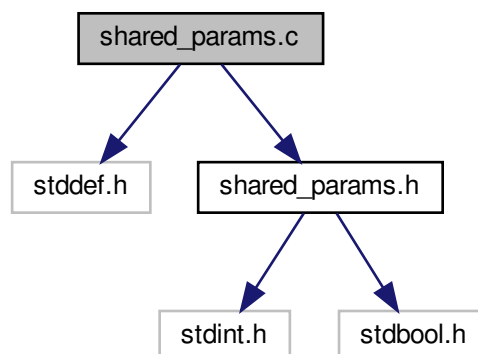
Referenced by SharedParamsInit(), and SharedParamsWriteByIndex().

## 7.1167 shared\_params.c File Reference

Shared RAM parameters source file.

```
#include <stddef.h>
#include "shared_params.h"
```

Include dependency graph for ARMCM7\_STM32F7\_Nucleo\_F767ZI\_IAR/Boot/shared\_params.c:



## Data Structures

- struct [tSharedParamsBuffer](#)

*Layout of the shared parameters RAM buffer.*

## Macros

- #define [SHARED\\_PARAMS\\_BUFFER\\_ID](#) (0xCE42E7A2u)

*Constant parameter buffer identifier. This value is always located as the start of the buffer to validate the the RAM contains valid shared parameters.*

## Functions

- static bool [SharedParamsValidateBuffer](#) (void)  
*Validates the shared parameter buffer contents by looking at the table identifier and verifying its checksum.*
- static void [SharedParamsWriteChecksum](#) (void)  
*Calculates and writes the checksum into the buffer.*
- static bool [SharedParamsVerifyChecksum](#) (void)  
*Calculates and verifies the checksum that is currently present in the buffer.*
- static uint16\_t [SharedParamsCalculateChecksum](#) (void)  
*Calculates and returns the checksum value for the current contents in the buffer. The checksum is calculated by taking the sum of all bytes in the parameter buffer (excluding the checksum at the end) and then taking the two's complement value of it.*
- void [SharedParamsInit](#) (void)  
*Initializes the shared RAM parameters module.*
- bool [SharedParamsReadByIndex](#) (uint32\_t idx, uint8\_t \*value)  
*Reads a data byte from the shared parameter buffer at the specified index.*
- bool [SharedParamsWriteByIndex](#) (uint32\_t idx, uint8\_t value)  
*Writes a data byte to the shared parameter buffer at the specified index.*

## Variables

- static \_\_no\_init [tSharedParamsBuffer](#) sharedParamsBuffer [shared](#)  
*Declaration of the actual parameter buffer that this module manages.*

### 7.1167.1 Detailed Description

Shared RAM parameters source file.

### 7.1167.2 Function Documentation

#### 7.1167.2.1 SharedParamsCalculateChecksum()

```
static uint16_t SharedParamsCalculateChecksum (
 void) [static]
```

Calculates and returns the checksum value for the current contents in the buffer. The checksum is calculated by taking the sum of all bytes in the parameter buffer (excluding the checksum at the end) and then taking the two's complement value of it.

#### Returns

The calculated checksum value.

Referenced by [SharedParamsVerifyChecksum\(\)](#), and [SharedParamsWriteChecksum\(\)](#).

### 7.1167.2.2 SharedParamsInit()

```
void SharedParamsInit (
 void)
```

Initializes the shared RAM parameters module.

#### Returns

none.

### 7.1167.2.3 SharedParamsReadByIndex()

```
bool SharedParamsReadByIndex (
 uint32_t idx,
 uint8_t * value)
```

Reads a data byte from the shared parameter buffer at the specified index.

#### Parameters

|              |                                                                                                              |
|--------------|--------------------------------------------------------------------------------------------------------------|
| <i>idx</i>   | Index into the parameter data array. A valid value is between 0 and (SHARED_PARAMS_CFG_BUFFER_DATA_LEN - 1). |
| <i>value</i> | Pointer to where the read data value is stored.                                                              |

#### Returns

True if successful, false otherwise.

### 7.1167.2.4 SharedParamsValidateBuffer()

```
static bool SharedParamsValidateBuffer (
 void) [static]
```

Validates the shared parameter buffer contents by looking at the table identifier and verifying its checksum.

#### Returns

True if successful, false otherwise.

Referenced by SharedParamsInit(), SharedParamsReadByIndex(), and SharedParamsWriteByIndex().

### 7.1167.2.5 SharedParamsVerifyChecksum()

```
static bool SharedParamsVerifyChecksum (
 void) [static]
```

Calculates and verifies the checksum that is currently present in the buffer.

#### Returns

True is the checksum is correct, false otherwise.

Referenced by SharedParamsValidateBuffer().

### 7.1167.2.6 SharedParamsWriteByIndex()

```
bool SharedParamsWriteByIndex (
 uint32_t idx,
 uint8_t value)
```

Writes a data byte to the shared parameter buffer at the specified index.

#### Parameters

|              |                                                                                                              |
|--------------|--------------------------------------------------------------------------------------------------------------|
| <i>idx</i>   | Index into the parameter data array. A valid value is between 0 and (SHARED_PARAMS_CFG_BUFFER_DATA_LEN - 1). |
| <i>value</i> | Value to write.                                                                                              |

#### Returns

True if successful, false otherwise.

### 7.1167.2.7 SharedParamsWriteChecksum()

```
static void SharedParamsWriteChecksum (
 void) [static]
```

Calculates and writes the checksum into the buffer.

#### Returns

none.

Referenced by SharedParamsInit(), and SharedParamsWriteByIndex().



### 7.1167.3 Variable Documentation

#### 7.1167.3.1 shared

```
__no_init tSharedParamsBuffer sharedParamsBuffer shared [static]
```

Declaration of the actual parameter buffer that this module manages.

#### Warning

For the shared RAM parameters to work properly for sharing information between the bootloader and user program, it is important that this variable is linked to the exact same RAM address in both the bootloader and the user program. Additionally, it should be configured such that the C-startup code does NOT zero its contents during system initialization. This is the code that runs in the reset event handler, before function `main()` is called. For GCC based embedded toolchains, the solution is to assign this variable to a custom section, in this case called ".shared". Then in the linker script, add the following to the SECTIONS:

```
.shared (NOLOAD) : { . = ALIGN(4); _sshared = .; shared_start = _sshared; *(.shared) *(.shared.*) KE←
EP(*(.(shared))) . = ALIGN(4); _eshared = .; shared_end = _eshared; } >SHARED
```

Next, add a new MEMORY entry for SHARED at the start of RAM and reduce the length of the remaining RAM:

```
SHARED (xrw) : ORIGIN = 0x200000C0, LENGTH = 64 RAM (xrw) : ORIGIN = 0x20000100, LENGTH = 32K - 192
- 64
```

Note that the previous example is for an STM32F0 microcontroller where the first 192 (0xC0) bytes in RAM are reserved for the user program vector table.

#### Remarks

This same approach can be applied with other toolchains such as Keil MDK and IAR EWARM. Consult the compiler and linker user manuals of your toolchain to find out how to place a RAM variable at a fixed memory address and to prevent the C-startup code from zeroing its contents. Here are a few links to get you started:

- IAR EWARM: <https://www.iar.com/support/tech-notes/compiler/linker-error-for-absolute-located-variable/>
- Keil MDK: <http://www.keil.com/support/docs/3480.htm>

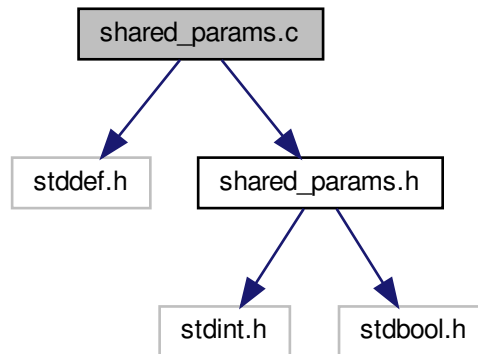
## 7.1168 shared\_params.c File Reference

Shared RAM parameters source file.

```
#include <stddef.h>
```

```
#include "shared_params.h"
```

Include dependency graph for ARMCM7\_STM32F7\_Nucleo\_F767ZI\_IAR/Prog/shared\_params.c:



## Data Structures

- struct [tSharedParamsBuffer](#)

*Layout of the shared parameters RAM buffer.*

## Macros

- `#define` [SHARED\\_PARAMS\\_BUFFER\\_ID](#) (0xCE42E7A2u)

*Constant parameter buffer identifier. This value is always located as the start of the buffer to validate the the RAM contains valid shared parameters.*

## Functions

- static bool [SharedParamsValidateBuffer](#) (void)  
*Validates the shared parameter buffer contents by looking at the table identifier and verifying its checksum.*
- static void [SharedParamsWriteChecksum](#) (void)  
*Calculates and writes the checksum into the buffer.*
- static bool [SharedParamsVerifyChecksum](#) (void)  
*Calculates and verifies the checksum that is currently present in the buffer.*
- static uint16\_t [SharedParamsCalculateChecksum](#) (void)  
*Calculates and returns the checksum value for the current contents in the buffer. The checksum is calculated by taking the sum of all bytes in the parameter buffer (excluding the checksum at the end) and then taking the two's complement value of it.*
- void [SharedParamsInit](#) (void)  
*Initializes the shared RAM parameters module.*
- bool [SharedParamsReadByIndex](#) (uint32\_t idx, uint8\_t \*value)  
*Reads a data byte from the shared parameter buffer at the specified index.*
- bool [SharedParamsWriteByIndex](#) (uint32\_t idx, uint8\_t value)  
*Writes a data byte to the shared parameter buffer at the specified index.*

## Variables

- static \_\_no\_init [tSharedParamsBuffer](#) sharedParamsBuffer [shared](#)  
*Declaration of the actual parameter buffer that this module manages.*

### 7.1168.1 Detailed Description

Shared RAM parameters source file.

### 7.1168.2 Function Documentation

#### 7.1168.2.1 SharedParamsCalculateChecksum()

```
static uint16_t SharedParamsCalculateChecksum (
 void) [static]
```

Calculates and returns the checksum value for the current contents in the buffer. The checksum is calculated by taking the sum of all bytes in the parameter buffer (excluding the checksum at the end) and then taking the two's complement value of it.

#### Returns

The calculated checksum value.

Referenced by SharedParamsVerifyChecksum(), and SharedParamsWriteChecksum().

#### 7.1168.2.2 SharedParamsInit()

```
void SharedParamsInit (
 void)
```

Initializes the shared RAM parameters module.

#### Returns

none.

#### 7.1168.2.3 SharedParamsReadByIndex()

```
bool SharedParamsReadByIndex (
 uint32_t idx,
 uint8_t * value)
```

Reads a data byte from the shared parameter buffer at the specified index.

**Parameters**

|              |                                                                                                              |
|--------------|--------------------------------------------------------------------------------------------------------------|
| <i>idx</i>   | Index into the parameter data array. A valid value is between 0 and (SHARED_PARAMS_CFG_BUFFER_DATA_LEN - 1). |
| <i>value</i> | Pointer to where the read data value is stored.                                                              |

**Returns**

True if successful, false otherwise.

**7.1168.2.4 SharedParamsValidateBuffer()**

```
static bool SharedParamsValidateBuffer (
 void) [static]
```

Validates the shared parameter buffer contents by looking at the table identifier and verifying its checksum.

**Returns**

True if successful, false otherwise.

Referenced by SharedParamsInit(), SharedParamsReadByIndex(), and SharedParamsWriteByIndex().

**7.1168.2.5 SharedParamsVerifyChecksum()**

```
static bool SharedParamsVerifyChecksum (
 void) [static]
```

Calculates and verifies the checksum that is currently present in the buffer.

**Returns**

True is the checksum is correct, false otherwise.

Referenced by SharedParamsValidateBuffer().

**7.1168.2.6 SharedParamsWriteByIndex()**

```
bool SharedParamsWriteByIndex (
 uint32_t idx,
 uint8_t value)
```

Writes a data byte to the shared parameter buffer at the specified index.

## Parameters

|              |                                                                                                              |
|--------------|--------------------------------------------------------------------------------------------------------------|
| <i>idx</i>   | Index into the parameter data array. A valid value is between 0 and (SHARED_PARAMS_CFG_BUFFER_DATA_LEN - 1). |
| <i>value</i> | Value to write.                                                                                              |

## Returns

True if successful, false otherwise.

## 7.1168.2.7 SharedParamsWriteChecksum()

```
static void SharedParamsWriteChecksum (
 void) [static]
```

Calculates and writes the checksum into the buffer.

## Returns

none.

Referenced by SharedParamsInit(), and SharedParamsWriteByIndex().

## 7.1168.3 Variable Documentation

## 7.1168.3.1 shared

```
__no_init tSharedParamsBuffer sharedParamsBuffer shared [static]
```

Declaration of the actual parameter buffer that this module manages.

## Warning

For the shared RAM parameters to work properly for sharing information between the bootloader and user program, it is important that this variable is linked to the exact same RAM address in both the bootloader and the user program. Additionally, it should be configured such that the C-startup code does NOT zero its contents during system initialization. This is the code that runs in the reset event handler, before function `main()` is called. For GCC based embedded toolchains, the solution is to assign this variable to a custom section, in this case called ".shared". Then in the linker script, add the following to the SECTIONS:

```
.shared (NOLOAD) : { . = ALIGN(4); _sshared = .; shared_start = _sshared; *(.shared) *(.shared.*) KE←
EP(*(.shared)) . = ALIGN(4); _eshared = .; shared_end = _eshared; } >SHARED
```

Next, add a new MEMORY entry for SHARED at the start of RAM and reduce the length of the remaining RAM:

```
SHARED (xrw) : ORIGIN = 0x200000C0, LENGTH = 64 RAM (xrw) : ORIGIN = 0x20000100, LENGTH = 32K - 192
- 64
```

Note that the previous example is for an STM32F0 microcontroller where the first 192 (0xC0) bytes in RAM are reserved for the user program vector table.

## Remarks

This same approach can be applied with other toolchains such as Keil MDK and IAR EWARM. Consult the compiler and linker user manuals of your toolchain to find out how to place a RAM variable at a fixed memory address and to prevent the C-startup code from zeroing its contents. Here are a few links to get you started:

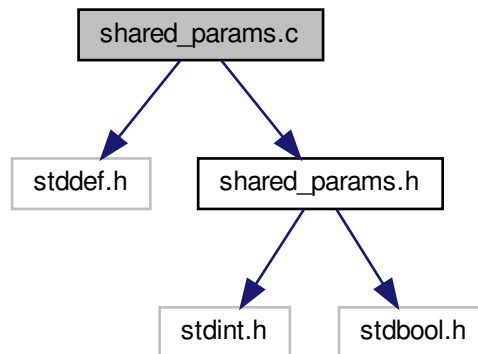
- IAR EWARM: <https://www.iar.com/support/tech-notes/compiler/linker-error-for-absolute-located-variable/>
- Keil MDK: <http://www.keil.com/support/docs/3480.htm>

## 7.1169 shared\_params.c File Reference

Shared RAM parameters source file.

```
#include <stddef.h>
#include "shared_params.h"
```

Include dependency graph for ARMCM7\_STM32F7\_Nucleo\_F767ZI\_Keil/Boot/shared\_params.c:



### Data Structures

- struct [tSharedParamsBuffer](#)

*Layout of the shared parameters RAM buffer.*

### Macros

- `#define` [SHARED\\_PARAMS\\_BUFFER\\_ID](#) (0xCE42E7A2u)

*Constant parameter buffer identifier. This value is always located as the start of the buffer to validate the the RAM contains valid shared parameters.*

### Functions

- static [tSharedParamsBuffer](#) `sharedParamsBuffer` `__attribute__((section("shared"), zero_init))`  
*Declaration of the actual parameter buffer that this module manages.*
- static bool [SharedParamsValidateBuffer](#) (void)  
*Validates the shared parameter buffer contents by looking at the table identifier and verifying its checksum.*
- static void [SharedParamsWriteChecksum](#) (void)  
*Calculates and writes the checksum into the buffer.*
- static bool [SharedParamsVerifyChecksum](#) (void)  
*Calculates and verifies the checksum that is currently present in the buffer.*
- static uint16\_t [SharedParamsCalculateChecksum](#) (void)

*Calculates and returns the checksum value for the current contents in the buffer. The checksum is calculated by taking the sum of all bytes in the parameter buffer (excluding the checksum at the end) and then taking the two's complement value of it.*

- void [SharedParamsInit](#) (void)  
*Initializes the shared RAM parameters module.*
- bool [SharedParamsReadByIndex](#) (uint32\_t idx, uint8\_t \*value)  
*Reads a data byte from the shared parameter buffer at the specified index.*
- bool [SharedParamsWriteByIndex](#) (uint32\_t idx, uint8\_t value)  
*Writes a data byte to the shared parameter buffer at the specified index.*

### 7.1169.1 Detailed Description

Shared RAM parameters source file.

### 7.1169.2 Function Documentation

#### 7.1169.2.1 \_\_attribute\_\_()

```
static tSharedParamsBuffer sharedParamsBuffer __attribute__ (
 (section("shared"), zero_init)) [static]
```

Declaration of the actual parameter buffer that this module manages.

#### Warning

For the shared RAM parameters to work properly for sharing information between the bootloader and user program, it is important that this variable is linked to the exact same RAM address in both the bootloader and the user program. Additionally, it should be configured such that the C-startup code does NOT zero its contents during system initialization. This is the code that runs in the reset event handler, before function [main\(\)](#) is called. For GCC based embedded toolchains, the solution is to assign this variable to a custom section, in this case called ".shared". Then in the linker script, add the following to the SECTIONS:

```
.shared (NOLOAD) : { . = ALIGN(4); _sshared = .; shared_start = _sshared; *(.shared) *(.shared.*) KE↔
EP(*(.shared)) . = ALIGN(4); _eshared = .; shared_end = _eshared; } >SHARED
```

Next, add a new MEMORY entry for SHARED at the start of RAM and reduce the length of the remaining RAM:

```
SHARED (xrw) : ORIGIN = 0x200000C0, LENGTH = 64 RAM (xrw) : ORIGIN = 0x20000100, LENGTH = 32K - 192
- 64
```

Note that the previous example is for an STM32F0 microcontroller where the first 192 (0xC0) bytes in RAM are reserved for the user program vector table.

#### Remarks

This same approach can be applied with other toolchains such as Keil MDK and IAR EWARM. Consult the compiler and linker user manuals of your toolchain to find out how to place a RAM variable at a fixed memory address and to prevent the C-startup code from zeroing its contents. Here are a few links to get you started:

- IAR EWARM: <https://www.iar.com/support/tech-notes/compiler/linker-error-for-absolute-located-variable/>
- Keil MDK: <http://www.keil.com/support/docs/3480.htm>

#### 7.1169.2.2 SharedParamsCalculateChecksum()

```
static uint16_t SharedParamsCalculateChecksum (
 void) [static]
```

Calculates and returns the checksum value for the current contents in the buffer. The checksum is calculated by taking the sum of all bytes in the parameter buffer (excluding the checksum at the end) and then taking the two's complement value of it.

##### Returns

The calculated checksum value.

Referenced by SharedParamsVerifyChecksum(), and SharedParamsWriteChecksum().

#### 7.1169.2.3 SharedParamsInit()

```
void SharedParamsInit (
 void)
```

Initializes the shared RAM parameters module.

##### Returns

none.

#### 7.1169.2.4 SharedParamsReadByIndex()

```
bool SharedParamsReadByIndex (
 uint32_t idx,
 uint8_t * value)
```

Reads a data byte from the shared parameter buffer at the specified index.

##### Parameters

|              |                                                                                                              |
|--------------|--------------------------------------------------------------------------------------------------------------|
| <i>idx</i>   | Index into the parameter data array. A valid value is between 0 and (SHARED_PARAMS_CFG_BUFFER_DATA_LEN - 1). |
| <i>value</i> | Pointer to where the read data value is stored.                                                              |

##### Returns

True if successful, false otherwise.



### 7.1169.2.5 SharedParamsValidateBuffer()

```
static bool SharedParamsValidateBuffer (
 void) [static]
```

Validates the shared parameter buffer contents by looking at the table identifier and verifying its checksum.

#### Returns

True if successful, false otherwise.

Referenced by SharedParamsInit(), SharedParamsReadByIndex(), and SharedParamsWriteByIndex().

### 7.1169.2.6 SharedParamsVerifyChecksum()

```
static bool SharedParamsVerifyChecksum (
 void) [static]
```

Calculates and verifies the checksum that is currently present in the buffer.

#### Returns

True is the checksum is correct, false otherwise.

Referenced by SharedParamsValidateBuffer().

### 7.1169.2.7 SharedParamsWriteByIndex()

```
bool SharedParamsWriteByIndex (
 uint32_t idx,
 uint8_t value)
```

Writes a data byte to the shared parameter buffer at the specified index.

#### Parameters

|              |                                                                                                              |
|--------------|--------------------------------------------------------------------------------------------------------------|
| <i>idx</i>   | Index into the parameter data array. A valid value is between 0 and (SHARED_PARAMS_CFG_BUFFER_DATA_LEN - 1). |
| <i>value</i> | Value to write.                                                                                              |

#### Returns

True if successful, false otherwise.

### 7.1169.2.8 SharedParamsWriteChecksum()

```
static void SharedParamsWriteChecksum (
 void) [static]
```

Calculates and writes the checksum into the buffer.

#### Returns

none.

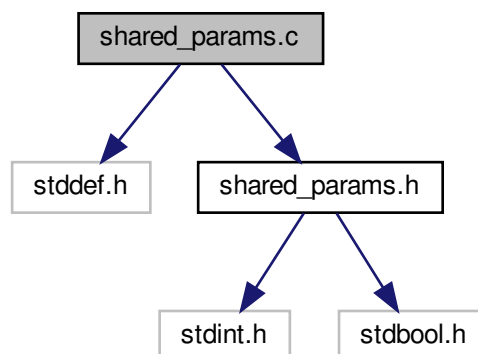
Referenced by SharedParamsInit(), and SharedParamsWriteByIndex().

## 7.1170 shared\_params.c File Reference

Shared RAM parameters source file.

```
#include <stddef.h>
#include "shared_params.h"
```

Include dependency graph for ARMCM7\_STM32F7\_Nucleo\_F767ZI\_Keil/Prog/shared\_params.c:



## Data Structures

- struct [tSharedParamsBuffer](#)

*Layout of the shared parameters RAM buffer.*

## Macros

- #define [SHARED\\_PARAMS\\_BUFFER\\_ID](#) (0xCE42E7A2u)

*Constant parameter buffer identifier. This value is always located as the start of the buffer to validate the the RAM contains valid shared parameters.*

## Functions

- static `tSharedParamsBuffer` sharedParamsBuffer `__attribute__` ((section("shared"), zero\_init))  
*Declaration of the actual parameter buffer that this module manages.*
- static bool `SharedParamsValidateBuffer` (void)  
*Validates the shared parameter buffer contents by looking at the table identifier and verifying its checksum.*
- static void `SharedParamsWriteChecksum` (void)  
*Calculates and writes the checksum into the buffer.*
- static bool `SharedParamsVerifyChecksum` (void)  
*Calculates and verifies the checksum that is currently present in the buffer.*
- static uint16\_t `SharedParamsCalculateChecksum` (void)  
*Calculates and returns the checksum value for the current contents in the buffer. The checksum is calculated by taking the sum of all bytes in the parameter buffer (excluding the checksum at the end) and then taking the two's complement value of it.*
- void `SharedParamsInit` (void)  
*Initializes the shared RAM parameters module.*
- bool `SharedParamsReadByIndex` (uint32\_t idx, uint8\_t \*value)  
*Reads a data byte from the shared parameter buffer at the specified index.*
- bool `SharedParamsWriteByIndex` (uint32\_t idx, uint8\_t value)  
*Writes a data byte to the shared parameter buffer at the specified index.*

### 7.1170.1 Detailed Description

Shared RAM parameters source file.

### 7.1170.2 Function Documentation

#### 7.1170.2.1 `__attribute__`()

```
static tSharedParamsBuffer sharedParamsBuffer __attribute__ (
 (section("shared"), zero_init)) [static]
```

Declaration of the actual parameter buffer that this module manages.

#### Warning

For the shared RAM parameters to work properly for sharing information between the bootloader and user program, it is important that this variable is linked to the exact same RAM address in both the bootloader and the user program. Additionally, it should be configured such that the C-startup code does NOT zero its contents during system initialization. This is the code that runs in the reset event handler, before function `main()` is called. For GCC based embedded toolchains, the solution is to assign this variable to a custom section, in this case called ".shared". Then in the linker script, add the following to the SECTIONS:

```
.shared (NOLOAD) : { . = ALIGN(4); _sshared = .; shared_start = _sshared; *(.shared) *(.shared.*) KE←
EP(*(.shared)) . = ALIGN(4); _eshared = .; shared_end = _eshared; } >SHARED
```

Next, add a new MEMORY entry for SHARED at the start of RAM and reduce the length of the remaining RAM:

```
SHARED (xrw) : ORIGIN = 0x200000C0, LENGTH = 64 RAM (xrw) : ORIGIN = 0x20000100, LENGTH = 32K - 192
- 64
```

Note that the previous example is for an STM32F0 microcontroller where the first 192 (0xC0) bytes in RAM are reserved for the user program vector table.

## Remarks

This same approach can be applied with other toolchains such as Keil MDK and IAR EWARM. Consult the compiler and linker user manuals of your toolchain to find out how to place a RAM variable at a fixed memory address and to prevent the C-startup code from zeroing its contents. Here are a few links to get you started:

- IAR EWARM: <https://www.iar.com/support/tech-notes/compiler/linker-error-for-absolute-located-variable/>
- Keil MDK: <http://www.keil.com/support/docs/3480.htm>

### 7.1170.2.2 SharedParamsCalculateChecksum()

```
static uint16_t SharedParamsCalculateChecksum (
 void) [static]
```

Calculates and returns the checksum value for the current contents in the buffer. The checksum is calculated by taking the sum of all bytes in the parameter buffer (excluding the checksum at the end) and then taking the two's complement value of it.

## Returns

The calculated checksum value.

Referenced by SharedParamsVerifyChecksum(), and SharedParamsWriteChecksum().

### 7.1170.2.3 SharedParamsInit()

```
void SharedParamsInit (
 void)
```

Initializes the shared RAM parameters module.

## Returns

none.

### 7.1170.2.4 SharedParamsReadByIndex()

```
bool SharedParamsReadByIndex (
 uint32_t idx,
 uint8_t * value)
```

Reads a data byte from the shared parameter buffer at the specified index.

**Parameters**

|              |                                                                                                              |
|--------------|--------------------------------------------------------------------------------------------------------------|
| <i>idx</i>   | Index into the parameter data array. A valid value is between 0 and (SHARED_PARAMS_CFG_BUFFER_DATA_LEN - 1). |
| <i>value</i> | Pointer to where the read data value is stored.                                                              |

**Returns**

True if successful, false otherwise.

**7.1170.2.5 SharedParamsValidateBuffer()**

```
static bool SharedParamsValidateBuffer (
 void) [static]
```

Validates the shared parameter buffer contents by looking at the table identifier and verifying its checksum.

**Returns**

True if successful, false otherwise.

Referenced by SharedParamsInit(), SharedParamsReadByIndex(), and SharedParamsWriteByIndex().

**7.1170.2.6 SharedParamsVerifyChecksum()**

```
static bool SharedParamsVerifyChecksum (
 void) [static]
```

Calculates and verifies the checksum that is currently present in the buffer.

**Returns**

True is the checksum is correct, false otherwise.

Referenced by SharedParamsValidateBuffer().

**7.1170.2.7 SharedParamsWriteByIndex()**

```
bool SharedParamsWriteByIndex (
 uint32_t idx,
 uint8_t value)
```

Writes a data byte to the shared parameter buffer at the specified index.

**Parameters**

|              |                                                                                                              |
|--------------|--------------------------------------------------------------------------------------------------------------|
| <i>idx</i>   | Index into the parameter data array. A valid value is between 0 and (SHARED_PARAMS_CFG_BUFFER_DATA_LEN - 1). |
| <i>value</i> | Value to write.                                                                                              |

**Returns**

True if successful, false otherwise.

**7.1170.2.8 SharedParamsWriteChecksum()**

```
static void SharedParamsWriteChecksum (
 void) [static]
```

Calculates and writes the checksum into the buffer.

**Returns**

none.

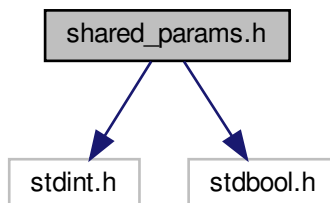
Referenced by SharedParamsInit(), and SharedParamsWriteByIndex().

**7.1171 shared\_params.h File Reference**

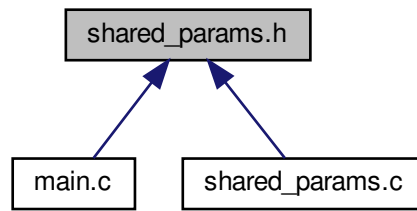
Shared RAM parameters header file.

```
#include <stdint.h>
#include <stdbool.h>
```

Include dependency graph for ARMCM3\_LM3S\_EK\_LM3S6965\_GCC/Boot/shared\_params.h:



This graph shows which files directly or indirectly include this file:



## Macros

- `#define SHARED_PARAMS_CFG_BUFFER_DATA_LEN (56u)`  
*Configuration macro for specifying the size of the data inside the parameter buffer. This is the length in bytes of the actual parameter data, so excluding the bufferId and checksum.*

## Functions

- `void SharedParamsInit (void)`  
*Initializes the shared RAM parameters module.*
- `bool SharedParamsReadByIndex (uint32_t idx, uint8_t *value)`  
*Reads a data byte from the shared parameter buffer at the specified index.*
- `bool SharedParamsWriteByIndex (uint32_t idx, uint8_t value)`  
*Writes a data byte to the shared parameter buffer at the specified index.*

### 7.1171.1 Detailed Description

Shared RAM parameters header file.

### 7.1171.2 Function Documentation

#### 7.1171.2.1 SharedParamsInit()

```
void SharedParamsInit (
 void)
```

Initializes the shared RAM parameters module.

#### Returns

none.

### 7.1171.2.2 SharedParamsReadByIndex()

```
bool SharedParamsReadByIndex (
 uint32_t idx,
 uint8_t * value)
```

Reads a data byte from the shared parameter buffer at the specified index.

#### Parameters

|              |                                                                                                              |
|--------------|--------------------------------------------------------------------------------------------------------------|
| <i>idx</i>   | Index into the parameter data array. A valid value is between 0 and (SHARED_PARAMS_CFG_BUFFER_DATA_LEN - 1). |
| <i>value</i> | Pointer to where the read data value is stored.                                                              |

#### Returns

True if successful, false otherwise.

### 7.1171.2.3 SharedParamsWriteByIndex()

```
bool SharedParamsWriteByIndex (
 uint32_t idx,
 uint8_t value)
```

Writes a data byte to the shared parameter buffer at the specified index.

#### Parameters

|              |                                                                                                              |
|--------------|--------------------------------------------------------------------------------------------------------------|
| <i>idx</i>   | Index into the parameter data array. A valid value is between 0 and (SHARED_PARAMS_CFG_BUFFER_DATA_LEN - 1). |
| <i>value</i> | Value to write.                                                                                              |

#### Returns

True if successful, false otherwise.

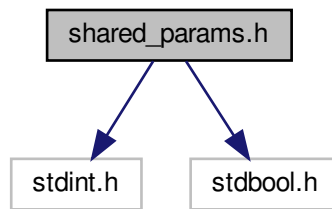
## 7.1172 shared\_params.h File Reference

Shared RAM parameters header file.

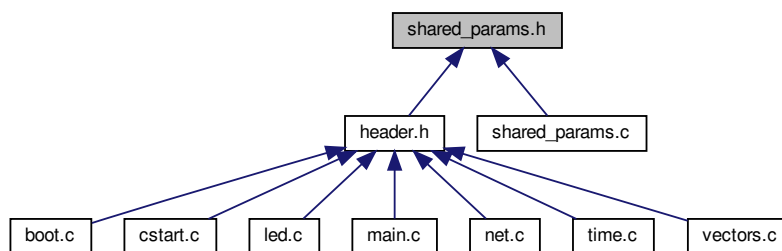
```
#include <stdint.h>
#include <stdbool.h>
```



Include dependency graph for ARMC3M3\_LM3S\_EK\_LM3S6965\_GCC/Prog/shared\_params.h:



This graph shows which files directly or indirectly include this file:



## Macros

- `#define` [SHARED\\_PARAMS\\_CFG\\_BUFFER\\_DATA\\_LEN](#) (56u)  
Configuration macro for specifying the size of the data inside the parameter buffer. This is the length in bytes of the actual parameter data, so excluding the bufferId and checksum.

## Functions

- void [SharedParamsInit](#) (void)  
Initializes the shared RAM parameters module.
- bool [SharedParamsReadByIndex](#) (uint32\_t idx, uint8\_t \*value)  
Reads a data byte from the shared parameter buffer at the specified index.
- bool [SharedParamsWriteByIndex](#) (uint32\_t idx, uint8\_t value)  
Writes a data byte to the shared parameter buffer at the specified index.

### 7.1172.1 Detailed Description

Shared RAM parameters header file.

## 7.1172.2 Function Documentation

### 7.1172.2.1 SharedParamsInit()

```
void SharedParamsInit (
 void)
```

Initializes the shared RAM parameters module.

#### Returns

none.

### 7.1172.2.2 SharedParamsReadByIndex()

```
bool SharedParamsReadByIndex (
 uint32_t idx,
 uint8_t * value)
```

Reads a data byte from the shared parameter buffer at the specified index.

#### Parameters

|              |                                                                                                              |
|--------------|--------------------------------------------------------------------------------------------------------------|
| <i>idx</i>   | Index into the parameter data array. A valid value is between 0 and (SHARED_PARAMS_CFG_BUFFER_DATA_LEN - 1). |
| <i>value</i> | Pointer to where the read data value is stored.                                                              |

#### Returns

True if successful, false otherwise.

### 7.1172.2.3 SharedParamsWriteByIndex()

```
bool SharedParamsWriteByIndex (
 uint32_t idx,
 uint8_t value)
```

Writes a data byte to the shared parameter buffer at the specified index.

#### Parameters

|              |                                                                                                              |
|--------------|--------------------------------------------------------------------------------------------------------------|
| <i>idx</i>   | Index into the parameter data array. A valid value is between 0 and (SHARED_PARAMS_CFG_BUFFER_DATA_LEN - 1). |
| <i>value</i> | Value to write.                                                                                              |

### Returns

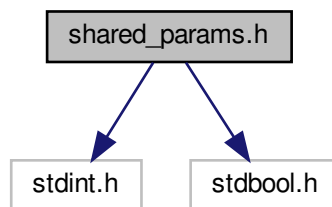
True if successful, false otherwise.

## 7.1173 shared\_params.h File Reference

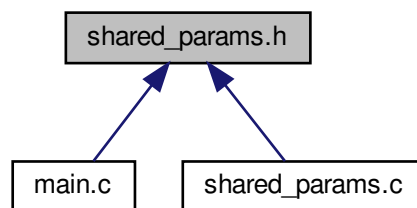
Shared RAM parameters header file.

```
#include <stdint.h>
#include <stdbool.h>
```

Include dependency graph for ARMCM3\_LM3S\_EK\_LM3S6965\_IAR/Boot/shared\_params.h:



This graph shows which files directly or indirectly include this file:



### Macros

- `#define SHARED_PARAMS_CFG_BUFFER_DATA_LEN (56u)`

*Configuration macro for specifying the size of the data inside the parameter buffer. This is the length in bytes of the actual parameter data, so excluding the bufferId and checksum.*

## Functions

- void [SharedParamsInit](#) (void)  
*Initializes the shared RAM parameters module.*
- bool [SharedParamsReadByIndex](#) (uint32\_t idx, uint8\_t \*value)  
*Reads a data byte from the shared parameter buffer at the specified index.*
- bool [SharedParamsWriteByIndex](#) (uint32\_t idx, uint8\_t value)  
*Writes a data byte to the shared parameter buffer at the specified index.*

### 7.1173.1 Detailed Description

Shared RAM parameters header file.

### 7.1173.2 Function Documentation

#### 7.1173.2.1 SharedParamsInit()

```
void SharedParamsInit (
 void)
```

Initializes the shared RAM parameters module.

#### Returns

none.

#### 7.1173.2.2 SharedParamsReadByIndex()

```
bool SharedParamsReadByIndex (
 uint32_t idx,
 uint8_t * value)
```

Reads a data byte from the shared parameter buffer at the specified index.

#### Parameters

|              |                                                                                                              |
|--------------|--------------------------------------------------------------------------------------------------------------|
| <i>idx</i>   | Index into the parameter data array. A valid value is between 0 and (SHARED_PARAMS_CFG_BUFFER_DATA_LEN - 1). |
| <i>value</i> | Pointer to where the read data value is stored.                                                              |

**Returns**

True if successful, false otherwise.

**7.1173.2.3 SharedParamsWriteByIndex()**

```
bool SharedParamsWriteByIndex (
 uint32_t idx,
 uint8_t value)
```

Writes a data byte to the shared parameter buffer at the specified index.

**Parameters**

|              |                                                                                                              |
|--------------|--------------------------------------------------------------------------------------------------------------|
| <i>idx</i>   | Index into the parameter data array. A valid value is between 0 and (SHARED_PARAMS_CFG_BUFFER_DATA_LEN - 1). |
| <i>value</i> | Value to write.                                                                                              |

**Returns**

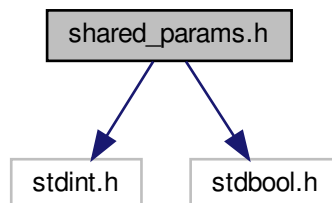
True if successful, false otherwise.

**7.1174 shared\_params.h File Reference**

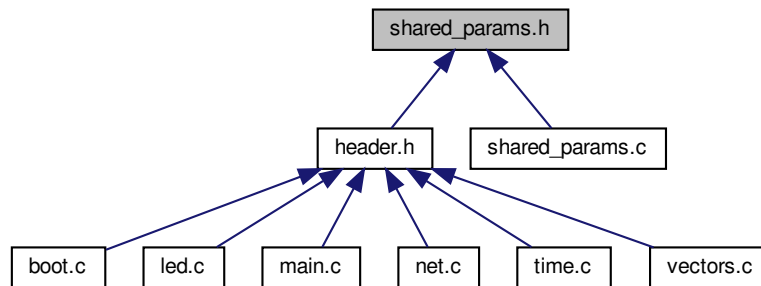
Shared RAM parameters header file.

```
#include <stdint.h>
#include <stdbool.h>
```

Include dependency graph for ARMCM3\_LM3S\_EK\_LM3S6965\_IAR/Prog/shared\_params.h:



This graph shows which files directly or indirectly include this file:



## Macros

- `#define` [SHARED\\_PARAMS\\_CFG\\_BUFFER\\_DATA\\_LEN](#) (56u)  
*Configuration macro for specifying the size of the data inside the parameter buffer. This is the length in bytes of the actual parameter data, so excluding the bufferId and checksum.*

## Functions

- `void` [SharedParamsInit](#) (void)  
*Initializes the shared RAM parameters module.*
- `bool` [SharedParamsReadByIndex](#) (uint32\_t idx, uint8\_t \*value)  
*Reads a data byte from the shared parameter buffer at the specified index.*
- `bool` [SharedParamsWriteByIndex](#) (uint32\_t idx, uint8\_t value)  
*Writes a data byte to the shared parameter buffer at the specified index.*

### 7.1174.1 Detailed Description

Shared RAM parameters header file.

### 7.1174.2 Function Documentation

#### 7.1174.2.1 SharedParamsInit()

```
void SharedParamsInit (
 void)
```

Initializes the shared RAM parameters module.

#### Returns

none.

### 7.1174.2.2 SharedParamsReadByIndex()

```
bool SharedParamsReadByIndex (
 uint32_t idx,
 uint8_t * value)
```

Reads a data byte from the shared parameter buffer at the specified index.

#### Parameters

|              |                                                                                                              |
|--------------|--------------------------------------------------------------------------------------------------------------|
| <i>idx</i>   | Index into the parameter data array. A valid value is between 0 and (SHARED_PARAMS_CFG_BUFFER_DATA_LEN - 1). |
| <i>value</i> | Pointer to where the read data value is stored.                                                              |

#### Returns

True if successful, false otherwise.

### 7.1174.2.3 SharedParamsWriteByIndex()

```
bool SharedParamsWriteByIndex (
 uint32_t idx,
 uint8_t value)
```

Writes a data byte to the shared parameter buffer at the specified index.

#### Parameters

|              |                                                                                                              |
|--------------|--------------------------------------------------------------------------------------------------------------|
| <i>idx</i>   | Index into the parameter data array. A valid value is between 0 and (SHARED_PARAMS_CFG_BUFFER_DATA_LEN - 1). |
| <i>value</i> | Value to write.                                                                                              |

#### Returns

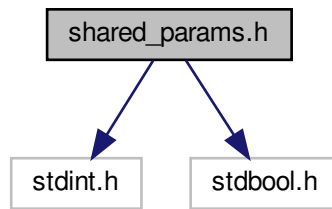
True if successful, false otherwise.

## 7.1175 shared\_params.h File Reference

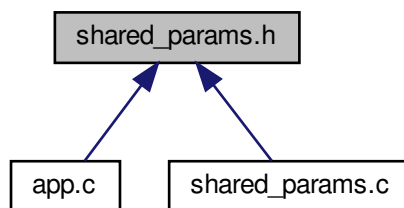
Shared RAM parameters header file.

```
#include <stdint.h>
#include <stdbool.h>
```

Include dependency graph for ARMCM4\_STM32F4\_Nucleo\_F429ZI\_CubeIDE/Boot/App/shared\_params.h:



This graph shows which files directly or indirectly include this file:



## Macros

- `#define` [SHARED\\_PARAMS\\_CFG\\_BUFFER\\_DATA\\_LEN](#) (56u)  
*Configuration macro for specifying the size of the data inside the parameter buffer. This is the length in bytes of the actual parameter data, so excluding the bufferId and checksum.*

## Functions

- void [SharedParamsInit](#) (void)  
*Initializes the shared RAM parameters module.*
- bool [SharedParamsReadByIndex](#) (uint32\_t idx, uint8\_t \*value)  
*Reads a data byte from the shared parameter buffer at the specified index.*
- bool [SharedParamsWriteByIndex](#) (uint32\_t idx, uint8\_t value)  
*Writes a data byte to the shared parameter buffer at the specified index.*

### 7.1175.1 Detailed Description

Shared RAM parameters header file.



## 7.1175.2 Function Documentation

### 7.1175.2.1 SharedParamsInit()

```
void SharedParamsInit (
 void)
```

Initializes the shared RAM parameters module.

#### Returns

none.

### 7.1175.2.2 SharedParamsReadByIndex()

```
bool SharedParamsReadByIndex (
 uint32_t idx,
 uint8_t * value)
```

Reads a data byte from the shared parameter buffer at the specified index.

#### Parameters

|              |                                                                                                              |
|--------------|--------------------------------------------------------------------------------------------------------------|
| <i>idx</i>   | Index into the parameter data array. A valid value is between 0 and (SHARED_PARAMS_CFG_BUFFER_DATA_LEN - 1). |
| <i>value</i> | Pointer to where the read data value is stored.                                                              |

#### Returns

True if successful, false otherwise.

### 7.1175.2.3 SharedParamsWriteByIndex()

```
bool SharedParamsWriteByIndex (
 uint32_t idx,
 uint8_t value)
```

Writes a data byte to the shared parameter buffer at the specified index.

#### Parameters

|              |                                                                                                              |
|--------------|--------------------------------------------------------------------------------------------------------------|
| <i>idx</i>   | Index into the parameter data array. A valid value is between 0 and (SHARED_PARAMS_CFG_BUFFER_DATA_LEN - 1). |
| <i>value</i> | Value to write.                                                                                              |

**Returns**

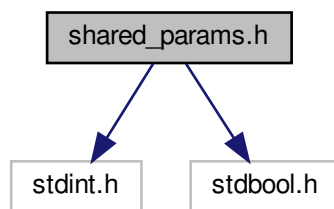
True if successful, false otherwise.

## 7.1176 shared\_params.h File Reference

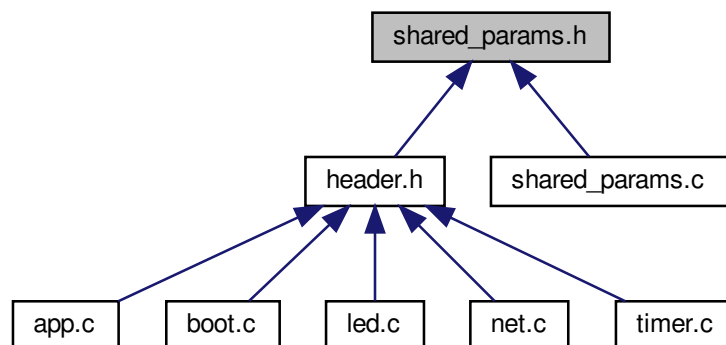
Shared RAM parameters header file.

```
#include <stdint.h>
#include <stdbool.h>
```

Include dependency graph for ARMCM4\_STM32F4\_Nucleo\_F429ZI\_CubeIDE/Prog/App/shared\_params.h:



This graph shows which files directly or indirectly include this file:

**Macros**

- `#define SHARED_PARAMS_CFG_BUFFER_DATA_LEN (56u)`

*Configuration macro for specifying the size of the data inside the parameter buffer. This is the length in bytes of the actual parameter data, so excluding the bufferId and checksum.*

## Functions

- void [SharedParamsInit](#) (void)  
*Initializes the shared RAM parameters module.*
- bool [SharedParamsReadByIndex](#) (uint32\_t idx, uint8\_t \*value)  
*Reads a data byte from the shared parameter buffer at the specified index.*
- bool [SharedParamsWriteByIndex](#) (uint32\_t idx, uint8\_t value)  
*Writes a data byte to the shared parameter buffer at the specified index.*

### 7.1176.1 Detailed Description

Shared RAM parameters header file.

### 7.1176.2 Function Documentation

#### 7.1176.2.1 SharedParamsInit()

```
void SharedParamsInit (
 void)
```

Initializes the shared RAM parameters module.

#### Returns

none.

#### 7.1176.2.2 SharedParamsReadByIndex()

```
bool SharedParamsReadByIndex (
 uint32_t idx,
 uint8_t * value)
```

Reads a data byte from the shared parameter buffer at the specified index.

#### Parameters

|              |                                                                                                              |
|--------------|--------------------------------------------------------------------------------------------------------------|
| <i>idx</i>   | Index into the parameter data array. A valid value is between 0 and (SHARED_PARAMS_CFG_BUFFER_DATA_LEN - 1). |
| <i>value</i> | Pointer to where the read data value is stored.                                                              |

**Returns**

True if successful, false otherwise.

**7.1176.2.3 SharedParamsWriteByIndex()**

```
bool SharedParamsWriteByIndex (
 uint32_t idx,
 uint8_t value)
```

Writes a data byte to the shared parameter buffer at the specified index.

**Parameters**

|              |                                                                                                              |
|--------------|--------------------------------------------------------------------------------------------------------------|
| <i>idx</i>   | Index into the parameter data array. A valid value is between 0 and (SHARED_PARAMS_CFG_BUFFER_DATA_LEN - 1). |
| <i>value</i> | Value to write.                                                                                              |

**Returns**

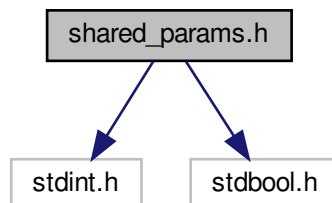
True if successful, false otherwise.

**7.1177 shared\_params.h File Reference**

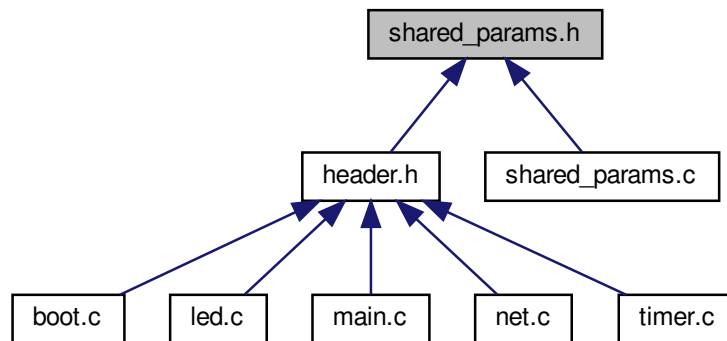
Shared RAM parameters header file.

```
#include <stdint.h>
#include <stdbool.h>
```

Include dependency graph for ARMCM4\_STM32F4\_Nucleo\_F429ZI\_GCC/Prog/shared\_params.h:



This graph shows which files directly or indirectly include this file:



## Macros

- `#define` [SHARED\\_PARAMS\\_CFG\\_BUFFER\\_DATA\\_LEN](#) (56u)  
*Configuration macro for specifying the size of the data inside the parameter buffer. This is the length in bytes of the actual parameter data, so excluding the bufferId and checksum.*

## Functions

- void [SharedParamsInit](#) (void)  
*Initializes the shared RAM parameters module.*
- bool [SharedParamsReadByIndex](#) (uint32\_t idx, uint8\_t \*value)  
*Reads a data byte from the shared parameter buffer at the specified index.*
- bool [SharedParamsWriteByIndex](#) (uint32\_t idx, uint8\_t value)  
*Writes a data byte to the shared parameter buffer at the specified index.*

### 7.1177.1 Detailed Description

Shared RAM parameters header file.

### 7.1177.2 Function Documentation

#### 7.1177.2.1 SharedParamsInit()

```
void SharedParamsInit (
 void)
```

Initializes the shared RAM parameters module.

#### Returns

none.

### 7.1177.2.2 SharedParamsReadByIndex()

```
bool SharedParamsReadByIndex (
 uint32_t idx,
 uint8_t * value)
```

Reads a data byte from the shared parameter buffer at the specified index.

#### Parameters

|              |                                                                                                              |
|--------------|--------------------------------------------------------------------------------------------------------------|
| <i>idx</i>   | Index into the parameter data array. A valid value is between 0 and (SHARED_PARAMS_CFG_BUFFER_DATA_LEN - 1). |
| <i>value</i> | Pointer to where the read data value is stored.                                                              |

#### Returns

True if successful, false otherwise.

### 7.1177.2.3 SharedParamsWriteByIndex()

```
bool SharedParamsWriteByIndex (
 uint32_t idx,
 uint8_t value)
```

Writes a data byte to the shared parameter buffer at the specified index.

#### Parameters

|              |                                                                                                              |
|--------------|--------------------------------------------------------------------------------------------------------------|
| <i>idx</i>   | Index into the parameter data array. A valid value is between 0 and (SHARED_PARAMS_CFG_BUFFER_DATA_LEN - 1). |
| <i>value</i> | Value to write.                                                                                              |

#### Returns

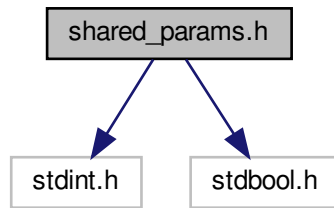
True if successful, false otherwise.

## 7.1178 shared\_params.h File Reference

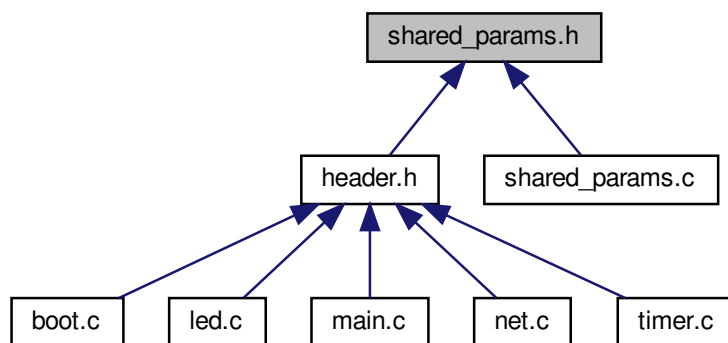
Shared RAM parameters header file.

```
#include <stdint.h>
#include <stdbool.h>
```

Include dependency graph for ARMC4\_STM32F4\_Nucleo\_F429ZI\_IAR/Prog/shared\_params.h:



This graph shows which files directly or indirectly include this file:



## Macros

- `#define` [SHARED\\_PARAMS\\_CFG\\_BUFFER\\_DATA\\_LEN](#) (56u)

*Configuration macro for specifying the size of the data inside the parameter buffer. This is the length in bytes of the actual parameter data, so excluding the bufferId and checksum.*

## Functions

- void [SharedParamsInit](#) (void)  
*Initializes the shared RAM parameters module.*
- bool [SharedParamsReadByIndex](#) (uint32\_t idx, uint8\_t \*value)  
*Reads a data byte from the shared parameter buffer at the specified index.*
- bool [SharedParamsWriteByIndex](#) (uint32\_t idx, uint8\_t value)  
*Writes a data byte to the shared parameter buffer at the specified index.*

### 7.1178.1 Detailed Description

Shared RAM parameters header file.

### 7.1178.2 Function Documentation

#### 7.1178.2.1 SharedParamsInit()

```
void SharedParamsInit (
 void)
```

Initializes the shared RAM parameters module.

##### Returns

none.

#### 7.1178.2.2 SharedParamsReadByIndex()

```
bool SharedParamsReadByIndex (
 uint32_t idx,
 uint8_t * value)
```

Reads a data byte from the shared parameter buffer at the specified index.

##### Parameters

|              |                                                                                                              |
|--------------|--------------------------------------------------------------------------------------------------------------|
| <i>idx</i>   | Index into the parameter data array. A valid value is between 0 and (SHARED_PARAMS_CFG_BUFFER_DATA_LEN - 1). |
| <i>value</i> | Pointer to where the read data value is stored.                                                              |

##### Returns

True if successful, false otherwise.

#### 7.1178.2.3 SharedParamsWriteByIndex()

```
bool SharedParamsWriteByIndex (
 uint32_t idx,
 uint8_t value)
```

Writes a data byte to the shared parameter buffer at the specified index.



## Parameters

|              |                                                                                                              |
|--------------|--------------------------------------------------------------------------------------------------------------|
| <i>idx</i>   | Index into the parameter data array. A valid value is between 0 and (SHARED_PARAMS_CFG_BUFFER_DATA_LEN - 1). |
| <i>value</i> | Value to write.                                                                                              |

## Returns

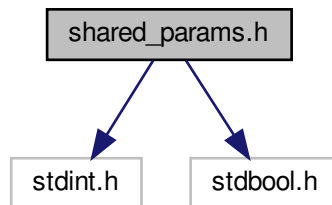
True if successful, false otherwise.

## 7.1179 shared\_params.h File Reference

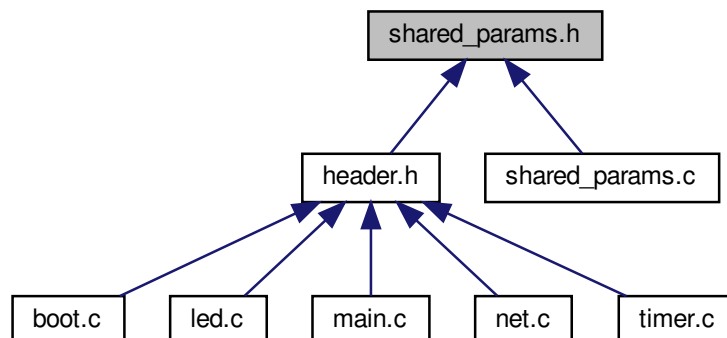
Shared RAM parameters header file.

```
#include <stdint.h>
#include <stdbool.h>
```

Include dependency graph for ARMCM4\_STM32F4\_Nucleo\_F429ZI\_Keil/Prog/shared\_params.h:



This graph shows which files directly or indirectly include this file:



## Macros

- #define [SHARED\\_PARAMS\\_CFG\\_BUFFER\\_DATA\\_LEN](#) (56u)

*Configuration macro for specifying the size of the data inside the parameter buffer. This is the length in bytes of the actual parameter data, so excluding the bufferId and checksum.*

## Functions

- void [SharedParamsInit](#) (void)  
*Initializes the shared RAM parameters module.*
- bool [SharedParamsReadByIndex](#) (uint32\_t idx, uint8\_t \*value)  
*Reads a data byte from the shared parameter buffer at the specified index.*
- bool [SharedParamsWriteByIndex](#) (uint32\_t idx, uint8\_t value)  
*Writes a data byte to the shared parameter buffer at the specified index.*

### 7.1179.1 Detailed Description

Shared RAM parameters header file.

### 7.1179.2 Function Documentation

#### 7.1179.2.1 SharedParamsInit()

```
void SharedParamsInit (
 void)
```

Initializes the shared RAM parameters module.

#### Returns

none.

#### 7.1179.2.2 SharedParamsReadByIndex()

```
bool SharedParamsReadByIndex (
 uint32_t idx,
 uint8_t * value)
```

Reads a data byte from the shared parameter buffer at the specified index.

## Parameters

|              |                                                                                                              |
|--------------|--------------------------------------------------------------------------------------------------------------|
| <i>idx</i>   | Index into the parameter data array. A valid value is between 0 and (SHARED_PARAMS_CFG_BUFFER_DATA_LEN - 1). |
| <i>value</i> | Pointer to where the read data value is stored.                                                              |

## Returns

True if successful, false otherwise.

## 7.1179.2.3 SharedParamsWriteByIndex()

```
bool SharedParamsWriteByIndex (
 uint32_t idx,
 uint8_t value)
```

Writes a data byte to the shared parameter buffer at the specified index.

## Parameters

|              |                                                                                                              |
|--------------|--------------------------------------------------------------------------------------------------------------|
| <i>idx</i>   | Index into the parameter data array. A valid value is between 0 and (SHARED_PARAMS_CFG_BUFFER_DATA_LEN - 1). |
| <i>value</i> | Value to write.                                                                                              |

## Returns

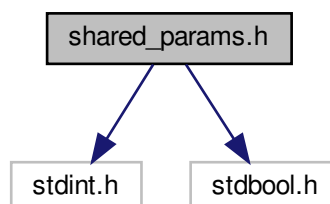
True if successful, false otherwise.

## 7.1180 shared\_params.h File Reference

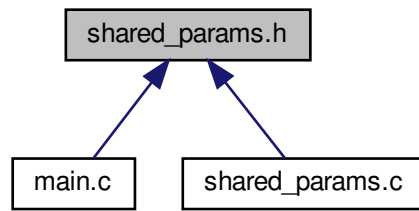
Shared RAM parameters header file.

```
#include <stdint.h>
#include <stdbool.h>
```

Include dependency graph for ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_GCC/Boot/shared\_params.h:



This graph shows which files directly or indirectly include this file:



## Macros

- `#define SHARED_PARAMS_CFG_BUFFER_DATA_LEN (56u)`  
*Configuration macro for specifying the size of the data inside the parameter buffer. This is the length in bytes of the actual parameter data, so excluding the bufferId and checksum.*

## Functions

- `void SharedParamsInit (void)`  
*Initializes the shared RAM parameters module.*
- `bool SharedParamsReadByIndex (uint32_t idx, uint8_t *value)`  
*Reads a data byte from the shared parameter buffer at the specified index.*
- `bool SharedParamsWriteByIndex (uint32_t idx, uint8_t value)`  
*Writes a data byte to the shared parameter buffer at the specified index.*

### 7.1180.1 Detailed Description

Shared RAM parameters header file.

### 7.1180.2 Function Documentation

#### 7.1180.2.1 SharedParamsInit()

```
void SharedParamsInit (
 void)
```

Initializes the shared RAM parameters module.

#### Returns

none.

### 7.1180.2.2 SharedParamsReadByIndex()

```
bool SharedParamsReadByIndex (
 uint32_t idx,
 uint8_t * value)
```

Reads a data byte from the shared parameter buffer at the specified index.

#### Parameters

|              |                                                                                                              |
|--------------|--------------------------------------------------------------------------------------------------------------|
| <i>idx</i>   | Index into the parameter data array. A valid value is between 0 and (SHARED_PARAMS_CFG_BUFFER_DATA_LEN - 1). |
| <i>value</i> | Pointer to where the read data value is stored.                                                              |

#### Returns

True if successful, false otherwise.

### 7.1180.2.3 SharedParamsWriteByIndex()

```
bool SharedParamsWriteByIndex (
 uint32_t idx,
 uint8_t value)
```

Writes a data byte to the shared parameter buffer at the specified index.

#### Parameters

|              |                                                                                                              |
|--------------|--------------------------------------------------------------------------------------------------------------|
| <i>idx</i>   | Index into the parameter data array. A valid value is between 0 and (SHARED_PARAMS_CFG_BUFFER_DATA_LEN - 1). |
| <i>value</i> | Value to write.                                                                                              |

#### Returns

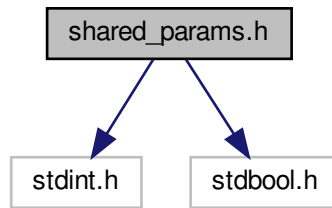
True if successful, false otherwise.

## 7.1181 shared\_params.h File Reference

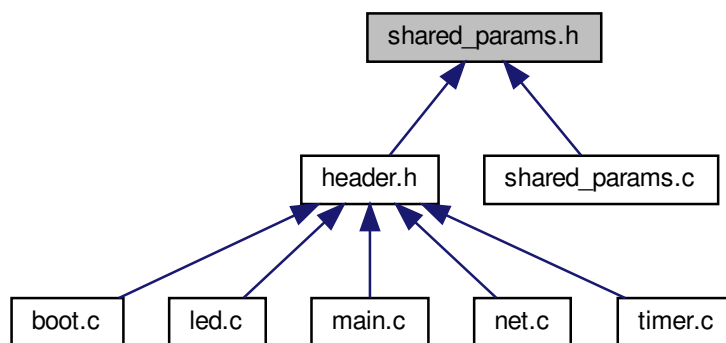
Shared RAM parameters header file.

```
#include <stdint.h>
#include <stdbool.h>
```

Include dependency graph for ARMC4\_XMC4\_XMC4700\_Relax\_Kit\_GCC/Prog/shared\_params.h:



This graph shows which files directly or indirectly include this file:



## Macros

- `#define SHARED_PARAMS_CFG_BUFFER_DATA_LEN (56u)`  
*Configuration macro for specifying the size of the data inside the parameter buffer. This is the length in bytes of the actual parameter data, so excluding the bufferId and checksum.*

## Functions

- void `SharedParamsInit` (void)  
*Initializes the shared RAM parameters module.*
- bool `SharedParamsReadByIndex` (uint32\_t idx, uint8\_t \*value)  
*Reads a data byte from the shared parameter buffer at the specified index.*
- bool `SharedParamsWriteByIndex` (uint32\_t idx, uint8\_t value)  
*Writes a data byte to the shared parameter buffer at the specified index.*

### 7.1181.1 Detailed Description

Shared RAM parameters header file.

### 7.1181.2 Function Documentation

#### 7.1181.2.1 SharedParamsInit()

```
void SharedParamsInit (
 void)
```

Initializes the shared RAM parameters module.

##### Returns

none.

#### 7.1181.2.2 SharedParamsReadByIndex()

```
bool SharedParamsReadByIndex (
 uint32_t idx,
 uint8_t * value)
```

Reads a data byte from the shared parameter buffer at the specified index.

##### Parameters

|              |                                                                                                              |
|--------------|--------------------------------------------------------------------------------------------------------------|
| <i>idx</i>   | Index into the parameter data array. A valid value is between 0 and (SHARED_PARAMS_CFG_BUFFER_DATA_LEN - 1). |
| <i>value</i> | Pointer to where the read data value is stored.                                                              |

##### Returns

True if successful, false otherwise.

#### 7.1181.2.3 SharedParamsWriteByIndex()

```
bool SharedParamsWriteByIndex (
 uint32_t idx,
 uint8_t value)
```

Writes a data byte to the shared parameter buffer at the specified index.

## Parameters

|              |                                                                                                              |
|--------------|--------------------------------------------------------------------------------------------------------------|
| <i>idx</i>   | Index into the parameter data array. A valid value is between 0 and (SHARED_PARAMS_CFG_BUFFER_DATA_LEN - 1). |
| <i>value</i> | Value to write.                                                                                              |

## Returns

True if successful, false otherwise.

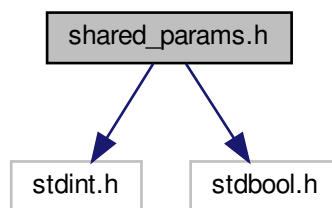
## 7.1182 shared\_params.h File Reference

Shared RAM parameters header file.

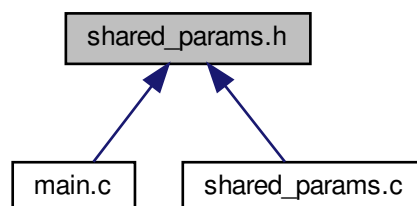
```
#include <stdint.h>
```

```
#include <stdbool.h>
```

Include dependency graph for ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_IAR/Boot/shared\_params.h:



This graph shows which files directly or indirectly include this file:



## Macros

- #define [SHARED\\_PARAMS\\_CFG\\_BUFFER\\_DATA\\_LEN](#) (56u)

*Configuration macro for specifying the size of the data inside the parameter buffer. This is the length in bytes of the actual parameter data, so excluding the bufferId and checksum.*



## Functions

- void [SharedParamsInit](#) (void)  
*Initializes the shared RAM parameters module.*
- bool [SharedParamsReadByIndex](#) (uint32\_t idx, uint8\_t \*value)  
*Reads a data byte from the shared parameter buffer at the specified index.*
- bool [SharedParamsWriteByIndex](#) (uint32\_t idx, uint8\_t value)  
*Writes a data byte to the shared parameter buffer at the specified index.*

### 7.1182.1 Detailed Description

Shared RAM parameters header file.

### 7.1182.2 Function Documentation

#### 7.1182.2.1 SharedParamsInit()

```
void SharedParamsInit (
 void)
```

Initializes the shared RAM parameters module.

#### Returns

none.

#### 7.1182.2.2 SharedParamsReadByIndex()

```
bool SharedParamsReadByIndex (
 uint32_t idx,
 uint8_t * value)
```

Reads a data byte from the shared parameter buffer at the specified index.

#### Parameters

|              |                                                                                                              |
|--------------|--------------------------------------------------------------------------------------------------------------|
| <i>idx</i>   | Index into the parameter data array. A valid value is between 0 and (SHARED_PARAMS_CFG_BUFFER_DATA_LEN - 1). |
| <i>value</i> | Pointer to where the read data value is stored.                                                              |

**Returns**

True if successful, false otherwise.

**7.1182.2.3 SharedParamsWriteByIndex()**

```
bool SharedParamsWriteByIndex (
 uint32_t idx,
 uint8_t value)
```

Writes a data byte to the shared parameter buffer at the specified index.

**Parameters**

|              |                                                                                                              |
|--------------|--------------------------------------------------------------------------------------------------------------|
| <i>idx</i>   | Index into the parameter data array. A valid value is between 0 and (SHARED_PARAMS_CFG_BUFFER_DATA_LEN - 1). |
| <i>value</i> | Value to write.                                                                                              |

**Returns**

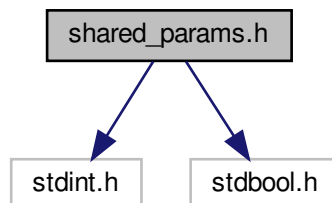
True if successful, false otherwise.

**7.1183 shared\_params.h File Reference**

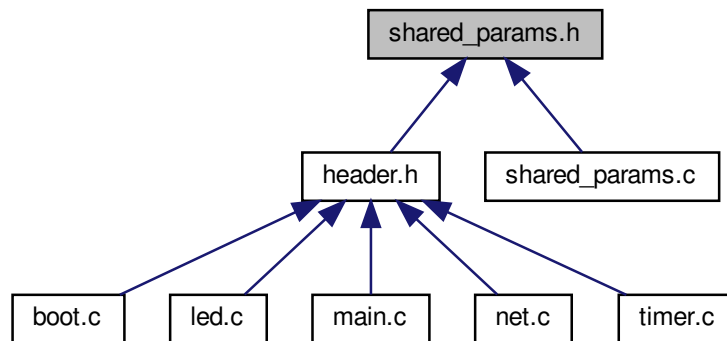
Shared RAM parameters header file.

```
#include <stdint.h>
#include <stdbool.h>
```

Include dependency graph for ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_IAR/Prog/shared\_params.h:



This graph shows which files directly or indirectly include this file:



## Macros

- `#define` [SHARED\\_PARAMS\\_CFG\\_BUFFER\\_DATA\\_LEN](#) (56u)  
*Configuration macro for specifying the size of the data inside the parameter buffer. This is the length in bytes of the actual parameter data, so excluding the bufferId and checksum.*

## Functions

- void [SharedParamsInit](#) (void)  
*Initializes the shared RAM parameters module.*
- bool [SharedParamsReadByIndex](#) (uint32\_t idx, uint8\_t \*value)  
*Reads a data byte from the shared parameter buffer at the specified index.*
- bool [SharedParamsWriteByIndex](#) (uint32\_t idx, uint8\_t value)  
*Writes a data byte to the shared parameter buffer at the specified index.*

### 7.1183.1 Detailed Description

Shared RAM parameters header file.

### 7.1183.2 Function Documentation

#### 7.1183.2.1 SharedParamsInit()

```
void SharedParamsInit (
 void)
```

Initializes the shared RAM parameters module.

#### Returns

none.

### 7.1183.2.2 SharedParamsReadByIndex()

```
bool SharedParamsReadByIndex (
 uint32_t idx,
 uint8_t * value)
```

Reads a data byte from the shared parameter buffer at the specified index.

#### Parameters

|              |                                                                                                              |
|--------------|--------------------------------------------------------------------------------------------------------------|
| <i>idx</i>   | Index into the parameter data array. A valid value is between 0 and (SHARED_PARAMS_CFG_BUFFER_DATA_LEN - 1). |
| <i>value</i> | Pointer to where the read data value is stored.                                                              |

#### Returns

True if successful, false otherwise.

### 7.1183.2.3 SharedParamsWriteByIndex()

```
bool SharedParamsWriteByIndex (
 uint32_t idx,
 uint8_t value)
```

Writes a data byte to the shared parameter buffer at the specified index.

#### Parameters

|              |                                                                                                              |
|--------------|--------------------------------------------------------------------------------------------------------------|
| <i>idx</i>   | Index into the parameter data array. A valid value is between 0 and (SHARED_PARAMS_CFG_BUFFER_DATA_LEN - 1). |
| <i>value</i> | Value to write.                                                                                              |

#### Returns

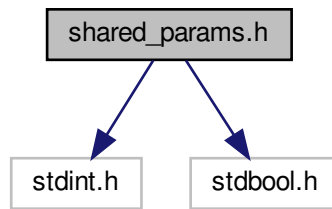
True if successful, false otherwise.

## 7.1184 shared\_params.h File Reference

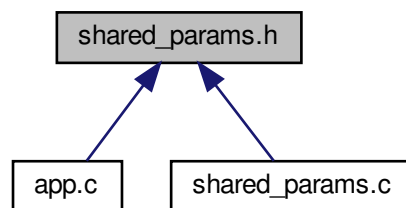
Shared RAM parameters header file.

```
#include <stdint.h>
#include <stdbool.h>
```

Include dependency graph for ARMC7\_STM32F7\_Nucleo\_F767ZI\_CubeIDE/Boot/App/shared\_params.h:



This graph shows which files directly or indirectly include this file:



## Macros

- `#define SHARED_PARAMS_CFG_BUFFER_DATA_LEN (56u)`  
*Configuration macro for specifying the size of the data inside the parameter buffer. This is the length in bytes of the actual parameter data, so excluding the bufferId and checksum.*

## Functions

- void `SharedParamsInit` (void)  
*Initializes the shared RAM parameters module.*
- bool `SharedParamsReadByIndex` (uint32\_t idx, uint8\_t \*value)  
*Reads a data byte from the shared parameter buffer at the specified index.*
- bool `SharedParamsWriteByIndex` (uint32\_t idx, uint8\_t value)  
*Writes a data byte to the shared parameter buffer at the specified index.*

### 7.1184.1 Detailed Description

Shared RAM parameters header file.

## 7.1184.2 Function Documentation

### 7.1184.2.1 SharedParamsInit()

```
void SharedParamsInit (
 void)
```

Initializes the shared RAM parameters module.

#### Returns

none.

### 7.1184.2.2 SharedParamsReadByIndex()

```
bool SharedParamsReadByIndex (
 uint32_t idx,
 uint8_t * value)
```

Reads a data byte from the shared parameter buffer at the specified index.

#### Parameters

|              |                                                                                                              |
|--------------|--------------------------------------------------------------------------------------------------------------|
| <i>idx</i>   | Index into the parameter data array. A valid value is between 0 and (SHARED_PARAMS_CFG_BUFFER_DATA_LEN - 1). |
| <i>value</i> | Pointer to where the read data value is stored.                                                              |

#### Returns

True if successful, false otherwise.

### 7.1184.2.3 SharedParamsWriteByIndex()

```
bool SharedParamsWriteByIndex (
 uint32_t idx,
 uint8_t value)
```

Writes a data byte to the shared parameter buffer at the specified index.

#### Parameters

|              |                                                                                                              |
|--------------|--------------------------------------------------------------------------------------------------------------|
| <i>idx</i>   | Index into the parameter data array. A valid value is between 0 and (SHARED_PARAMS_CFG_BUFFER_DATA_LEN - 1). |
| <i>value</i> | Value to write.                                                                                              |

### Returns

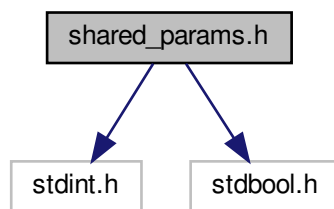
True if successful, false otherwise.

## 7.1185 shared\_params.h File Reference

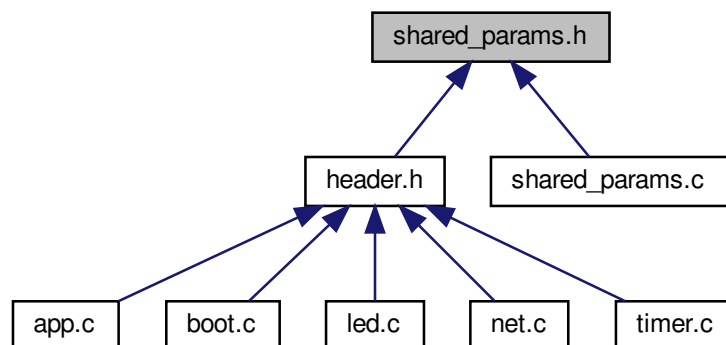
Shared RAM parameters header file.

```
#include <stdint.h>
#include <stdbool.h>
```

Include dependency graph for ARMCM7\_STM32F7\_Nucleo\_F767ZI\_CubeIDE/Prog/App/shared\_params.h:



This graph shows which files directly or indirectly include this file:



### Macros

- `#define SHARED_PARAMS_CFG_BUFFER_DATA_LEN (56u)`

*Configuration macro for specifying the size of the data inside the parameter buffer. This is the length in bytes of the actual parameter data, so excluding the bufferId and checksum.*

## Functions

- void [SharedParamsInit](#) (void)  
*Initializes the shared RAM parameters module.*
- bool [SharedParamsReadByIndex](#) (uint32\_t idx, uint8\_t \*value)  
*Reads a data byte from the shared parameter buffer at the specified index.*
- bool [SharedParamsWriteByIndex](#) (uint32\_t idx, uint8\_t value)  
*Writes a data byte to the shared parameter buffer at the specified index.*

### 7.1185.1 Detailed Description

Shared RAM parameters header file.

### 7.1185.2 Function Documentation

#### 7.1185.2.1 SharedParamsInit()

```
void SharedParamsInit (
 void)
```

Initializes the shared RAM parameters module.

#### Returns

none.

#### 7.1185.2.2 SharedParamsReadByIndex()

```
bool SharedParamsReadByIndex (
 uint32_t idx,
 uint8_t * value)
```

Reads a data byte from the shared parameter buffer at the specified index.

#### Parameters

|              |                                                                                                              |
|--------------|--------------------------------------------------------------------------------------------------------------|
| <i>idx</i>   | Index into the parameter data array. A valid value is between 0 and (SHARED_PARAMS_CFG_BUFFER_DATA_LEN - 1). |
| <i>value</i> | Pointer to where the read data value is stored.                                                              |



**Returns**

True if successful, false otherwise.

**7.1185.2.3 SharedParamsWriteByIndex()**

```
bool SharedParamsWriteByIndex (
 uint32_t idx,
 uint8_t value)
```

Writes a data byte to the shared parameter buffer at the specified index.

**Parameters**

|              |                                                                                                              |
|--------------|--------------------------------------------------------------------------------------------------------------|
| <i>idx</i>   | Index into the parameter data array. A valid value is between 0 and (SHARED_PARAMS_CFG_BUFFER_DATA_LEN - 1). |
| <i>value</i> | Value to write.                                                                                              |

**Returns**

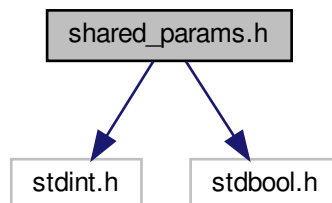
True if successful, false otherwise.

**7.1186 shared\_params.h File Reference**

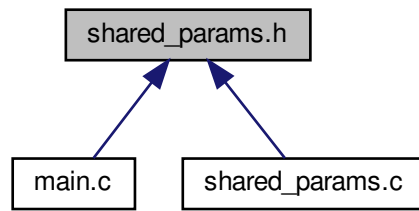
Shared RAM parameters header file.

```
#include <stdint.h>
#include <stdbool.h>
```

Include dependency graph for ARMCM7\_STM32F7\_Nucleo\_F767ZI\_GCC/Boot/shared\_params.h:



This graph shows which files directly or indirectly include this file:



## Macros

- `#define SHARED_PARAMS_CFG_BUFFER_DATA_LEN (56u)`  
*Configuration macro for specifying the size of the data inside the parameter buffer. This is the length in bytes of the actual parameter data, so excluding the bufferId and checksum.*

## Functions

- `void SharedParamsInit (void)`  
*Initializes the shared RAM parameters module.*
- `bool SharedParamsReadByIndex (uint32_t idx, uint8_t *value)`  
*Reads a data byte from the shared parameter buffer at the specified index.*
- `bool SharedParamsWriteByIndex (uint32_t idx, uint8_t value)`  
*Writes a data byte to the shared parameter buffer at the specified index.*

### 7.1186.1 Detailed Description

Shared RAM parameters header file.

### 7.1186.2 Function Documentation

#### 7.1186.2.1 SharedParamsInit()

```
void SharedParamsInit (
 void)
```

Initializes the shared RAM parameters module.

#### Returns

none.

### 7.1186.2.2 SharedParamsReadByIndex()

```
bool SharedParamsReadByIndex (
 uint32_t idx,
 uint8_t * value)
```

Reads a data byte from the shared parameter buffer at the specified index.

#### Parameters

|              |                                                                                                              |
|--------------|--------------------------------------------------------------------------------------------------------------|
| <i>idx</i>   | Index into the parameter data array. A valid value is between 0 and (SHARED_PARAMS_CFG_BUFFER_DATA_LEN - 1). |
| <i>value</i> | Pointer to where the read data value is stored.                                                              |

#### Returns

True if successful, false otherwise.

### 7.1186.2.3 SharedParamsWriteByIndex()

```
bool SharedParamsWriteByIndex (
 uint32_t idx,
 uint8_t value)
```

Writes a data byte to the shared parameter buffer at the specified index.

#### Parameters

|              |                                                                                                              |
|--------------|--------------------------------------------------------------------------------------------------------------|
| <i>idx</i>   | Index into the parameter data array. A valid value is between 0 and (SHARED_PARAMS_CFG_BUFFER_DATA_LEN - 1). |
| <i>value</i> | Value to write.                                                                                              |

#### Returns

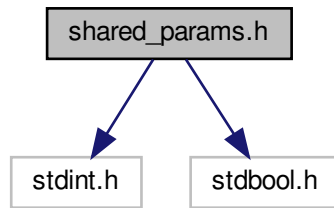
True if successful, false otherwise.

## 7.1187 shared\_params.h File Reference

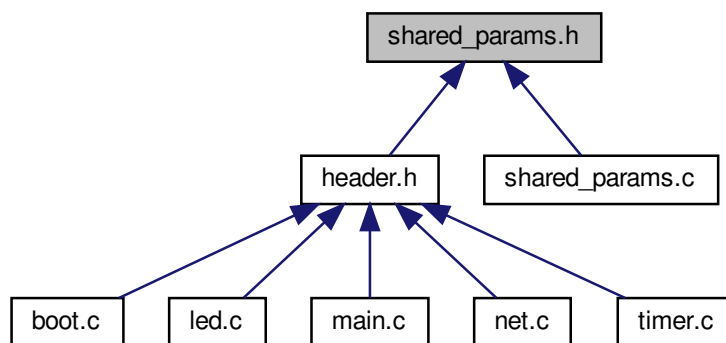
Shared RAM parameters header file.

```
#include <stdint.h>
#include <stdbool.h>
```

Include dependency graph for ARMC7\_STM32F7\_Nucleo\_F767ZI\_GCC/Prog/shared\_params.h:



This graph shows which files directly or indirectly include this file:



## Macros

- `#define` [SHARED\\_PARAMS\\_CFG\\_BUFFER\\_DATA\\_LEN](#) (56u)

*Configuration macro for specifying the size of the data inside the parameter buffer. This is the length in bytes of the actual parameter data, so excluding the bufferId and checksum.*

## Functions

- void [SharedParamsInit](#) (void)  
*Initializes the shared RAM parameters module.*
- bool [SharedParamsReadByIndex](#) (uint32\_t idx, uint8\_t \*value)  
*Reads a data byte from the shared parameter buffer at the specified index.*
- bool [SharedParamsWriteByIndex](#) (uint32\_t idx, uint8\_t value)  
*Writes a data byte to the shared parameter buffer at the specified index.*

## 7.1187.1 Detailed Description

Shared RAM parameters header file.

## 7.1187.2 Function Documentation

### 7.1187.2.1 SharedParamsInit()

```
void SharedParamsInit (
 void)
```

Initializes the shared RAM parameters module.

#### Returns

none.

### 7.1187.2.2 SharedParamsReadByIndex()

```
bool SharedParamsReadByIndex (
 uint32_t idx,
 uint8_t * value)
```

Reads a data byte from the shared parameter buffer at the specified index.

#### Parameters

|              |                                                                                                              |
|--------------|--------------------------------------------------------------------------------------------------------------|
| <i>idx</i>   | Index into the parameter data array. A valid value is between 0 and (SHARED_PARAMS_CFG_BUFFER_DATA_LEN - 1). |
| <i>value</i> | Pointer to where the read data value is stored.                                                              |

#### Returns

True if successful, false otherwise.

### 7.1187.2.3 SharedParamsWriteByIndex()

```
bool SharedParamsWriteByIndex (
 uint32_t idx,
 uint8_t value)
```

Writes a data byte to the shared parameter buffer at the specified index.

## Parameters

|              |                                                                                                              |
|--------------|--------------------------------------------------------------------------------------------------------------|
| <i>idx</i>   | Index into the parameter data array. A valid value is between 0 and (SHARED_PARAMS_CFG_BUFFER_DATA_LEN - 1). |
| <i>value</i> | Value to write.                                                                                              |

## Returns

True if successful, false otherwise.

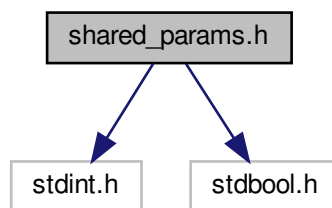
## 7.1188 shared\_params.h File Reference

Shared RAM parameters header file.

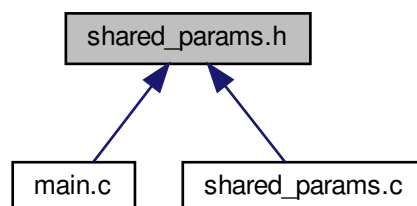
```
#include <stdint.h>
```

```
#include <stdbool.h>
```

Include dependency graph for ARMCM7\_STM32F7\_Nucleo\_F767ZI\_IAR/Boot/shared\_params.h:



This graph shows which files directly or indirectly include this file:



## Macros

- #define [SHARED\\_PARAMS\\_CFG\\_BUFFER\\_DATA\\_LEN](#) (56u)

*Configuration macro for specifying the size of the data inside the parameter buffer. This is the length in bytes of the actual parameter data, so excluding the bufferId and checksum.*

## Functions

- void [SharedParamsInit](#) (void)  
*Initializes the shared RAM parameters module.*
- bool [SharedParamsReadByIndex](#) (uint32\_t idx, uint8\_t \*value)  
*Reads a data byte from the shared parameter buffer at the specified index.*
- bool [SharedParamsWriteByIndex](#) (uint32\_t idx, uint8\_t value)  
*Writes a data byte to the shared parameter buffer at the specified index.*

### 7.1188.1 Detailed Description

Shared RAM parameters header file.

### 7.1188.2 Function Documentation

#### 7.1188.2.1 SharedParamsInit()

```
void SharedParamsInit (
 void)
```

Initializes the shared RAM parameters module.

#### Returns

none.

#### 7.1188.2.2 SharedParamsReadByIndex()

```
bool SharedParamsReadByIndex (
 uint32_t idx,
 uint8_t * value)
```

Reads a data byte from the shared parameter buffer at the specified index.

#### Parameters

|              |                                                                                                              |
|--------------|--------------------------------------------------------------------------------------------------------------|
| <i>idx</i>   | Index into the parameter data array. A valid value is between 0 and (SHARED_PARAMS_CFG_BUFFER_DATA_LEN - 1). |
| <i>value</i> | Pointer to where the read data value is stored.                                                              |

**Returns**

True if successful, false otherwise.

**7.1188.2.3 SharedParamsWriteByIndex()**

```
bool SharedParamsWriteByIndex (
 uint32_t idx,
 uint8_t value)
```

Writes a data byte to the shared parameter buffer at the specified index.

**Parameters**

|              |                                                                                                              |
|--------------|--------------------------------------------------------------------------------------------------------------|
| <i>idx</i>   | Index into the parameter data array. A valid value is between 0 and (SHARED_PARAMS_CFG_BUFFER_DATA_LEN - 1). |
| <i>value</i> | Value to write.                                                                                              |

**Returns**

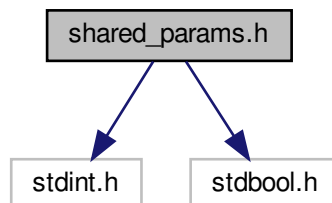
True if successful, false otherwise.

**7.1189 shared\_params.h File Reference**

Shared RAM parameters header file.

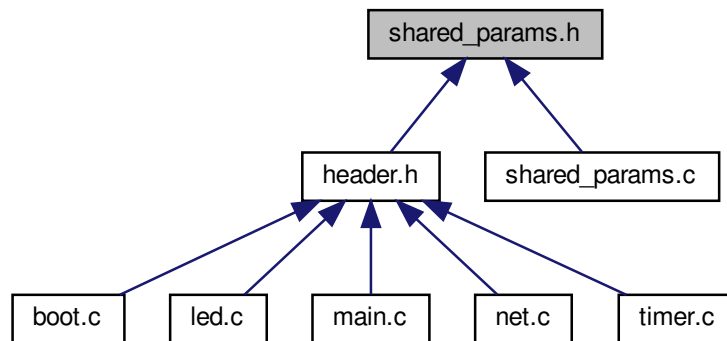
```
#include <stdint.h>
#include <stdbool.h>
```

Include dependency graph for ARMCM7\_STM32F7\_Nucleo\_F767ZI\_IAR/Prog/shared\_params.h:





This graph shows which files directly or indirectly include this file:



## Macros

- `#define` [SHARED\\_PARAMS\\_CFG\\_BUFFER\\_DATA\\_LEN](#) (56u)  
*Configuration macro for specifying the size of the data inside the parameter buffer. This is the length in bytes of the actual parameter data, so excluding the bufferId and checksum.*

## Functions

- `void` [SharedParamsInit](#) (void)  
*Initializes the shared RAM parameters module.*
- `bool` [SharedParamsReadByIndex](#) (uint32\_t idx, uint8\_t \*value)  
*Reads a data byte from the shared parameter buffer at the specified index.*
- `bool` [SharedParamsWriteByIndex](#) (uint32\_t idx, uint8\_t value)  
*Writes a data byte to the shared parameter buffer at the specified index.*

### 7.1189.1 Detailed Description

Shared RAM parameters header file.

### 7.1189.2 Function Documentation

#### 7.1189.2.1 SharedParamsInit()

```
void SharedParamsInit (
 void)
```

Initializes the shared RAM parameters module.

#### Returns

none.

### 7.1189.2.2 SharedParamsReadByIndex()

```
bool SharedParamsReadByIndex (
 uint32_t idx,
 uint8_t * value)
```

Reads a data byte from the shared parameter buffer at the specified index.

#### Parameters

|              |                                                                                                              |
|--------------|--------------------------------------------------------------------------------------------------------------|
| <i>idx</i>   | Index into the parameter data array. A valid value is between 0 and (SHARED_PARAMS_CFG_BUFFER_DATA_LEN - 1). |
| <i>value</i> | Pointer to where the read data value is stored.                                                              |

#### Returns

True if successful, false otherwise.

### 7.1189.2.3 SharedParamsWriteByIndex()

```
bool SharedParamsWriteByIndex (
 uint32_t idx,
 uint8_t value)
```

Writes a data byte to the shared parameter buffer at the specified index.

#### Parameters

|              |                                                                                                              |
|--------------|--------------------------------------------------------------------------------------------------------------|
| <i>idx</i>   | Index into the parameter data array. A valid value is between 0 and (SHARED_PARAMS_CFG_BUFFER_DATA_LEN - 1). |
| <i>value</i> | Value to write.                                                                                              |

#### Returns

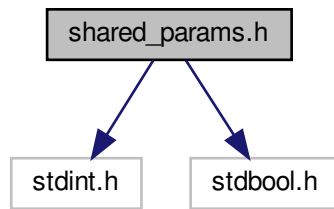
True if successful, false otherwise.

## 7.1190 shared\_params.h File Reference

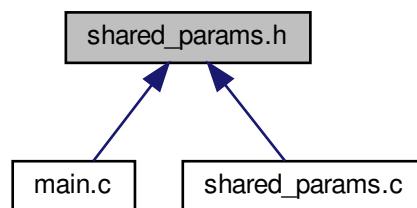
Shared RAM parameters header file.

```
#include <stdint.h>
#include <stdbool.h>
```

Include dependency graph for ARMC7\_STM32F7\_Nucleo\_F767ZI\_Keil/Boot/shared\_params.h:



This graph shows which files directly or indirectly include this file:



## Macros

- `#define` [SHARED\\_PARAMS\\_CFG\\_BUFFER\\_DATA\\_LEN](#) (56u)  
*Configuration macro for specifying the size of the data inside the parameter buffer. This is the length in bytes of the actual parameter data, so excluding the bufferId and checksum.*

## Functions

- void [SharedParamsInit](#) (void)  
*Initializes the shared RAM parameters module.*
- bool [SharedParamsReadByIndex](#) (uint32\_t idx, uint8\_t \*value)  
*Reads a data byte from the shared parameter buffer at the specified index.*
- bool [SharedParamsWriteByIndex](#) (uint32\_t idx, uint8\_t value)  
*Writes a data byte to the shared parameter buffer at the specified index.*

### 7.1190.1 Detailed Description

Shared RAM parameters header file.

## 7.1190.2 Function Documentation

### 7.1190.2.1 SharedParamsInit()

```
void SharedParamsInit (
 void)
```

Initializes the shared RAM parameters module.

#### Returns

none.

### 7.1190.2.2 SharedParamsReadByIndex()

```
bool SharedParamsReadByIndex (
 uint32_t idx,
 uint8_t * value)
```

Reads a data byte from the shared parameter buffer at the specified index.

#### Parameters

|              |                                                                                                              |
|--------------|--------------------------------------------------------------------------------------------------------------|
| <i>idx</i>   | Index into the parameter data array. A valid value is between 0 and (SHARED_PARAMS_CFG_BUFFER_DATA_LEN - 1). |
| <i>value</i> | Pointer to where the read data value is stored.                                                              |

#### Returns

True if successful, false otherwise.

### 7.1190.2.3 SharedParamsWriteByIndex()

```
bool SharedParamsWriteByIndex (
 uint32_t idx,
 uint8_t value)
```

Writes a data byte to the shared parameter buffer at the specified index.

#### Parameters

|              |                                                                                                              |
|--------------|--------------------------------------------------------------------------------------------------------------|
| <i>idx</i>   | Index into the parameter data array. A valid value is between 0 and (SHARED_PARAMS_CFG_BUFFER_DATA_LEN - 1). |
| <i>value</i> | Value to write.                                                                                              |

### Returns

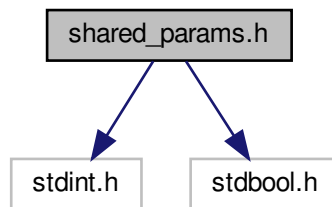
True if successful, false otherwise.

## 7.1191 shared\_params.h File Reference

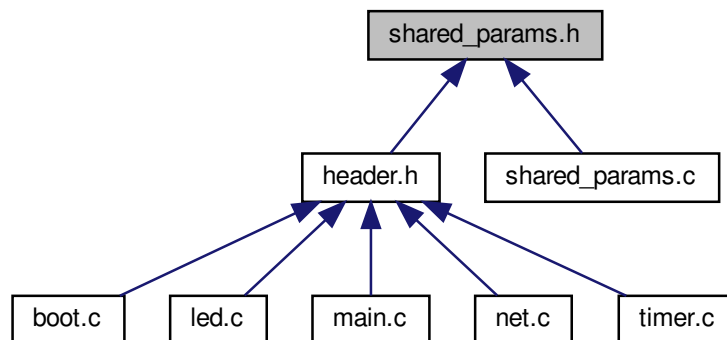
Shared RAM parameters header file.

```
#include <stdint.h>
#include <stdbool.h>
```

Include dependency graph for ARMCM7\_STM32F7\_Nucleo\_F767ZI\_Keil/Prog/shared\_params.h:



This graph shows which files directly or indirectly include this file:



### Macros

- `#define SHARED_PARAMS_CFG_BUFFER_DATA_LEN (56u)`

*Configuration macro for specifying the size of the data inside the parameter buffer. This is the length in bytes of the actual parameter data, so excluding the bufferId and checksum.*

## Functions

- void [SharedParamsInit](#) (void)  
*Initializes the shared RAM parameters module.*
- bool [SharedParamsReadByIndex](#) (uint32\_t idx, uint8\_t \*value)  
*Reads a data byte from the shared parameter buffer at the specified index.*
- bool [SharedParamsWriteByIndex](#) (uint32\_t idx, uint8\_t value)  
*Writes a data byte to the shared parameter buffer at the specified index.*

### 7.1191.1 Detailed Description

Shared RAM parameters header file.

### 7.1191.2 Function Documentation

#### 7.1191.2.1 SharedParamsInit()

```
void SharedParamsInit (
 void)
```

Initializes the shared RAM parameters module.

#### Returns

none.

Referenced by [ApplInit\(\)](#), and [main\(\)](#).

#### 7.1191.2.2 SharedParamsReadByIndex()

```
bool SharedParamsReadByIndex (
 uint32_t idx,
 uint8_t * value)
```

Reads a data byte from the shared parameter buffer at the specified index.

#### Parameters

|              |                                                                                                              |
|--------------|--------------------------------------------------------------------------------------------------------------|
| <i>idx</i>   | Index into the parameter data array. A valid value is between 0 and (SHARED_PARAMS_CFG_BUFFER_DATA_LEN - 1). |
| <i>value</i> | Pointer to where the read data value is stored.                                                              |

**Returns**

True if successful, false otherwise.

Referenced by `ApplInit()`, and `main()`.

**7.1191.2.3 SharedParamsWriteByIndex()**

```
bool SharedParamsWriteByIndex (
 uint32_t idx,
 uint8_t value)
```

Writes a data byte to the shared parameter buffer at the specified index.

**Parameters**

|              |                                                                                                              |
|--------------|--------------------------------------------------------------------------------------------------------------|
| <i>idx</i>   | Index into the parameter data array. A valid value is between 0 and (SHARED_PARAMS_CFG_BUFFER_DATA_LEN - 1). |
| <i>value</i> | Value to write.                                                                                              |

**Returns**

True if successful, false otherwise.

Referenced by `ApplInit()`, `main()`, and `NetApp()`.

**7.1192 time.c File Reference**

Timer driver source file.

```
#include "header.h"
```

Include dependency graph for ARMCM3\_LM3S\_EK\_LM3S6965\_GCC/Prog/time.c:

**Functions**

- void **TimerInit** (void)  
*Initializes the timer.*
- void **TimerDeinit** (void)  
*Stops and disables the timer.*
- void **TimerSet** (unsigned long timer\_value)  
*Sets the initial counter value of the millisecond timer.*
- unsigned long **TimerGet** (void)  
*Obtains the counter value of the millisecond timer.*
- void **TimerISRHandler** (void)  
*Interrupt service routine of the timer.*

## Variables

- static unsigned long [millisecond\\_counter](#)  
*Local variable for storing the number of milliseconds that have elapsed since startup.*

## 7.1192.1 Detailed Description

Timer driver source file.

## 7.1192.2 Function Documentation

### 7.1192.2.1 TimerDeinit()

```
void TimerDeinit (
 void)
```

Stops and disables the timer.

#### Returns

none.

### 7.1192.2.2 TimerGet()

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

#### Returns

Current value of the millisecond timer.

### 7.1192.2.3 TimerInit()

```
void TimerInit (
 void)
```

Initializes the timer.

#### Returns

none.



## 7.1192.2.4 TimerISRHandler()

```
void TimerISRHandler (
 void)
```

Interrupt service routine of the timer.

## Returns

none.

## 7.1192.2.5 TimerSet()

```
void TimerSet (
 unsigned long timer_value)
```

Sets the initial counter value of the millisecond timer.

## Parameters

|                    |                                            |
|--------------------|--------------------------------------------|
| <i>timer_value</i> | initialize value of the millisecond timer. |
|--------------------|--------------------------------------------|

## Returns

none.

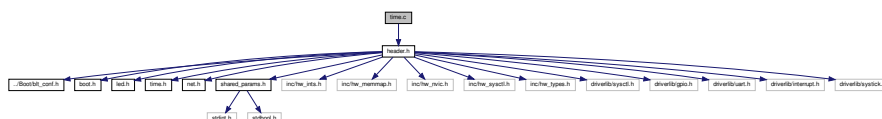
Referenced by TimerInit().

## 7.1193 time.c File Reference

Timer driver source file.

```
#include "header.h"
```

Include dependency graph for ARMCM3\_LM3S\_EK\_LM3S6965\_IAR/Prog/time.c:



## Functions

- void `TimerInit` (void)  
*Initializes the timer.*
- void `TimerDeinit` (void)  
*Stops and disables the timer.*
- void `TimerSet` (unsigned long timer\_value)  
*Sets the initial counter value of the millisecond timer.*
- unsigned long `TimerGet` (void)  
*Obtains the counter value of the millisecond timer.*
- void `TimerISRHandler` (void)  
*Interrupt service routine of the timer.*

## Variables

- static unsigned long `millisecond_counter`  
*Local variable for storing the number of milliseconds that have elapsed since startup.*

### 7.1193.1 Detailed Description

Timer driver source file.

### 7.1193.2 Function Documentation

#### 7.1193.2.1 `TimerDeinit()`

```
void TimerDeinit (
 void)
```

Stops and disables the timer.

#### Returns

none.

#### 7.1193.2.2 `TimerGet()`

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

#### Returns

Current value of the millisecond timer.

### 7.1193.2.3 TimerInit()

```
void TimerInit (
 void)
```

Initializes the timer.

#### Returns

none.

### 7.1193.2.4 TimerISRHandler()

```
void TimerISRHandler (
 void)
```

Interrupt service routine of the timer.

#### Returns

none.

### 7.1193.2.5 TimerSet()

```
void TimerSet (
 unsigned long timer_value)
```

Sets the initial counter value of the millisecond timer.

#### Parameters

|                    |                                            |
|--------------------|--------------------------------------------|
| <i>timer_value</i> | initialize value of the millisecond timer. |
|--------------------|--------------------------------------------|

#### Returns

none.

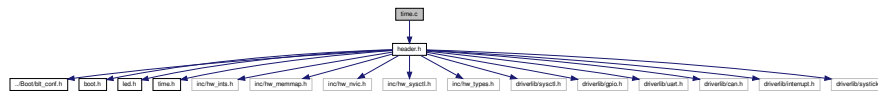
Referenced by TimerInit().

## 7.1194 time.c File Reference

Timer driver source file.

```
#include "header.h"
```

Include dependency graph for ARMCM3\_LM3S\_EK\_LM3S8962\_GCC/Prog/time.c:



## Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- void [TimerDeinit](#) (void)  
*Stops and disables the timer.*
- void [TimerSet](#) (unsigned long timer\_value)  
*Sets the initial counter value of the millisecond timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*
- void [TimerISRHandler](#) (void)  
*Interrupt service routine of the timer.*

## Variables

- static unsigned long [millisecond\\_counter](#)  
*Local variable for storing the number of milliseconds that have elapsed since startup.*

### 7.1194.1 Detailed Description

Timer driver source file.

### 7.1194.2 Function Documentation

#### 7.1194.2.1 TimerDeinit()

```
void TimerDeinit (
 void)
```

Stops and disables the timer.

#### Returns

none.

### 7.1194.2.2 TimerGet()

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

#### Returns

Current value of the millisecond timer.

### 7.1194.2.3 TimerInit()

```
void TimerInit (
 void)
```

Initializes the timer.

#### Returns

none.

### 7.1194.2.4 TimerISRHandler()

```
void TimerISRHandler (
 void)
```

Interrupt service routine of the timer.

#### Returns

none.

### 7.1194.2.5 TimerSet()

```
void TimerSet (
 unsigned long timer_value)
```

Sets the initial counter value of the millisecond timer.

#### Parameters

|                    |                                            |
|--------------------|--------------------------------------------|
| <i>timer_value</i> | initialize value of the millisecond timer. |
|--------------------|--------------------------------------------|



### 7.1195.2.1 TimerDeinit()

```
void TimerDeinit (
 void)
```

Stops and disables the timer.

#### Returns

none.

### 7.1195.2.2 TimerGet()

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

#### Returns

Current value of the millisecond timer.

### 7.1195.2.3 TimerInit()

```
void TimerInit (
 void)
```

Initializes the timer.

#### Returns

none.

### 7.1195.2.4 TimerISRHandler()

```
void TimerISRHandler (
 void)
```

Interrupt service routine of the timer.

#### Returns

none.

### 7.1195.2.5 TimerSet()

```
void TimerSet (
 unsigned long timer_value)
```

Sets the initial counter value of the millisecond timer.





## 7.1196.2 Function Documentation

### 7.1196.2.1 TimerDeinit()

```
void TimerDeinit (
 void)
```

Stops and disables the timer.

#### Returns

none.

### 7.1196.2.2 TimerGet()

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

#### Returns

Current value of the millisecond timer.

### 7.1196.2.3 TimerInit()

```
void TimerInit (
 void)
```

Initializes the timer.

#### Returns

none.

### 7.1196.2.4 TimerISRHandler()

```
void TimerISRHandler (
 void)
```

Interrupt service routine of the timer.

#### Returns

none.

### 7.1196.2.5 TimerSet()

```
void TimerSet (
 unsigned long timer_value)
```

Sets the initial counter value of the millisecond timer.

**Parameters**

|                          |                                            |
|--------------------------|--------------------------------------------|
| <code>timer_value</code> | initialize value of the millisecond timer. |
|--------------------------|--------------------------------------------|

**Returns**

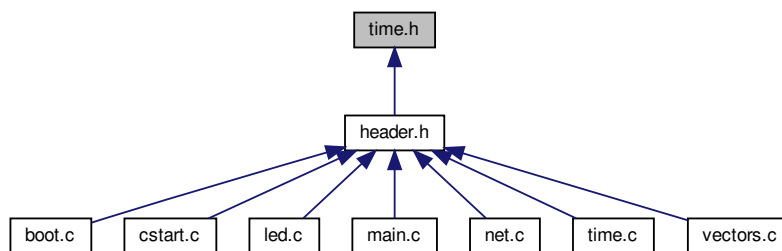
none.

Referenced by `TimerInit()`.

## 7.1197 time.h File Reference

Timer driver header file.

This graph shows which files directly or indirectly include this file:

**Functions**

- void `TimerInit` (void)  
*Initializes the timer.*
- void `TimerDeinit` (void)  
*Stops and disables the timer.*
- void `TimerSet` (unsigned long timer\_value)  
*Sets the initial counter value of the millisecond timer.*
- unsigned long `TimerGet` (void)  
*Obtains the counter value of the millisecond timer.*
- void `TimerISRHandler` (void)  
*Interrupt service routine of the timer.*

### 7.1197.1 Detailed Description

Timer driver header file.

## 7.1197.2 Function Documentation

### 7.1197.2.1 TimerDeinit()

```
void TimerDeinit (
 void)
```

Stops and disables the timer.

#### Returns

none.

### 7.1197.2.2 TimerGet()

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

#### Returns

Current value of the millisecond timer.

### 7.1197.2.3 TimerInit()

```
void TimerInit (
 void)
```

Initializes the timer.

#### Returns

none.

Initializes the timer.

#### Returns

none.

#### 7.1197.2.4 TimerISRHandler()

```
void TimerISRHandler (
 void)
```

Interrupt service routine of the timer.

##### Returns

none.

#### 7.1197.2.5 TimerSet()

```
void TimerSet (
 unsigned long timer_value)
```

Sets the initial counter value of the millisecond timer.

##### Parameters

|                    |                                            |
|--------------------|--------------------------------------------|
| <i>timer_value</i> | initialize value of the millisecond timer. |
|--------------------|--------------------------------------------|

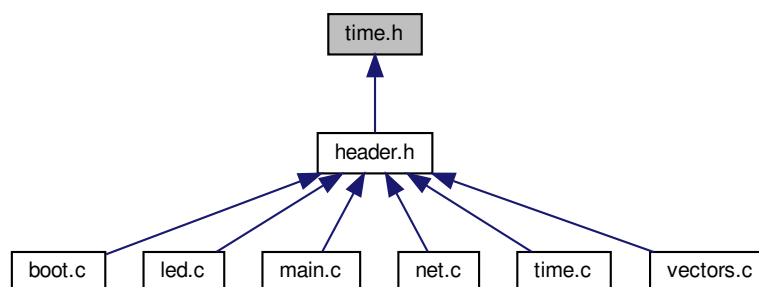
##### Returns

none.

## 7.1198 time.h File Reference

Timer driver header file.

This graph shows which files directly or indirectly include this file:



## Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- void [TimerDeinit](#) (void)  
*Stops and disables the timer.*
- void [TimerSet](#) (unsigned long timer\_value)  
*Sets the initial counter value of the millisecond timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*
- void [TimerISRHandler](#) (void)  
*Interrupt service routine of the timer.*

### 7.1198.1 Detailed Description

Timer driver header file.

### 7.1198.2 Function Documentation

#### 7.1198.2.1 [TimerDeinit\(\)](#)

```
void TimerDeinit (
 void)
```

Stops and disables the timer.

##### Returns

none.

#### 7.1198.2.2 [TimerGet\(\)](#)

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

##### Returns

Current value of the millisecond timer.

### 7.1198.2.3 TimerInit()

```
void TimerInit (
 void)
```

Initializes the timer.

#### Returns

none.

Initializes the timer.

#### Returns

none.

### 7.1198.2.4 TimerISRHandler()

```
void TimerISRHandler (
 void)
```

Interrupt service routine of the timer.

#### Returns

none.

### 7.1198.2.5 TimerSet()

```
void TimerSet (
 unsigned long timer_value)
```

Sets the initial counter value of the millisecond timer.

#### Parameters

|                    |                                            |
|--------------------|--------------------------------------------|
| <i>timer_value</i> | initialize value of the millisecond timer. |
|--------------------|--------------------------------------------|

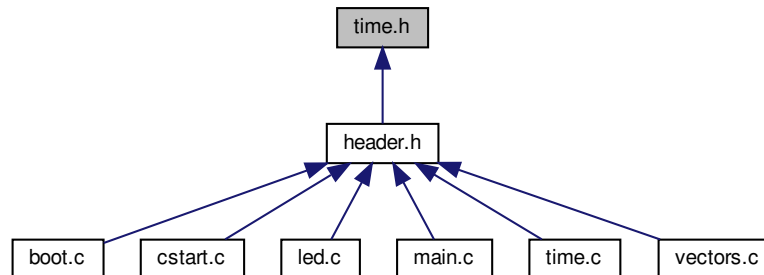
#### Returns

none.

## 7.1199 time.h File Reference

Timer driver header file.

This graph shows which files directly or indirectly include this file:



### Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- void [TimerDeinit](#) (void)  
*Stops and disables the timer.*
- void [TimerSet](#) (unsigned long timer\_value)  
*Sets the initial counter value of the millisecond timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*
- void [TimerISRHandler](#) (void)  
*Interrupt service routine of the timer.*

### 7.1199.1 Detailed Description

Timer driver header file.

### 7.1199.2 Function Documentation

#### 7.1199.2.1 TimerDeinit()

```
void TimerDeinit (
 void)
```

Stops and disables the timer.

#### Returns

none.

#### 7.1199.2.2 TimerGet()

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

##### Returns

Current value of the millisecond timer.

#### 7.1199.2.3 TimerInit()

```
void TimerInit (
 void)
```

Initializes the timer.

##### Returns

none.

Initializes the timer.

##### Returns

none.

#### 7.1199.2.4 TimerISRHandler()

```
void TimerISRHandler (
 void)
```

Interrupt service routine of the timer.

##### Returns

none.

#### 7.1199.2.5 TimerSet()

```
void TimerSet (
 unsigned long timer_value)
```

Sets the initial counter value of the millisecond timer.



## Parameters

|                          |                                            |
|--------------------------|--------------------------------------------|
| <code>timer_value</code> | initialize value of the millisecond timer. |
|--------------------------|--------------------------------------------|

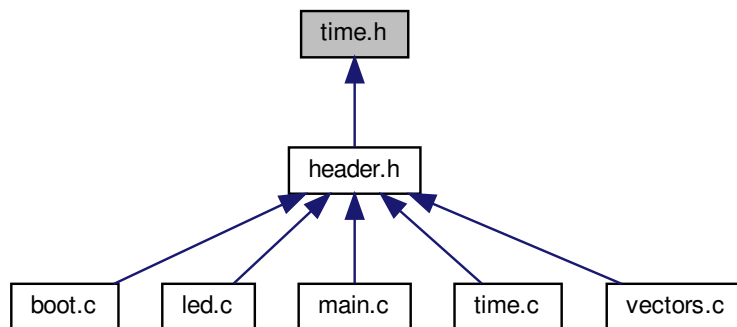
## Returns

none.

## 7.1200 time.h File Reference

Timer driver header file.

This graph shows which files directly or indirectly include this file:



## Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- void [TimerDeinit](#) (void)  
*Stops and disables the timer.*
- void [TimerSet](#) (unsigned long timer\_value)  
*Sets the initial counter value of the millisecond timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*
- void [TimerISRHandler](#) (void)  
*Interrupt service routine of the timer.*

### 7.1200.1 Detailed Description

Timer driver header file.

## 7.1200.2 Function Documentation

### 7.1200.2.1 TimerDeinit()

```
void TimerDeinit (
 void)
```

Stops and disables the timer.

#### Returns

none.

### 7.1200.2.2 TimerGet()

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

#### Returns

Current value of the millisecond timer.

### 7.1200.2.3 TimerInit()

```
void TimerInit (
 void)
```

Initializes the timer.

#### Returns

none.

Initializes the timer.

#### Returns

none.

#### 7.1200.2.4 TimerISRHandler()

```
void TimerISRHandler (
 void)
```

Interrupt service routine of the timer.

##### Returns

none.

#### 7.1200.2.5 TimerSet()

```
void TimerSet (
 unsigned long timer_value)
```

Sets the initial counter value of the millisecond timer.

##### Parameters

|                    |                                            |
|--------------------|--------------------------------------------|
| <i>timer_value</i> | initialize value of the millisecond timer. |
|--------------------|--------------------------------------------|

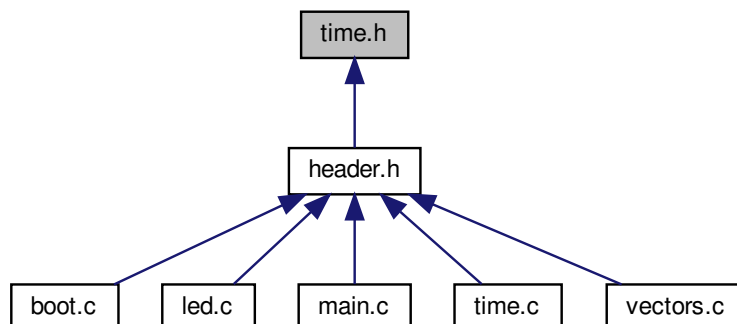
##### Returns

none.

## 7.1201 time.h File Reference

Timer driver header file.

This graph shows which files directly or indirectly include this file:



## Functions

- void `TimerInit` (void)  
*Initializes the timer.*
- void `TimerDeinit` (void)  
*Stops and disables the timer.*
- void `TimerSet` (unsigned long timer\_value)  
*Sets the initial counter value of the millisecond timer.*
- unsigned long `TimerGet` (void)  
*Obtains the counter value of the millisecond timer.*
- void `TimerISRHandler` (void)  
*Interrupt service routine of the timer.*

### 7.1201.1 Detailed Description

Timer driver header file.

### 7.1201.2 Function Documentation

#### 7.1201.2.1 `TimerDeinit()`

```
void TimerDeinit (
 void)
```

Stops and disables the timer.

#### Returns

none.

#### 7.1201.2.2 `TimerGet()`

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

#### Returns

Current value of the millisecond timer.

### 7.1201.2.3 TimerInit()

```
void TimerInit (
 void)
```

Initializes the timer.

#### Returns

none.

Initializes the timer.

#### Returns

none.

### 7.1201.2.4 TimerISRHandler()

```
void TimerISRHandler (
 void)
```

Interrupt service routine of the timer.

#### Returns

none.

### 7.1201.2.5 TimerSet()

```
void TimerSet (
 unsigned long timer_value)
```

Sets the initial counter value of the millisecond timer.

#### Parameters

|                    |                                            |
|--------------------|--------------------------------------------|
| <i>timer_value</i> | initialize value of the millisecond timer. |
|--------------------|--------------------------------------------|

#### Returns

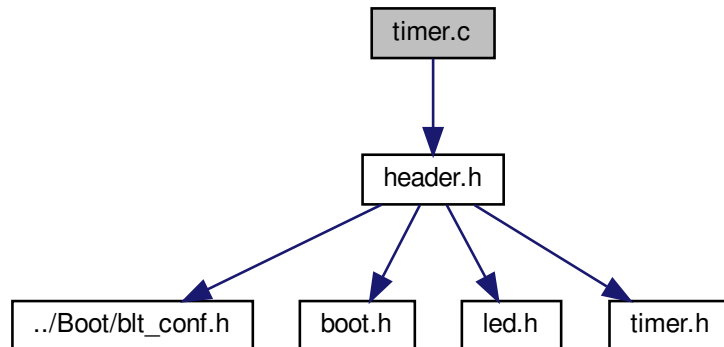
none.

## 7.1202 timer.c File Reference

Timer driver source file.

```
#include "header.h"
```

Include dependency graph for Demo/\_template/Prog/timer.c:



### Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*
- void [TimerInterrupt](#) (void)  
*Interrupt service routine of the timer.*

### Variables

- static unsigned long [millisecond\\_counter](#)  
*Local variable for storing the number of milliseconds that have elapsed since startup.*

#### 7.1202.1 Detailed Description

Timer driver source file.

#### 7.1202.2 Function Documentation

### 7.1202.2.1 TimerGet()

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

#### Returns

Current value of the millisecond timer.

Referenced by BackDoorCheck(), BootComCanInit(), BootComRs232CheckActivationRequest(), CanDisabled↵  
ModeEnter(), CanDisabledModeExit(), CanFreezeModeEnter(), CanFreezeModeExit(), CanInit(), CanTransmit↵  
Packet(), FileFirmwareUpdateCompletedHook(), FileFirmwareUpdateLogHook(), FlashEraseSectors(), Flash↵  
WriteBlock(), LedBlinkTask(), LedToggle(), NetInit(), NetServerTask(), and NetTask().

### 7.1202.2.2 TimerInit()

```
void TimerInit (
 void)
```

Initializes the timer.

#### Returns

none.

Referenced by ApplInit(), BootInit(), and Init().

### 7.1202.2.3 TimerInterrupt()

```
void TimerInterrupt (
 void)
```

Interrupt service routine of the timer.

#### Returns

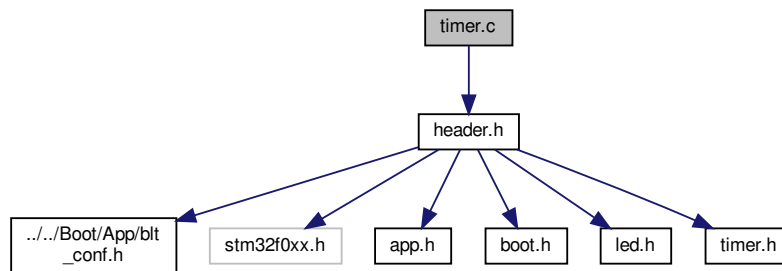
none.

## 7.1203 timer.c File Reference

Timer driver source file.

```
#include "header.h"
```

Include dependency graph for Demo/ARMCM0\_STM32F0\_Discovery\_STM32F051\_CubeIDE/Prog/App/timer.c:



### Functions

- void `TimerInit` (void)  
*Initializes the timer.*
- unsigned long `TimerGet` (void)  
*Obtains the counter value of the millisecond timer.*

### 7.1203.1 Detailed Description

Timer driver source file.

### 7.1203.2 Function Documentation

#### 7.1203.2.1 TimerGet()

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

#### Returns

Current value of the millisecond timer.



### 7.1203.2.2 TimerInit()

```
void TimerInit (
 void)
```

Initializes the timer.

#### Returns

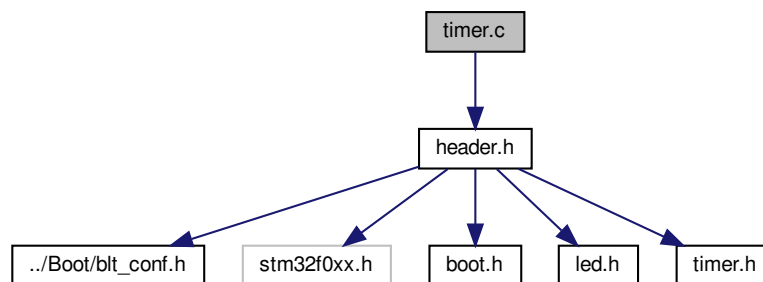
none.

## 7.1204 timer.c File Reference

Timer driver source file.

```
#include "header.h"
```

Include dependency graph for Demo/ARMCM0\_STM32F0\_Discovery\_STM32F051\_GCC/Prog/timer.c:



## Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*
- void [SysTick\\_Handler](#) (void)  
*Interrupt service routine of the timer.*

### 7.1204.1 Detailed Description

Timer driver source file.

### 7.1204.2 Function Documentation

### 7.1204.2.1 SysTick\_Handler()

```
void SysTick_Handler (
 void)
```

Interrupt service routine of the timer.

#### Returns

none.

### 7.1204.2.2 TimerGet()

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

#### Returns

Current value of the millisecond timer.

### 7.1204.2.3 TimerInit()

```
void TimerInit (
 void)
```

Initializes the timer.

#### Returns

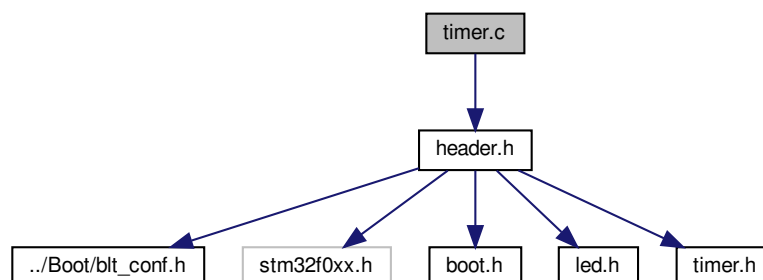
none.

## 7.1205 timer.c File Reference

Timer driver source file.

```
#include "header.h"
```

Include dependency graph for Demo/ARMCM0\_STM32F0\_Discovery\_STM32F051\_IAR/Prog/timer.c:



## Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*
- void [SysTick\\_Handler](#) (void)  
*Interrupt service routine of the timer.*

### 7.1205.1 Detailed Description

Timer driver source file.

### 7.1205.2 Function Documentation

#### 7.1205.2.1 SysTick\_Handler()

```
void SysTick_Handler (
 void)
```

Interrupt service routine of the timer.

#### Returns

none.

#### 7.1205.2.2 TimerGet()

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

#### Returns

Current value of the millisecond timer.

### 7.1205.2.3 TimerInit()

```
void TimerInit (
 void)
```

Initializes the timer.

#### Returns

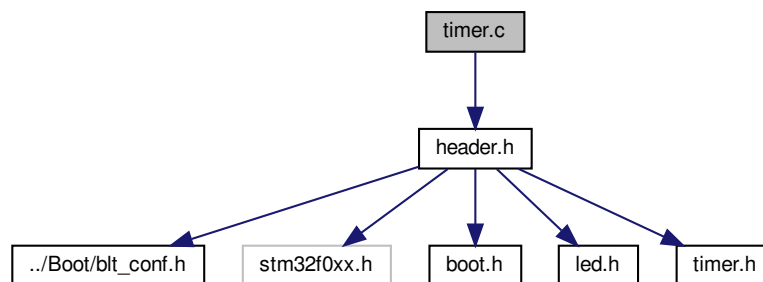
none.

## 7.1206 timer.c File Reference

Timer driver source file.

```
#include "header.h"
```

Include dependency graph for Demo/ARMCM0\_STM32F0\_Discovery\_STM32F051\_Keil/Prog/timer.c:



## Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*
- void [SysTick\\_Handler](#) (void)  
*Interrupt service routine of the timer.*

### 7.1206.1 Detailed Description

Timer driver source file.

### 7.1206.2 Function Documentation

### 7.1206.2.1 SysTick\_Handler()

```
void SysTick_Handler (
 void)
```

Interrupt service routine of the timer.

#### Returns

none.

### 7.1206.2.2 TimerGet()

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

#### Returns

Current value of the millisecond timer.

### 7.1206.2.3 TimerInit()

```
void TimerInit (
 void)
```

Initializes the timer.

#### Returns

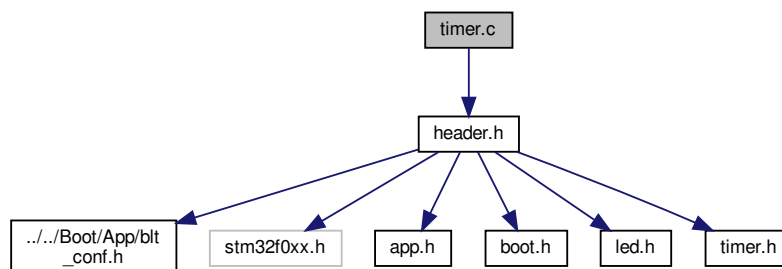
none.

## 7.1207 timer.c File Reference

Timer driver source file.

```
#include "header.h"
```

Include dependency graph for Demo/ARMCM0\_STM32F0\_Nucleo\_F091RC\_CubeIDE/Prog/App/timer.c:



## Functions

- void `TimerInit` (void)  
*Initializes the timer.*
- unsigned long `TimerGet` (void)  
*Obtains the counter value of the millisecond timer.*

### 7.1207.1 Detailed Description

Timer driver source file.

### 7.1207.2 Function Documentation

#### 7.1207.2.1 `TimerGet()`

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

#### Returns

Current value of the millisecond timer.

#### 7.1207.2.2 `TimerInit()`

```
void TimerInit (
 void)
```

Initializes the timer.

#### Returns

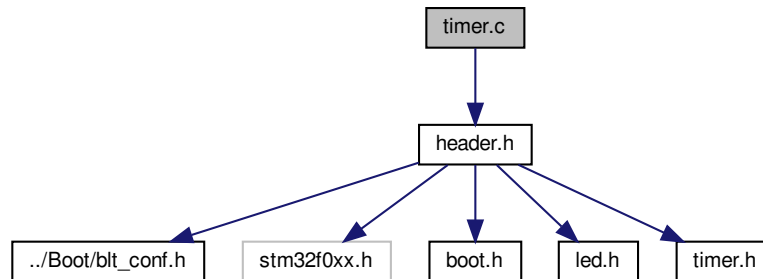
none.

## 7.1208 timer.c File Reference

Timer driver source file.

```
#include "header.h"
```

Include dependency graph for Demo/ARMCM0\_STM32F0\_Nucleo\_F091RC\_GCC/Prog/timer.c:



### Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*
- void [SysTick\\_Handler](#) (void)  
*Interrupt service routine of the timer.*

### 7.1208.1 Detailed Description

Timer driver source file.

### 7.1208.2 Function Documentation

#### 7.1208.2.1 SysTick\_Handler()

```
void SysTick_Handler (
 void)
```

Interrupt service routine of the timer.

#### Returns

none.

### 7.1208.2.2 TimerGet()

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

#### Returns

Current value of the millisecond timer.

### 7.1208.2.3 TimerInit()

```
void TimerInit (
 void)
```

Initializes the timer.

#### Returns

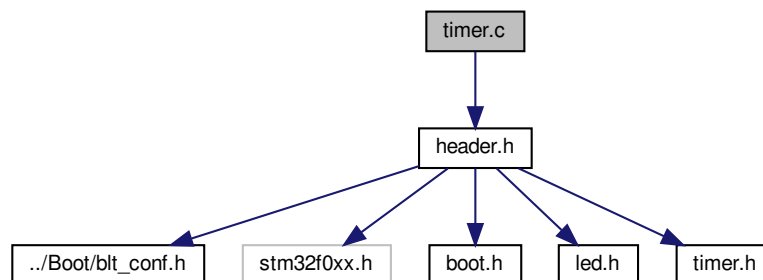
none.

## 7.1209 timer.c File Reference

Timer driver source file.

```
#include "header.h"
```

Include dependency graph for Demo/ARMCM0\_STM32F0\_Nucleo\_F091RC\_IAR/Prog/timer.c:



## Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*
- void [SysTick\\_Handler](#) (void)  
*Interrupt service routine of the timer.*



## 7.1209.1 Detailed Description

Timer driver source file.

## 7.1209.2 Function Documentation

### 7.1209.2.1 SysTick\_Handler()

```
void SysTick_Handler (
 void)
```

Interrupt service routine of the timer.

#### Returns

none.

### 7.1209.2.2 TimerGet()

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

#### Returns

Current value of the millisecond timer.

### 7.1209.2.3 TimerInit()

```
void TimerInit (
 void)
```

Initializes the timer.

#### Returns

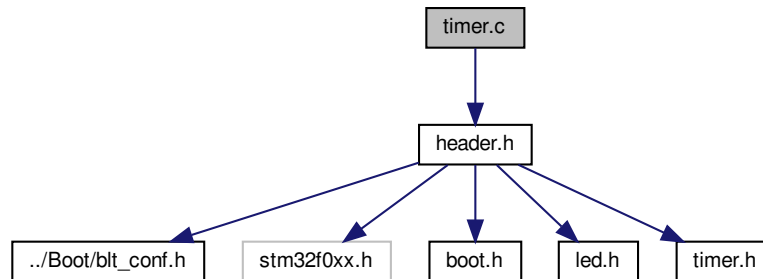
none.

## 7.1210 timer.c File Reference

Timer driver source file.

```
#include "header.h"
```

Include dependency graph for Demo/ARMCM0\_STM32F0\_Nucleo\_F091RC\_Keil/Prog/timer.c:



### Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*
- void [SysTick\\_Handler](#) (void)  
*Interrupt service routine of the timer.*

### 7.1210.1 Detailed Description

Timer driver source file.

### 7.1210.2 Function Documentation

#### 7.1210.2.1 SysTick\_Handler()

```
void SysTick_Handler (
 void)
```

Interrupt service routine of the timer.

#### Returns

none.

### 7.1210.2.2 TimerGet()

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

#### Returns

Current value of the millisecond timer.

### 7.1210.2.3 TimerInit()

```
void TimerInit (
 void)
```

Initializes the timer.

#### Returns

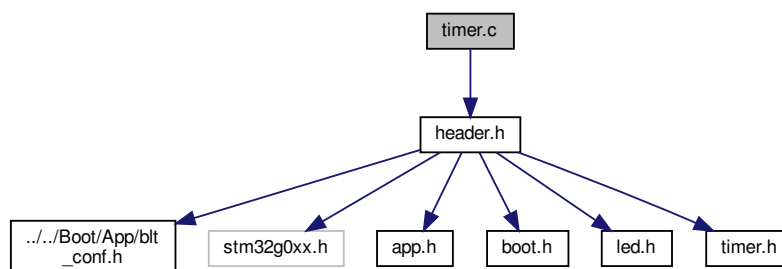
none.

## 7.1211 timer.c File Reference

Timer driver source file.

```
#include "header.h"
```

Include dependency graph for Demo/ARMCM0\_STM32G0\_Nucleo\_G071RB\_CubeIDE/Prog/App/timer.c:



## Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*

### 7.1211.1 Detailed Description

Timer driver source file.

### 7.1211.2 Function Documentation

#### 7.1211.2.1 TimerGet()

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

##### Returns

Current value of the millisecond timer.

#### 7.1211.2.2 TimerInit()

```
void TimerInit (
 void)
```

Initializes the timer.

##### Returns

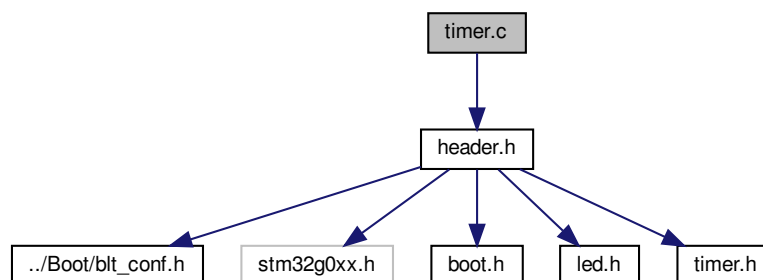
none.

## 7.1212 timer.c File Reference

Timer driver source file.

```
#include "header.h"
```

Include dependency graph for Demo/ARMCM0\_STM32G0\_Nucleo\_G071RB\_GCC/Prog/timer.c:



## Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*
- void [SysTick\\_Handler](#) (void)  
*Interrupt service routine of the timer.*

### 7.1212.1 Detailed Description

Timer driver source file.

### 7.1212.2 Function Documentation

#### 7.1212.2.1 SysTick\_Handler()

```
void SysTick_Handler (
 void)
```

Interrupt service routine of the timer.

#### Returns

none.

#### 7.1212.2.2 TimerGet()

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

#### Returns

Current value of the millisecond timer.

### 7.1212.2.3 TimerInit()

```
void TimerInit (
 void)
```

Initializes the timer.

#### Returns

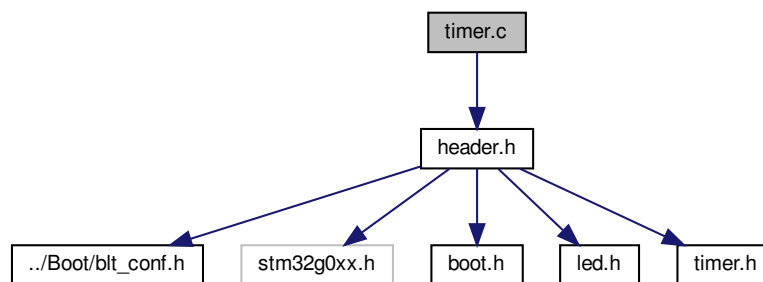
none.

## 7.1213 timer.c File Reference

Timer driver source file.

```
#include "header.h"
```

Include dependency graph for Demo/ARMCM0\_STM32G0\_Nucleo\_G071RB\_IAR/Prog/timer.c:



## Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*
- void [SysTick\\_Handler](#) (void)  
*Interrupt service routine of the timer.*

### 7.1213.1 Detailed Description

Timer driver source file.

### 7.1213.2 Function Documentation

### 7.1213.2.1 SysTick\_Handler()

```
void SysTick_Handler (
 void)
```

Interrupt service routine of the timer.

#### Returns

none.

### 7.1213.2.2 TimerGet()

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

#### Returns

Current value of the millisecond timer.

### 7.1213.2.3 TimerInit()

```
void TimerInit (
 void)
```

Initializes the timer.

#### Returns

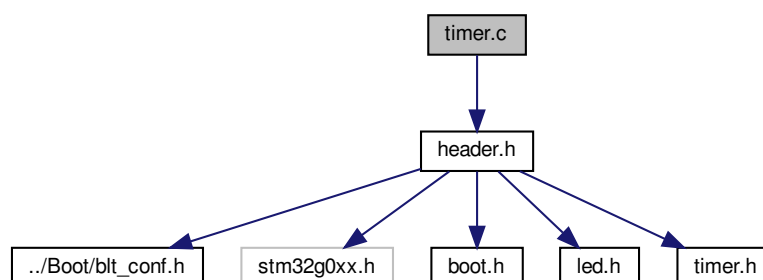
none.

## 7.1214 timer.c File Reference

Timer driver source file.

```
#include "header.h"
```

Include dependency graph for Demo/ARMCM0\_STM32G0\_Nucleo\_G071RB\_Keil/Prog/timer.c:



## Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*
- void [SysTick\\_Handler](#) (void)  
*Interrupt service routine of the timer.*

### 7.1214.1 Detailed Description

Timer driver source file.

### 7.1214.2 Function Documentation

#### 7.1214.2.1 SysTick\_Handler()

```
void SysTick_Handler (
 void)
```

Interrupt service routine of the timer.

#### Returns

none.

#### 7.1214.2.2 TimerGet()

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

#### Returns

Current value of the millisecond timer.



## 7.1214.2.3 TimerInit()

```
void TimerInit (
 void)
```

Initializes the timer.

## Returns

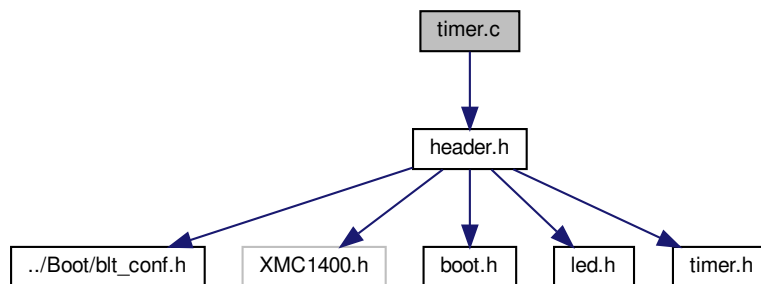
none.

## 7.1215 timer.c File Reference

Timer driver source file.

```
#include "header.h"
```

Include dependency graph for Demo/ARMCM0\_XMC1\_XMC1400\_Boot\_Kit\_GCC/Prog/timer.c:



## Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- void [TimerDeinit](#) (void)  
*Stops and disables the timer.*
- void [TimerSet](#) (unsigned long timer\_value)  
*Sets the initial counter value of the millisecond timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*
- void [SysTick\\_Handler](#) (void)  
*Interrupt service routine of the timer.*

## Variables

- static unsigned long [millisecond\\_counter](#)  
*Local variable for storing the number of milliseconds that have elapsed since startup.*

### 7.1215.1 Detailed Description

Timer driver source file.

### 7.1215.2 Function Documentation

#### 7.1215.2.1 SysTick\_Handler()

```
void SysTick_Handler (
 void)
```

Interrupt service routine of the timer.

##### Returns

none.

#### 7.1215.2.2 TimerDeinit()

```
void TimerDeinit (
 void)
```

Stops and disables the timer.

##### Returns

none.

#### 7.1215.2.3 TimerGet()

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

##### Returns

Current value of the millisecond timer.

#### 7.1215.2.4 TimerInit()

```
void TimerInit (
 void)
```

Initializes the timer.

##### Returns

none.

#### 7.1215.2.5 TimerSet()

```
void TimerSet (
 unsigned long timer_value)
```

Sets the initial counter value of the millisecond timer.

##### Parameters

|                    |                                            |
|--------------------|--------------------------------------------|
| <i>timer_value</i> | initialize value of the millisecond timer. |
|--------------------|--------------------------------------------|

##### Returns

none.

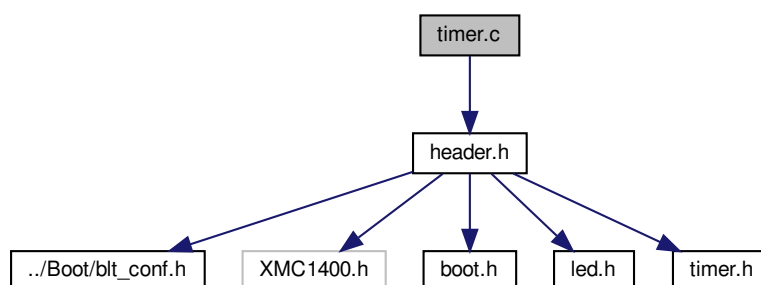
Referenced by TimerInit().

## 7.1216 timer.c File Reference

Timer driver source file.

```
#include "header.h"
```

Include dependency graph for Demo/ARMCM0\_XMC1\_XMC1400\_Boot\_Kit\_IAR/Prog/timer.c:



## Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- void [TimerDeinit](#) (void)  
*Stops and disables the timer.*
- void [TimerSet](#) (unsigned long timer\_value)  
*Sets the initial counter value of the millisecond timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*
- void [SysTick\\_Handler](#) (void)  
*Interrupt service routine of the timer.*

## Variables

- static unsigned long [millisecond\\_counter](#)  
*Local variable for storing the number of milliseconds that have elapsed since startup.*

### 7.1216.1 Detailed Description

Timer driver source file.

### 7.1216.2 Function Documentation

#### 7.1216.2.1 SysTick\_Handler()

```
void SysTick_Handler (
 void)
```

Interrupt service routine of the timer.

#### Returns

none.

#### 7.1216.2.2 TimerDeinit()

```
void TimerDeinit (
 void)
```

Stops and disables the timer.

#### Returns

none.

### 7.1216.2.3 TimerGet()

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

#### Returns

Current value of the millisecond timer.

### 7.1216.2.4 TimerInit()

```
void TimerInit (
 void)
```

Initializes the timer.

#### Returns

none.

### 7.1216.2.5 TimerSet()

```
void TimerSet (
 unsigned long timer_value)
```

Sets the initial counter value of the millisecond timer.

#### Parameters

|                    |                                            |
|--------------------|--------------------------------------------|
| <i>timer_value</i> | initialize value of the millisecond timer. |
|--------------------|--------------------------------------------|

#### Returns

none.

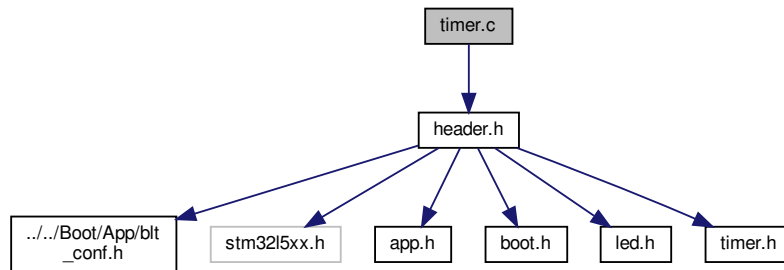
Referenced by TimerInit().

## 7.1217 timer.c File Reference

Timer driver source file.

```
#include "header.h"
```

Include dependency graph for Demo/ARMCM33\_STM32L5\_Nucleo\_L552ZE\_CubeIDE/Prog/App/timer.c:



## Functions

- void `TimerInit` (void)  
*Initializes the timer.*
- unsigned long `TimerGet` (void)  
*Obtains the counter value of the millisecond timer.*

### 7.1217.1 Detailed Description

Timer driver source file.

### 7.1217.2 Function Documentation

#### 7.1217.2.1 `TimerGet()`

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

#### Returns

Current value of the millisecond timer.

### 7.1217.2.2 TimerInit()

```
void TimerInit (
 void)
```

Initializes the timer.

#### Returns

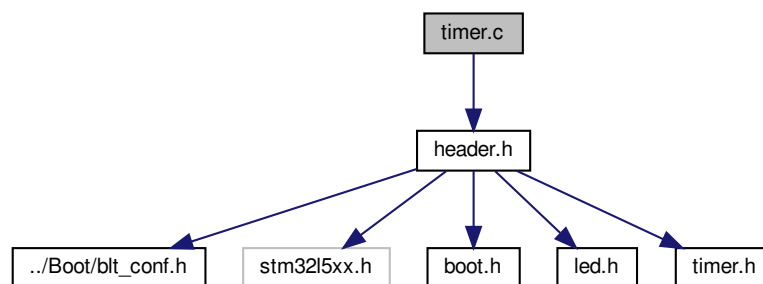
none.

## 7.1218 timer.c File Reference

Timer driver source file.

```
#include "header.h"
```

Include dependency graph for Demo/ARMCM33\_STM32L5\_Nucleo\_L552ZE\_GCC/Prog/timer.c:



## Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*
- void [SysTick\\_Handler](#) (void)  
*Interrupt service routine of the timer.*

### 7.1218.1 Detailed Description

Timer driver source file.

### 7.1218.2 Function Documentation

### 7.1218.2.1 SysTick\_Handler()

```
void SysTick_Handler (
 void)
```

Interrupt service routine of the timer.

#### Returns

none.

### 7.1218.2.2 TimerGet()

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

#### Returns

Current value of the millisecond timer.

### 7.1218.2.3 TimerInit()

```
void TimerInit (
 void)
```

Initializes the timer.

#### Returns

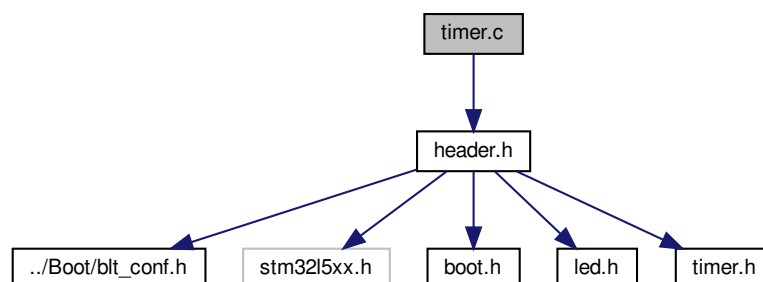
none.

## 7.1219 timer.c File Reference

Timer driver source file.

```
#include "header.h"
```

Include dependency graph for Demo/ARMCM33\_STM32L5\_Nucleo\_L552ZE\_IAR/Prog/timer.c:





## Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*
- void [SysTick\\_Handler](#) (void)  
*Interrupt service routine of the timer.*

### 7.1219.1 Detailed Description

Timer driver source file.

### 7.1219.2 Function Documentation

#### 7.1219.2.1 SysTick\_Handler()

```
void SysTick_Handler (
 void)
```

Interrupt service routine of the timer.

#### Returns

none.

#### 7.1219.2.2 TimerGet()

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

#### Returns

Current value of the millisecond timer.

### 7.1219.2.3 TimerInit()

```
void TimerInit (
 void)
```

Initializes the timer.

#### Returns

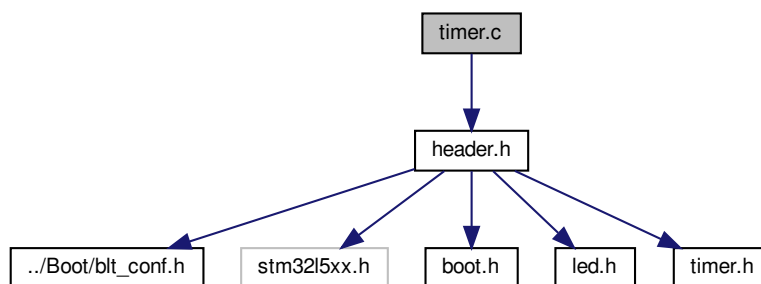
none.

## 7.1220 timer.c File Reference

Timer driver source file.

```
#include "header.h"
```

Include dependency graph for Demo/ARMCM33\_STM32L5\_Nucleo\_L552ZE\_Keil/Prog/timer.c:



## Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*
- void [SysTick\\_Handler](#) (void)  
*Interrupt service routine of the timer.*

### 7.1220.1 Detailed Description

Timer driver source file.

### 7.1220.2 Function Documentation

## 7.1220.2.1 SysTick\_Handler()

```
void SysTick_Handler (
 void)
```

Interrupt service routine of the timer.

## Returns

none.

## 7.1220.2.2 TimerGet()

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

## Returns

Current value of the millisecond timer.

## 7.1220.2.3 TimerInit()

```
void TimerInit (
 void)
```

Initializes the timer.

## Returns

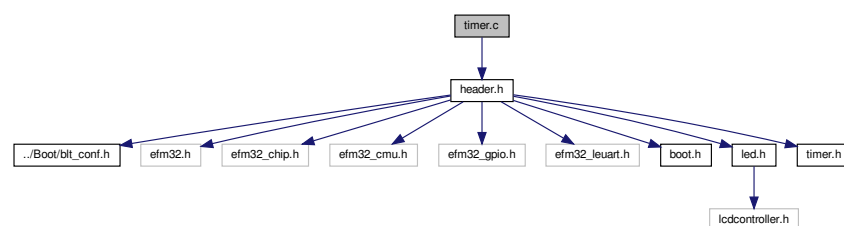
none.

## 7.1221 timer.c File Reference

Timer driver source file.

```
#include "header.h"
```

Include dependency graph for Demo/ARMCM3\_EFM32\_Olimex\_EM32G880F128STK\_GCC/Prog/timer.c:



## Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- void [TimerDeinit](#) (void)  
*Stops and disables the timer.*
- void [TimerSet](#) (unsigned long timer\_value)  
*Sets the initial counter value of the millisecond timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*
- void [TimerISRHandler](#) (void)  
*Interrupt service routine of the timer.*

## Variables

- static unsigned long [millisecond\\_counter](#)  
*Local variable for storing the number of milliseconds that have elapsed since startup.*

### 7.1221.1 Detailed Description

Timer driver source file.

### 7.1221.2 Function Documentation

#### 7.1221.2.1 [TimerDeinit\(\)](#)

```
void TimerDeinit (
 void)
```

Stops and disables the timer.

#### Returns

none.

#### 7.1221.2.2 [TimerGet\(\)](#)

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

#### Returns

Current value of the millisecond timer.

### 7.1221.2.3 TimerInit()

```
void TimerInit (
 void)
```

Initializes the timer.

#### Returns

none.

### 7.1221.2.4 TimerISRHandler()

```
void TimerISRHandler (
 void)
```

Interrupt service routine of the timer.

#### Returns

none.

Referenced by `__attribute__()`.

### 7.1221.2.5 TimerSet()

```
void TimerSet (
 unsigned long timer_value)
```

Sets the initial counter value of the millisecond timer.

#### Parameters

|                    |                                            |
|--------------------|--------------------------------------------|
| <i>timer_value</i> | initialize value of the millisecond timer. |
|--------------------|--------------------------------------------|

#### Returns

none.

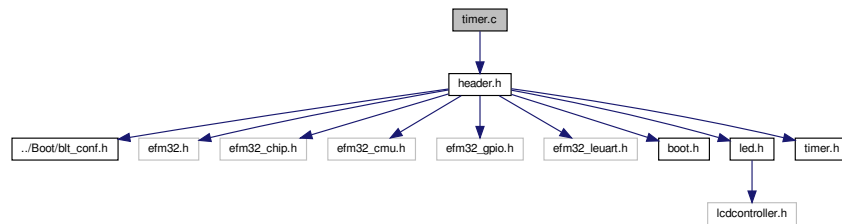
Referenced by `TimerInit()`.

## 7.1222 timer.c File Reference

Timer driver source file.

```
#include "header.h"
```

Include dependency graph for Demo/ARMCM3\_EFM32\_Olimex\_EM32G880F128STK\_IAR/Prog/timer.c:



## Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- void [TimerDeinit](#) (void)  
*Stops and disables the timer.*
- void [TimerSet](#) (unsigned long timer\_value)  
*Sets the initial counter value of the millisecond timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*
- void [TimerISRHandler](#) (void)  
*Interrupt service routine of the timer.*

## Variables

- static unsigned long [millisecond\\_counter](#)  
*Local variable for storing the number of milliseconds that have elapsed since startup.*

### 7.1222.1 Detailed Description

Timer driver source file.

### 7.1222.2 Function Documentation

#### 7.1222.2.1 TimerDeinit()

```
void TimerDeinit (
 void)
```

Stops and disables the timer.

#### Returns

none.

### 7.1222.2.2 TimerGet()

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

#### Returns

Current value of the millisecond timer.

### 7.1222.2.3 TimerInit()

```
void TimerInit (
 void)
```

Initializes the timer.

#### Returns

none.

### 7.1222.2.4 TimerISRHandler()

```
void TimerISRHandler (
 void)
```

Interrupt service routine of the timer.

#### Returns

none.

### 7.1222.2.5 TimerSet()

```
void TimerSet (
 unsigned long timer_value)
```

Sets the initial counter value of the millisecond timer.

#### Parameters

|                    |                                            |
|--------------------|--------------------------------------------|
| <i>timer_value</i> | initialize value of the millisecond timer. |
|--------------------|--------------------------------------------|

**Returns**

none.

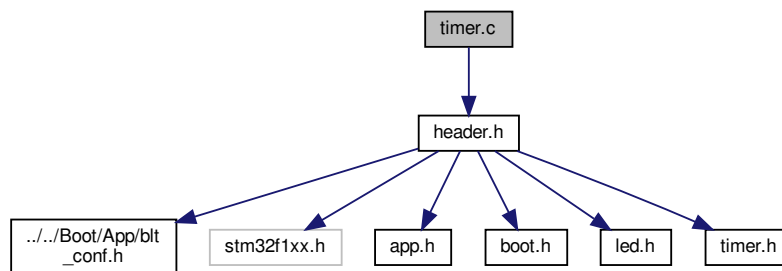
Referenced by TimerInit().

## 7.1223 timer.c File Reference

Timer driver source file.

```
#include "header.h"
```

Include dependency graph for Demo/ARMCM3\_STM32F1\_Nucleo\_F103RB\_CubeIDE/Prog/App/timer.c:

**Functions**

- void [TimerInit](#) (void)  
*Initializes the timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*

### 7.1223.1 Detailed Description

Timer driver source file.

### 7.1223.2 Function Documentation

#### 7.1223.2.1 TimerGet()

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

**Returns**

Current value of the millisecond timer.



### 7.1223.2.2 TimerInit()

```
void TimerInit (
 void)
```

Initializes the timer.

#### Returns

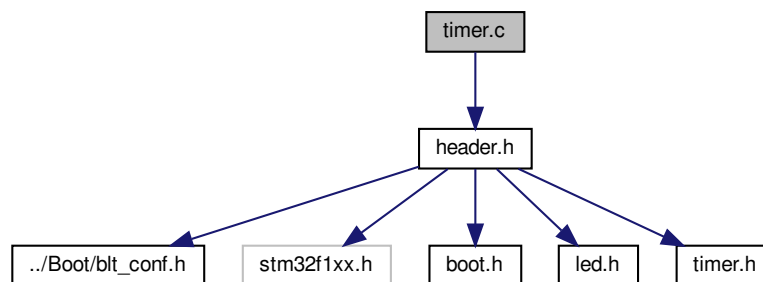
none.

## 7.1224 timer.c File Reference

Timer driver source file.

```
#include "header.h"
```

Include dependency graph for Demo/ARMCM3\_STM32F1\_Nucleo\_F103RB\_GCC/Prog/timer.c:



## Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*
- void [SysTick\\_Handler](#) (void)  
*Interrupt service routine of the timer.*

### 7.1224.1 Detailed Description

Timer driver source file.

### 7.1224.2 Function Documentation

### 7.1224.2.1 SysTick\_Handler()

```
void SysTick_Handler (
 void)
```

Interrupt service routine of the timer.

#### Returns

none.

### 7.1224.2.2 TimerGet()

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

#### Returns

Current value of the millisecond timer.

### 7.1224.2.3 TimerInit()

```
void TimerInit (
 void)
```

Initializes the timer.

#### Returns

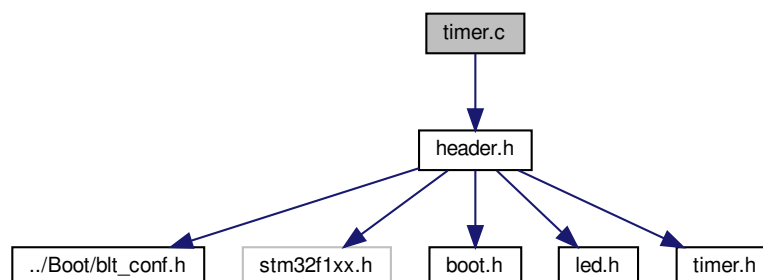
none.

## 7.1225 timer.c File Reference

Timer driver source file.

```
#include "header.h"
```

Include dependency graph for Demo/ARMCM3\_STM32F1\_Nucleo\_F103RB\_IAR/Prog/timer.c:



## Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*
- void [SysTick\\_Handler](#) (void)  
*Interrupt service routine of the timer.*

### 7.1225.1 Detailed Description

Timer driver source file.

### 7.1225.2 Function Documentation

#### 7.1225.2.1 SysTick\_Handler()

```
void SysTick_Handler (
 void)
```

Interrupt service routine of the timer.

#### Returns

none.

#### 7.1225.2.2 TimerGet()

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

#### Returns

Current value of the millisecond timer.

### 7.1225.2.3 TimerInit()

```
void TimerInit (
 void)
```

Initializes the timer.

#### Returns

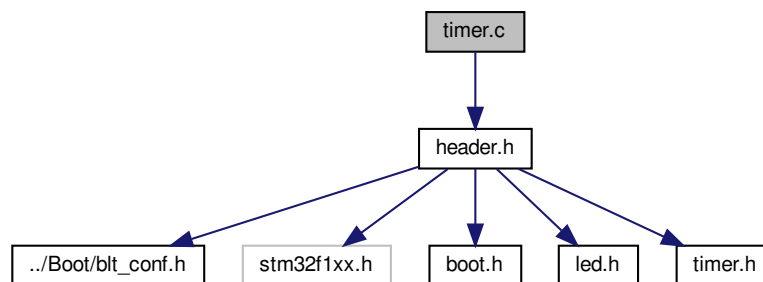
none.

## 7.1226 timer.c File Reference

Timer driver source file.

```
#include "header.h"
```

Include dependency graph for Demo/ARMCM3\_STM32F1\_Nucleo\_F103RB\_Keil/Prog/timer.c:



## Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*
- void [SysTick\\_Handler](#) (void)  
*Interrupt service routine of the timer.*

### 7.1226.1 Detailed Description

Timer driver source file.

### 7.1226.2 Function Documentation

### 7.1226.2.1 SysTick\_Handler()

```
void SysTick_Handler (
 void)
```

Interrupt service routine of the timer.

#### Returns

none.

### 7.1226.2.2 TimerGet()

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

#### Returns

Current value of the millisecond timer.

### 7.1226.2.3 TimerInit()

```
void TimerInit (
 void)
```

Initializes the timer.

#### Returns

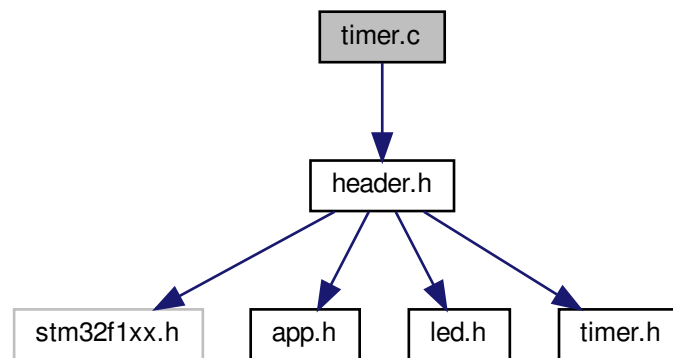
none.

## 7.1227 timer.c File Reference

Timer driver source file.

```
#include "header.h"
```

Include dependency graph for Demo/ARMCM3\_STM32F1\_Olimex\_STM32H103\_CubeIDE/Prog/App/timer.c:



### Functions

- void `TimerInit` (void)  
*Initializes the timer.*
- unsigned long `TimerGet` (void)  
*Obtains the counter value of the millisecond timer.*

#### 7.1227.1 Detailed Description

Timer driver source file.

#### 7.1227.2 Function Documentation

##### 7.1227.2.1 TimerGet()

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

##### Returns

Current value of the millisecond timer.

### 7.1227.2.2 TimerInit()

```
void TimerInit (
 void)
```

Initializes the timer.

#### Returns

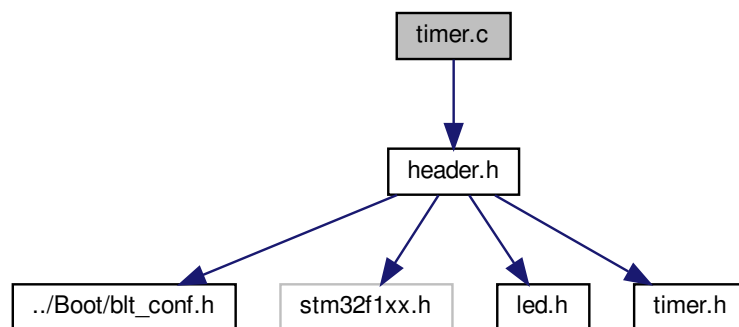
none.

## 7.1228 timer.c File Reference

Timer driver source file.

```
#include "header.h"
```

Include dependency graph for Demo/ARMCM3\_STM32F1\_Olimex\_STM32H103\_GCC/Prog/timer.c:



## Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*
- void [SysTick\\_Handler](#) (void)  
*Interrupt service routine of the timer.*

### 7.1228.1 Detailed Description

Timer driver source file.

## 7.1228.2 Function Documentation

### 7.1228.2.1 SysTick\_Handler()

```
void SysTick_Handler (
 void)
```

Interrupt service routine of the timer.

#### Returns

none.

### 7.1228.2.2 TimerGet()

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

#### Returns

Current value of the millisecond timer.

### 7.1228.2.3 TimerInit()

```
void TimerInit (
 void)
```

Initializes the timer.

#### Returns

none.

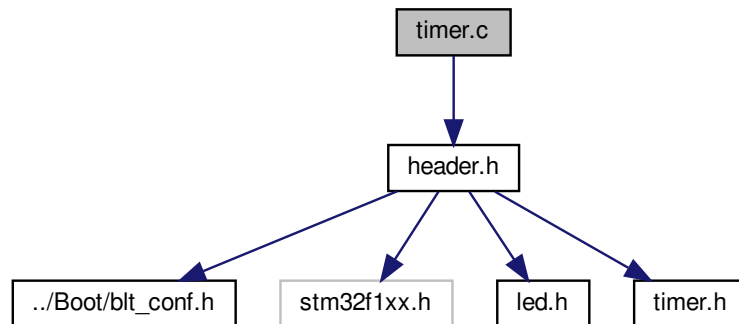


## 7.1229 timer.c File Reference

Timer driver source file.

```
#include "header.h"
```

Include dependency graph for Demo/ARMCM3\_STM32F1\_Olimex\_STM32H103\_IAR/Prog/timer.c:



### Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*
- void [SysTick\\_Handler](#) (void)  
*Interrupt service routine of the timer.*

### 7.1229.1 Detailed Description

Timer driver source file.

### 7.1229.2 Function Documentation

#### 7.1229.2.1 SysTick\_Handler()

```
void SysTick_Handler (
 void)
```

Interrupt service routine of the timer.

#### Returns

none.

### 7.1229.2.2 TimerGet()

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

#### Returns

Current value of the millisecond timer.

### 7.1229.2.3 TimerInit()

```
void TimerInit (
 void)
```

Initializes the timer.

#### Returns

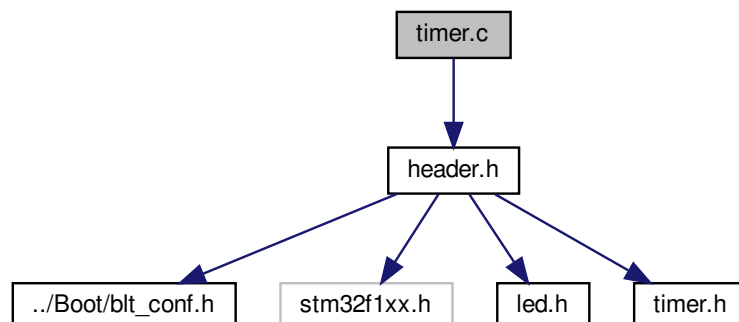
none.

## 7.1230 timer.c File Reference

Timer driver source file.

```
#include "header.h"
```

Include dependency graph for Demo/ARMCM3\_STM32F1\_Olimex\_STM32H103\_Keil/Prog/timer.c:



## Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*
- void [SysTick\\_Handler](#) (void)  
*Interrupt service routine of the timer.*

### 7.1230.1 Detailed Description

Timer driver source file.

### 7.1230.2 Function Documentation

#### 7.1230.2.1 SysTick\_Handler()

```
void SysTick_Handler (
 void)
```

Interrupt service routine of the timer.

#### Returns

none.

#### 7.1230.2.2 TimerGet()

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

#### Returns

Current value of the millisecond timer.

### 7.1230.2.3 TimerInit()

```
void TimerInit (
 void)
```

Initializes the timer.

#### Returns

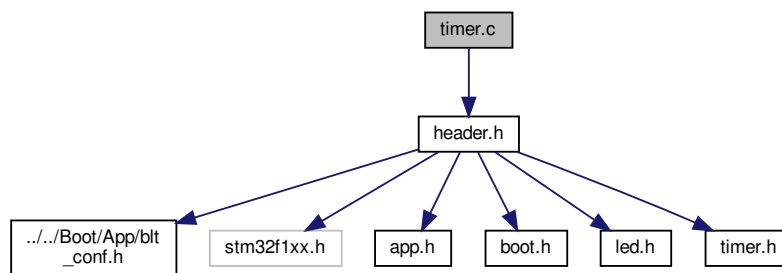
none.

## 7.1231 timer.c File Reference

Timer driver source file.

```
#include "header.h"
```

Include dependency graph for Demo/ARMCM3\_STM32F1\_Olimex\_STM32P103\_CubeIDE/Prog/App/timer.c:



## Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*

### 7.1231.1 Detailed Description

Timer driver source file.

### 7.1231.2 Function Documentation

### 7.1231.2.1 TimerGet()

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

#### Returns

Current value of the millisecond timer.

### 7.1231.2.2 TimerInit()

```
void TimerInit (
 void)
```

Initializes the timer.

#### Returns

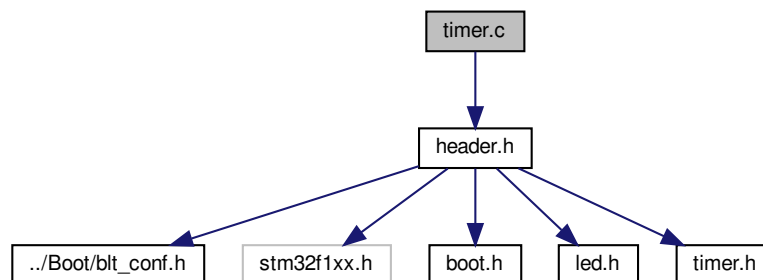
none.

## 7.1232 timer.c File Reference

Timer driver source file.

```
#include "header.h"
```

Include dependency graph for Demo/ARMCM3\_STM32F1\_Olimex\_STM32P103\_GCC/Prog/timer.c:



## Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*
- void [SysTick\\_Handler](#) (void)  
*Interrupt service routine of the timer.*

### 7.1232.1 Detailed Description

Timer driver source file.

### 7.1232.2 Function Documentation

#### 7.1232.2.1 SysTick\_Handler()

```
void SysTick_Handler (
 void)
```

Interrupt service routine of the timer.

##### Returns

none.

#### 7.1232.2.2 TimerGet()

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

##### Returns

Current value of the millisecond timer.

#### 7.1232.2.3 TimerInit()

```
void TimerInit (
 void)
```

Initializes the timer.

##### Returns

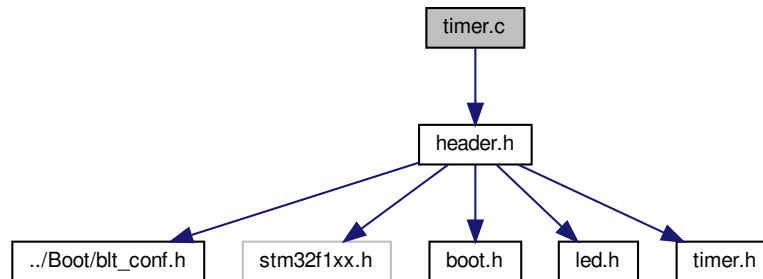
none.

## 7.1233 timer.c File Reference

Timer driver source file.

```
#include "header.h"
```

Include dependency graph for Demo/ARMCM3\_STM32F1\_Olimex\_STM32P103\_IAR/Prog/timer.c:



### Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*
- void [SysTick\\_Handler](#) (void)  
*Interrupt service routine of the timer.*

### 7.1233.1 Detailed Description

Timer driver source file.

### 7.1233.2 Function Documentation

#### 7.1233.2.1 SysTick\_Handler()

```
void SysTick_Handler (
 void)
```

Interrupt service routine of the timer.

#### Returns

none.

### 7.1233.2.2 TimerGet()

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

#### Returns

Current value of the millisecond timer.

### 7.1233.2.3 TimerInit()

```
void TimerInit (
 void)
```

Initializes the timer.

#### Returns

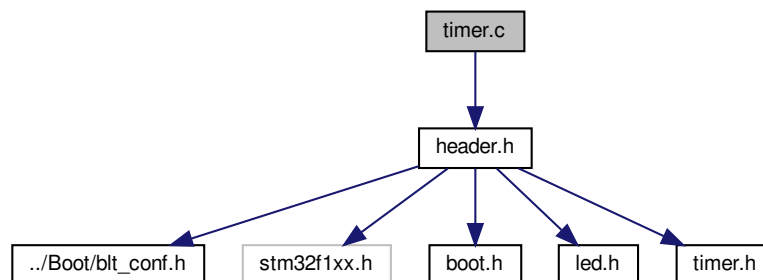
none.

## 7.1234 timer.c File Reference

Timer driver source file.

```
#include "header.h"
```

Include dependency graph for Demo/ARMCM3\_STM32F1\_Olimex\_STM32P103\_Keil/Prog/timer.c:



## Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*
- void [SysTick\\_Handler](#) (void)  
*Interrupt service routine of the timer.*



### 7.1234.1 Detailed Description

Timer driver source file.

### 7.1234.2 Function Documentation

#### 7.1234.2.1 SysTick\_Handler()

```
void SysTick_Handler (
 void)
```

Interrupt service routine of the timer.

##### Returns

none.

#### 7.1234.2.2 TimerGet()

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

##### Returns

Current value of the millisecond timer.

#### 7.1234.2.3 TimerInit()

```
void TimerInit (
 void)
```

Initializes the timer.

##### Returns

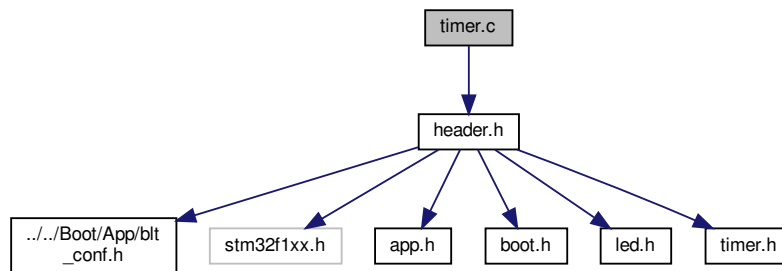
none.

## 7.1235 timer.c File Reference

Timer driver source file.

```
#include "header.h"
```

Include dependency graph for Demo/ARMCM3\_STM32F1\_Olimexino\_STM32\_CubeIDE/Prog/App/timer.c:



### Functions

- void `TimerInit` (void)  
*Initializes the timer.*
- unsigned long `TimerGet` (void)  
*Obtains the counter value of the millisecond timer.*

### 7.1235.1 Detailed Description

Timer driver source file.

### 7.1235.2 Function Documentation

#### 7.1235.2.1 TimerGet()

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

#### Returns

Current value of the millisecond timer.

### 7.1235.2.2 TimerInit()

```
void TimerInit (
 void)
```

Initializes the timer.

#### Returns

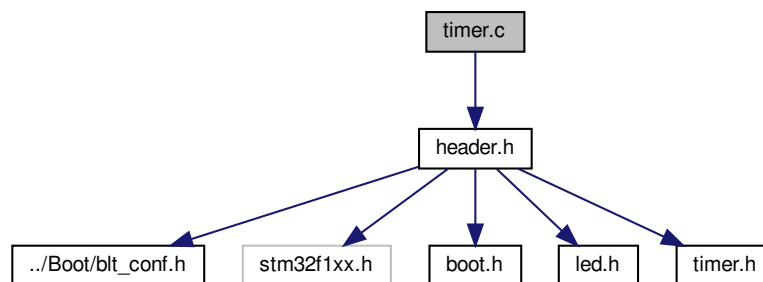
none.

## 7.1236 timer.c File Reference

Timer driver source file.

```
#include "header.h"
```

Include dependency graph for Demo/ARMCM3\_STM32F1\_Olimexino\_STM32\_GCC/Prog/timer.c:



## Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*
- void [SysTick\\_Handler](#) (void)  
*Interrupt service routine of the timer.*

### 7.1236.1 Detailed Description

Timer driver source file.

### 7.1236.2 Function Documentation

### 7.1236.2.1 SysTick\_Handler()

```
void SysTick_Handler (
 void)
```

Interrupt service routine of the timer.

#### Returns

none.

### 7.1236.2.2 TimerGet()

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

#### Returns

Current value of the millisecond timer.

### 7.1236.2.3 TimerInit()

```
void TimerInit (
 void)
```

Initializes the timer.

#### Returns

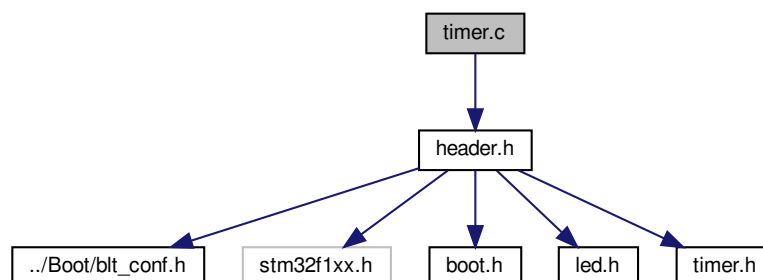
none.

## 7.1237 timer.c File Reference

Timer driver source file.

```
#include "header.h"
```

Include dependency graph for Demo/ARMCM3\_STM32F1\_Olimexino\_STM32\_IAR/Prog/timer.c:



## Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*
- void [SysTick\\_Handler](#) (void)  
*Interrupt service routine of the timer.*

### 7.1237.1 Detailed Description

Timer driver source file.

### 7.1237.2 Function Documentation

#### 7.1237.2.1 SysTick\_Handler()

```
void SysTick_Handler (
 void)
```

Interrupt service routine of the timer.

#### Returns

none.

#### 7.1237.2.2 TimerGet()

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

#### Returns

Current value of the millisecond timer.

### 7.1237.2.3 TimerInit()

```
void TimerInit (
 void)
```

Initializes the timer.

#### Returns

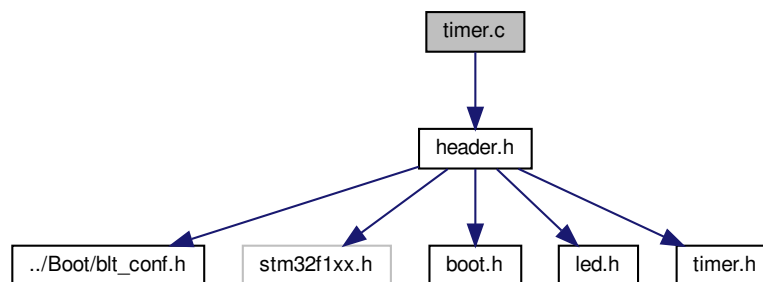
none.

## 7.1238 timer.c File Reference

Timer driver source file.

```
#include "header.h"
```

Include dependency graph for Demo/ARMCM3\_STM32F1\_Olimexino\_STM32\_Keil/Prog/timer.c:



## Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*
- void [SysTick\\_Handler](#) (void)  
*Interrupt service routine of the timer.*

### 7.1238.1 Detailed Description

Timer driver source file.

### 7.1238.2 Function Documentation

### 7.1238.2.1 SysTick\_Handler()

```
void SysTick_Handler (
 void)
```

Interrupt service routine of the timer.

#### Returns

none.

### 7.1238.2.2 TimerGet()

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

#### Returns

Current value of the millisecond timer.

### 7.1238.2.3 TimerInit()

```
void TimerInit (
 void)
```

Initializes the timer.

#### Returns

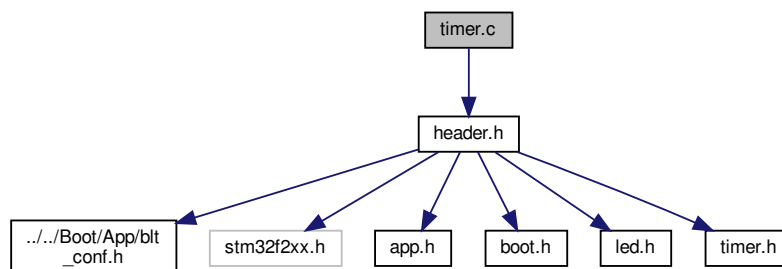
none.

## 7.1239 timer.c File Reference

Timer driver source file.

```
#include "header.h"
```

Include dependency graph for Demo/ARMCM3\_STM32F2\_Olimex\_STM32P207\_CubeIDE/Prog/App/timer.c:



## Functions

- void `TimerInit` (void)  
*Initializes the timer.*
- unsigned long `TimerGet` (void)  
*Obtains the counter value of the millisecond timer.*

### 7.1239.1 Detailed Description

Timer driver source file.

### 7.1239.2 Function Documentation

#### 7.1239.2.1 `TimerGet()`

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

#### Returns

Current value of the millisecond timer.

#### 7.1239.2.2 `TimerInit()`

```
void TimerInit (
 void)
```

Initializes the timer.

#### Returns

none.

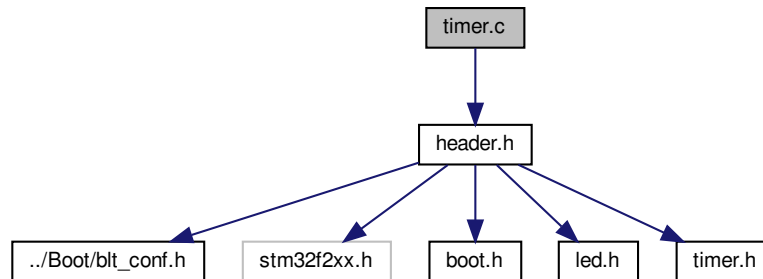


## 7.1240 timer.c File Reference

Timer driver source file.

```
#include "header.h"
```

Include dependency graph for Demo/ARMCM3\_STM32F2\_Olimex\_STM32P207\_GCC/Prog/timer.c:



### Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*
- void [SysTick\\_Handler](#) (void)  
*Interrupt service routine of the timer.*

#### 7.1240.1 Detailed Description

Timer driver source file.

#### 7.1240.2 Function Documentation

##### 7.1240.2.1 SysTick\_Handler()

```
void SysTick_Handler (
 void)
```

Interrupt service routine of the timer.

#### Returns

none.

### 7.1240.2.2 TimerGet()

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

#### Returns

Current value of the millisecond timer.

### 7.1240.2.3 TimerInit()

```
void TimerInit (
 void)
```

Initializes the timer.

#### Returns

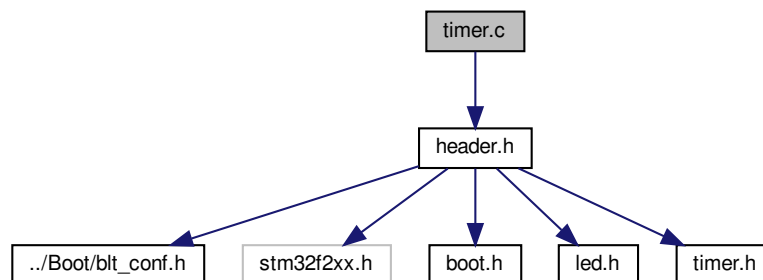
none.

## 7.1241 timer.c File Reference

Timer driver source file.

```
#include "header.h"
```

Include dependency graph for Demo/ARMCM3\_STM32F2\_Olimex\_STM32P207\_IAR/Prog/timer.c:



## Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*
- void [SysTick\\_Handler](#) (void)  
*Interrupt service routine of the timer.*

### 7.1241.1 Detailed Description

Timer driver source file.

### 7.1241.2 Function Documentation

#### 7.1241.2.1 SysTick\_Handler()

```
void SysTick_Handler (
 void)
```

Interrupt service routine of the timer.

##### Returns

none.

#### 7.1241.2.2 TimerGet()

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

##### Returns

Current value of the millisecond timer.

#### 7.1241.2.3 TimerInit()

```
void TimerInit (
 void)
```

Initializes the timer.

##### Returns

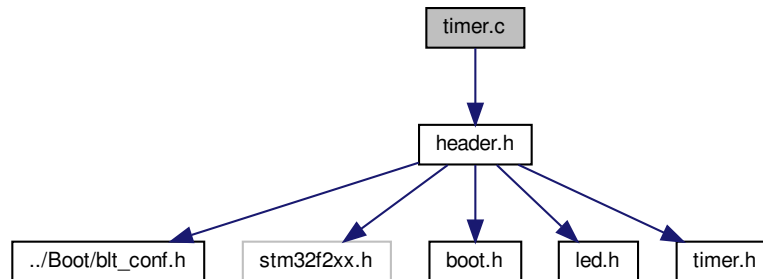
none.

## 7.1242 timer.c File Reference

Timer driver source file.

```
#include "header.h"
```

Include dependency graph for Demo/ARMCM3\_STM32F2\_Olimex\_STM32P207\_Keil/Prog/timer.c:



### Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*
- void [SysTick\\_Handler](#) (void)  
*Interrupt service routine of the timer.*

### 7.1242.1 Detailed Description

Timer driver source file.

### 7.1242.2 Function Documentation

#### 7.1242.2.1 SysTick\_Handler()

```
void SysTick_Handler (
 void)
```

Interrupt service routine of the timer.

#### Returns

none.

### 7.1242.2.2 TimerGet()

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

#### Returns

Current value of the millisecond timer.

### 7.1242.2.3 TimerInit()

```
void TimerInit (
 void)
```

Initializes the timer.

#### Returns

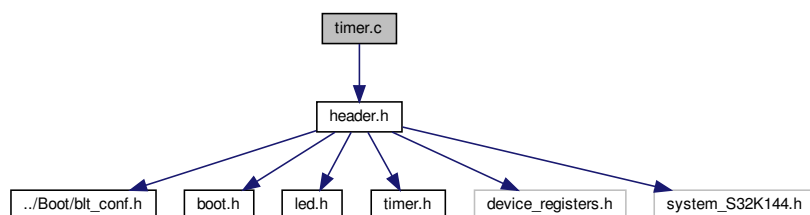
none.

## 7.1243 timer.c File Reference

Timer driver source file.

```
#include "header.h"
```

Include dependency graph for Demo/ARMCM4\_S32K14\_S32K144EVB\_GCC/Prog/timer.c:



## Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*
- void [SysTick\\_Handler](#) (void)  
*Interrupt service routine of the timer.*

## Variables

- static unsigned long [millisecond\\_counter](#)  
*Local variable for storing the number of milliseconds that have elapsed since startup.*

## 7.1243.1 Detailed Description

Timer driver source file.

## 7.1243.2 Function Documentation

### 7.1243.2.1 SysTick\_Handler()

```
void SysTick_Handler (
 void)
```

Interrupt service routine of the timer.

#### Returns

none.

### 7.1243.2.2 TimerGet()

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

#### Returns

Current value of the millisecond timer.

### 7.1243.2.3 TimerInit()

```
void TimerInit (
 void)
```

Initializes the timer.

#### Returns

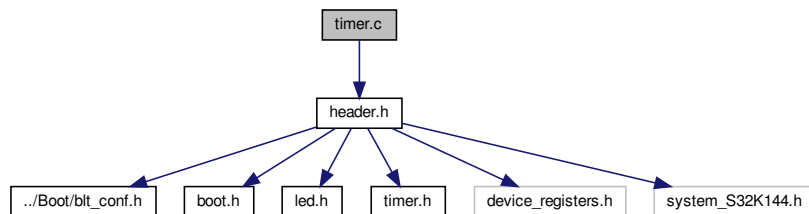
none.

## 7.1244 timer.c File Reference

Timer driver source file.

```
#include "header.h"
```

Include dependency graph for Demo/ARMCM4\_S32K14\_S32K144EVB\_IAR/Prog/timer.c:



### Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*
- void [SysTick\\_Handler](#) (void)  
*Interrupt service routine of the timer.*

### Variables

- static unsigned long [millisecond\\_counter](#)  
*Local variable for storing the number of milliseconds that have elapsed since startup.*

### 7.1244.1 Detailed Description

Timer driver source file.

### 7.1244.2 Function Documentation

#### 7.1244.2.1 SysTick\_Handler()

```
void SysTick_Handler (
 void)
```

Interrupt service routine of the timer.

#### Returns

none.

### 7.1244.2.2 TimerGet()

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

#### Returns

Current value of the millisecond timer.

### 7.1244.2.3 TimerInit()

```
void TimerInit (
 void)
```

Initializes the timer.

#### Returns

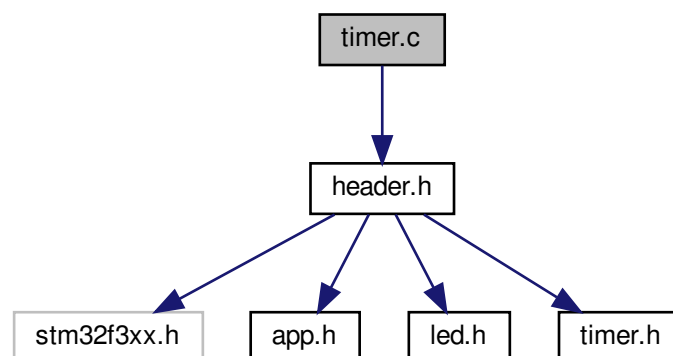
none.

## 7.1245 timer.c File Reference

Timer driver source file.

```
#include "header.h"
```

Include dependency graph for Demo/ARMCM4\_STM32F3\_Discovery\_F303VC\_CubeIDE/Prog/App/timer.c:





## Functions

- void `TimerInit` (void)  
*Initializes the timer.*
- unsigned long `TimerGet` (void)  
*Obtains the counter value of the millisecond timer.*

### 7.1245.1 Detailed Description

Timer driver source file.

### 7.1245.2 Function Documentation

#### 7.1245.2.1 `TimerGet()`

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

#### Returns

Current value of the millisecond timer.

#### 7.1245.2.2 `TimerInit()`

```
void TimerInit (
 void)
```

Initializes the timer.

#### Returns

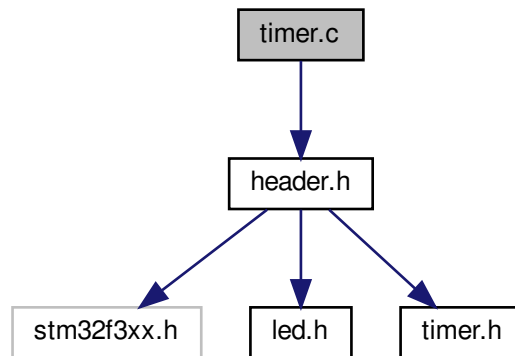
none.

## 7.1246 timer.c File Reference

Timer driver source file.

```
#include "header.h"
```

Include dependency graph for Demo/ARMCM4\_STM32F3\_Discovery\_F303VC\_GCC/Prog/timer.c:



### Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*
- void [SysTick\\_Handler](#) (void)  
*Interrupt service routine of the timer.*

### 7.1246.1 Detailed Description

Timer driver source file.

### 7.1246.2 Function Documentation

#### 7.1246.2.1 SysTick\_Handler()

```
void SysTick_Handler (
 void)
```

Interrupt service routine of the timer.

#### Returns

none.

### 7.1246.2.2 TimerGet()

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

#### Returns

Current value of the millisecond timer.

### 7.1246.2.3 TimerInit()

```
void TimerInit (
 void)
```

Initializes the timer.

#### Returns

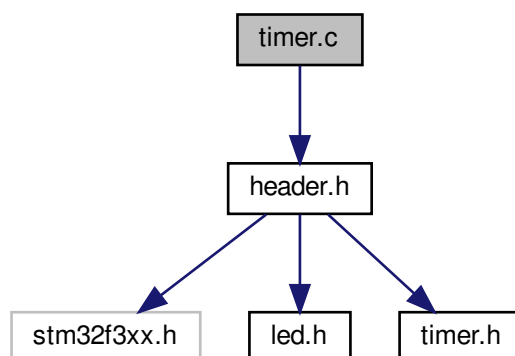
none.

## 7.1247 timer.c File Reference

Timer driver source file.

```
#include "header.h"
```

Include dependency graph for Demo/ARMCM4\_STM32F3\_Discovery\_F303VC\_IAR/Prog/timer.c:



## Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*
- void [SysTick\\_Handler](#) (void)  
*Interrupt service routine of the timer.*

### 7.1247.1 Detailed Description

Timer driver source file.

### 7.1247.2 Function Documentation

#### 7.1247.2.1 SysTick\_Handler()

```
void SysTick_Handler (
 void)
```

Interrupt service routine of the timer.

#### Returns

none.

#### 7.1247.2.2 TimerGet()

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

#### Returns

Current value of the millisecond timer.

### 7.1247.2.3 TimerInit()

```
void TimerInit (
 void)
```

Initializes the timer.

#### Returns

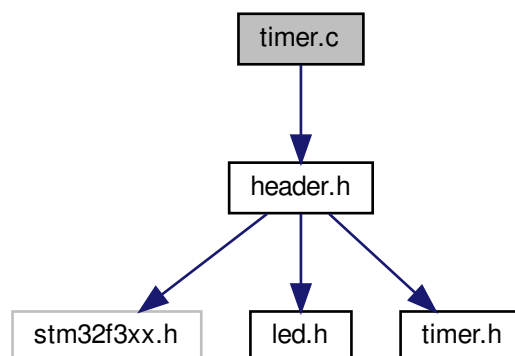
none.

## 7.1248 timer.c File Reference

Timer driver source file.

```
#include "header.h"
```

Include dependency graph for Demo/ARMCM4\_STM32F3\_Discovery\_F303VC\_Keil/Prog/timer.c:



### Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*
- void [SysTick\\_Handler](#) (void)  
*Interrupt service routine of the timer.*

### 7.1248.1 Detailed Description

Timer driver source file.

## 7.1248.2 Function Documentation

### 7.1248.2.1 SysTick\_Handler()

```
void SysTick_Handler (
 void)
```

Interrupt service routine of the timer.

#### Returns

none.

### 7.1248.2.2 TimerGet()

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

#### Returns

Current value of the millisecond timer.

### 7.1248.2.3 TimerInit()

```
void TimerInit (
 void)
```

Initializes the timer.

#### Returns

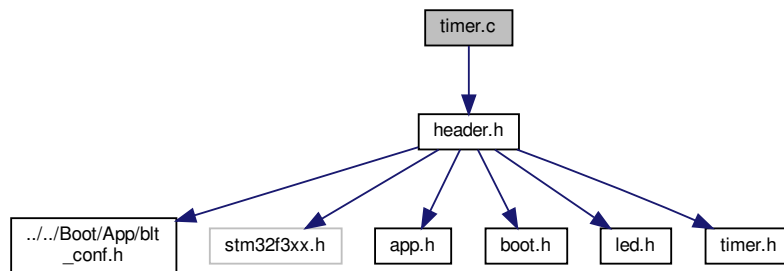
none.

## 7.1249 timer.c File Reference

Timer driver source file.

```
#include "header.h"
```

Include dependency graph for Demo/ARMCM4\_STM32F3\_Nucleo\_F303K8\_CubeIDE/Prog/App/timer.c:



### Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*

### 7.1249.1 Detailed Description

Timer driver source file.

### 7.1249.2 Function Documentation

#### 7.1249.2.1 TimerGet()

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

#### Returns

Current value of the millisecond timer.

### 7.1249.2.2 TimerInit()

```
void TimerInit (
 void)
```

Initializes the timer.

#### Returns

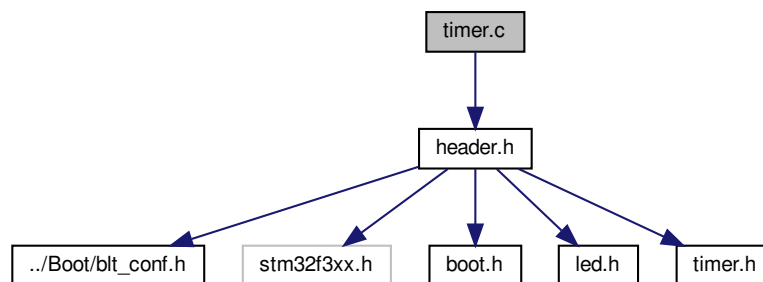
none.

## 7.1250 timer.c File Reference

Timer driver source file.

```
#include "header.h"
```

Include dependency graph for Demo/ARMCM4\_STM32F3\_Nucleo\_F303K8\_GCC/Prog/timer.c:



## Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*
- void [SysTick\\_Handler](#) (void)  
*Interrupt service routine of the timer.*

### 7.1250.1 Detailed Description

Timer driver source file.

### 7.1250.2 Function Documentation



### 7.1250.2.1 SysTick\_Handler()

```
void SysTick_Handler (
 void)
```

Interrupt service routine of the timer.

#### Returns

none.

### 7.1250.2.2 TimerGet()

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

#### Returns

Current value of the millisecond timer.

### 7.1250.2.3 TimerInit()

```
void TimerInit (
 void)
```

Initializes the timer.

#### Returns

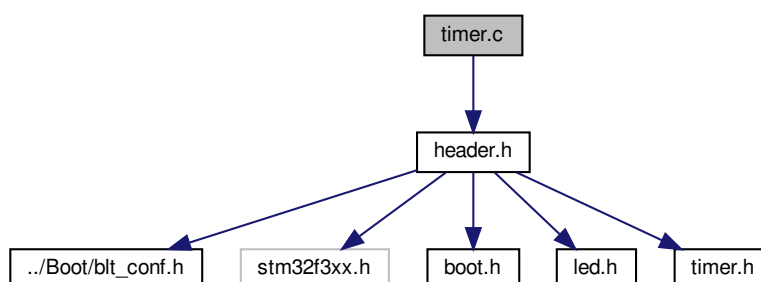
none.

## 7.1251 timer.c File Reference

Timer driver source file.

```
#include "header.h"
```

Include dependency graph for Demo/ARMCM4\_STM32F3\_Nucleo\_F303K8\_IAR/Prog/timer.c:



## Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*
- void [SysTick\\_Handler](#) (void)  
*Interrupt service routine of the timer.*

### 7.1251.1 Detailed Description

Timer driver source file.

### 7.1251.2 Function Documentation

#### 7.1251.2.1 SysTick\_Handler()

```
void SysTick_Handler (
 void)
```

Interrupt service routine of the timer.

#### Returns

none.

#### 7.1251.2.2 TimerGet()

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

#### Returns

Current value of the millisecond timer.

### 7.1251.2.3 TimerInit()

```
void TimerInit (
 void)
```

Initializes the timer.

#### Returns

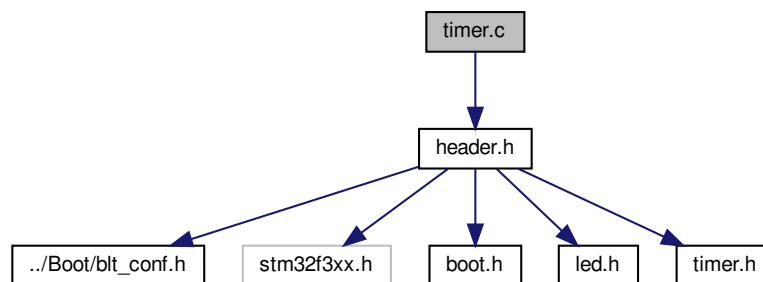
none.

## 7.1252 timer.c File Reference

Timer driver source file.

```
#include "header.h"
```

Include dependency graph for Demo/ARMCM4\_STM32F3\_Nucleo\_F303K8\_Keil/Prog/timer.c:



## Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*
- void [SysTick\\_Handler](#) (void)  
*Interrupt service routine of the timer.*

### 7.1252.1 Detailed Description

Timer driver source file.

### 7.1252.2 Function Documentation

### 7.1252.2.1 SysTick\_Handler()

```
void SysTick_Handler (
 void)
```

Interrupt service routine of the timer.

#### Returns

none.

### 7.1252.2.2 TimerGet()

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

#### Returns

Current value of the millisecond timer.

### 7.1252.2.3 TimerInit()

```
void TimerInit (
 void)
```

Initializes the timer.

#### Returns

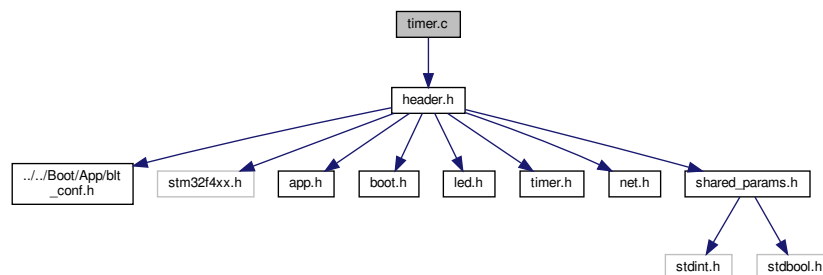
none.

## 7.1253 timer.c File Reference

Timer driver source file.

```
#include "header.h"
```

Include dependency graph for Demo/ARMCM4\_STM32F4\_Nucleo\_F429ZI\_CubeIDE/Prog/App/timer.c:



## Functions

- void `TimerInit` (void)  
*Initializes the timer.*
- unsigned long `TimerGet` (void)  
*Obtains the counter value of the millisecond timer.*

### 7.1253.1 Detailed Description

Timer driver source file.

### 7.1253.2 Function Documentation

#### 7.1253.2.1 `TimerGet()`

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

#### Returns

Current value of the millisecond timer.

#### 7.1253.2.2 `TimerInit()`

```
void TimerInit (
 void)
```

Initializes the timer.

#### Returns

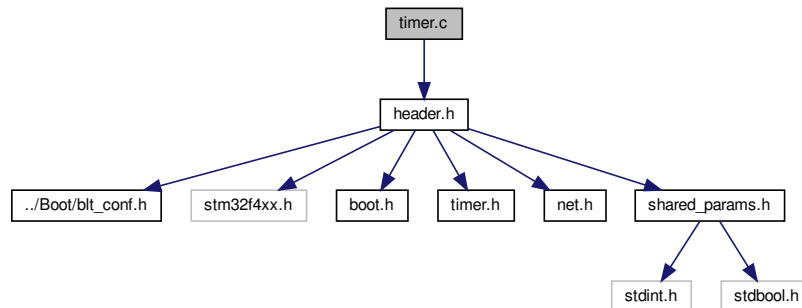
none.

## 7.1254 timer.c File Reference

Timer driver source file.

```
#include "header.h"
```

Include dependency graph for Demo/ARMCM4\_STM32F4\_Nucleo\_F429ZI\_GCC/Prog/timer.c:



### Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*
- void [SysTick\\_Handler](#) (void)  
*Interrupt service routine of the timer.*

### 7.1254.1 Detailed Description

Timer driver source file.

### 7.1254.2 Function Documentation

#### 7.1254.2.1 SysTick\_Handler()

```
void SysTick_Handler (
 void)
```

Interrupt service routine of the timer.

#### Returns

none.

### 7.1254.2.2 TimerGet()

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

#### Returns

Current value of the millisecond timer.

### 7.1254.2.3 TimerInit()

```
void TimerInit (
 void)
```

Initializes the timer.

#### Returns

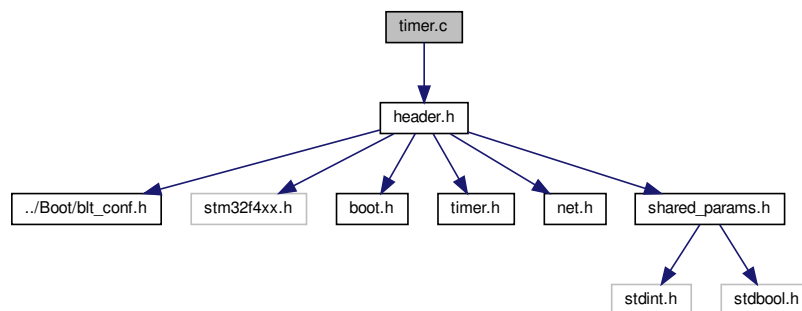
none.

## 7.1255 timer.c File Reference

Timer driver source file.

```
#include "header.h"
```

Include dependency graph for Demo/ARMCM4\_STM32F4\_Nucleo\_F429ZI\_IAR/Prog/timer.c:



## Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*
- void [SysTick\\_Handler](#) (void)  
*Interrupt service routine of the timer.*

### 7.1255.1 Detailed Description

Timer driver source file.

### 7.1255.2 Function Documentation

#### 7.1255.2.1 SysTick\_Handler()

```
void SysTick_Handler (
 void)
```

Interrupt service routine of the timer.

##### Returns

none.

#### 7.1255.2.2 TimerGet()

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

##### Returns

Current value of the millisecond timer.

#### 7.1255.2.3 TimerInit()

```
void TimerInit (
 void)
```

Initializes the timer.

##### Returns

none.

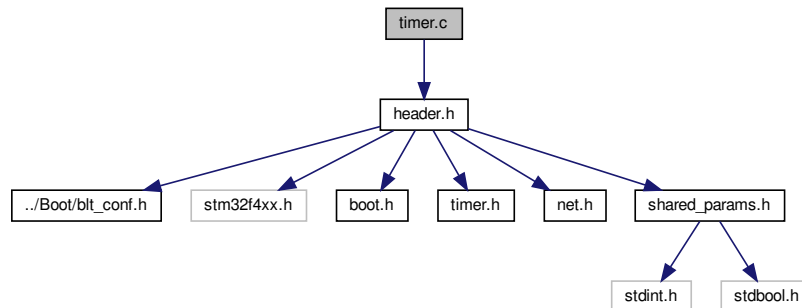


## 7.1256 timer.c File Reference

Timer driver source file.

```
#include "header.h"
```

Include dependency graph for Demo/ARMCM4\_STM32F4\_Nucleo\_F429ZI\_Keil/Prog/timer.c:



### Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*
- void [SysTick\\_Handler](#) (void)  
*Interrupt service routine of the timer.*

### 7.1256.1 Detailed Description

Timer driver source file.

### 7.1256.2 Function Documentation

#### 7.1256.2.1 SysTick\_Handler()

```
void SysTick_Handler (
 void)
```

Interrupt service routine of the timer.

#### Returns

none.

### 7.1256.2.2 TimerGet()

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

#### Returns

Current value of the millisecond timer.

### 7.1256.2.3 TimerInit()

```
void TimerInit (
 void)
```

Initializes the timer.

#### Returns

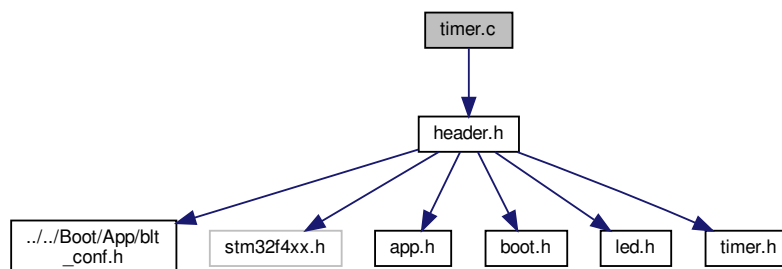
none.

## 7.1257 timer.c File Reference

Timer driver source file.

```
#include "header.h"
```

Include dependency graph for Demo/ARMCM4\_STM32F4\_Olimex\_STM32P405\_CubeIDE/Prog/App/timer.c:



## Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*

### 7.1257.1 Detailed Description

Timer driver source file.

### 7.1257.2 Function Documentation

#### 7.1257.2.1 TimerGet()

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

##### Returns

Current value of the millisecond timer.

#### 7.1257.2.2 TimerInit()

```
void TimerInit (
 void)
```

Initializes the timer.

##### Returns

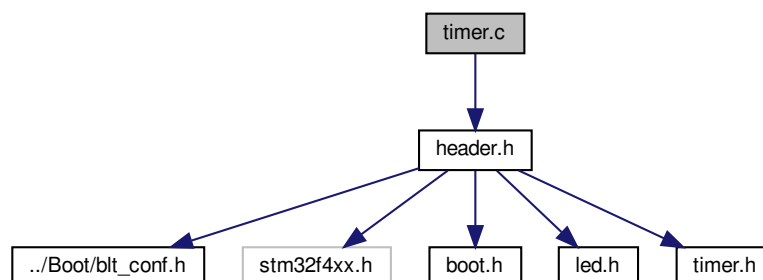
none.

## 7.1258 timer.c File Reference

Timer driver source file.

```
#include "header.h"
```

Include dependency graph for Demo/ARMCM4\_STM32F4\_Olimex\_STM32P405\_GCC/Prog/timer.c:



## Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*
- void [SysTick\\_Handler](#) (void)  
*Interrupt service routine of the timer.*

### 7.1258.1 Detailed Description

Timer driver source file.

### 7.1258.2 Function Documentation

#### 7.1258.2.1 SysTick\_Handler()

```
void SysTick_Handler (
 void)
```

Interrupt service routine of the timer.

#### Returns

none.

#### 7.1258.2.2 TimerGet()

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

#### Returns

Current value of the millisecond timer.

### 7.1258.2.3 TimerInit()

```
void TimerInit (
 void)
```

Initializes the timer.

#### Returns

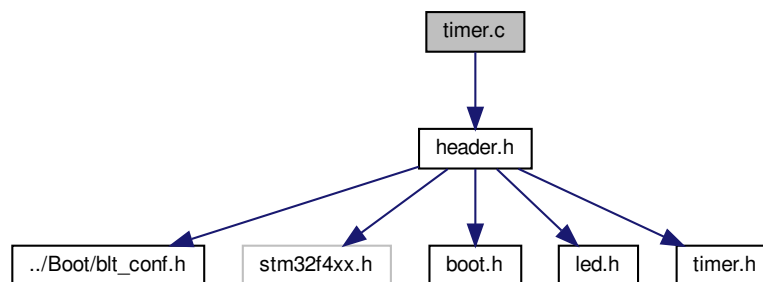
none.

## 7.1259 timer.c File Reference

Timer driver source file.

```
#include "header.h"
```

Include dependency graph for Demo/ARMCM4\_STM32F4\_Olimex\_STM32P405\_IAR/Prog/timer.c:



## Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*
- void [SysTick\\_Handler](#) (void)  
*Interrupt service routine of the timer.*

### 7.1259.1 Detailed Description

Timer driver source file.

### 7.1259.2 Function Documentation

### 7.1259.2.1 SysTick\_Handler()

```
void SysTick_Handler (
 void)
```

Interrupt service routine of the timer.

#### Returns

none.

### 7.1259.2.2 TimerGet()

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

#### Returns

Current value of the millisecond timer.

### 7.1259.2.3 TimerInit()

```
void TimerInit (
 void)
```

Initializes the timer.

#### Returns

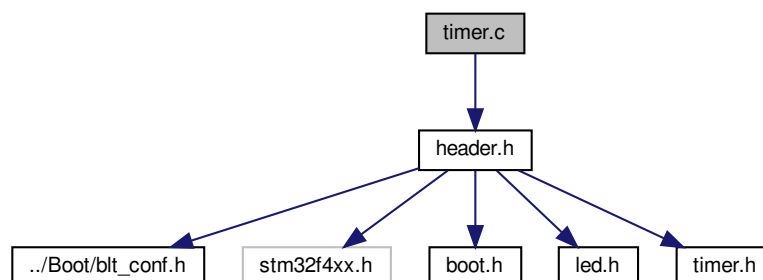
none.

## 7.1260 timer.c File Reference

Timer driver source file.

```
#include "header.h"
```

Include dependency graph for Demo/ARMCM4\_STM32F4\_Olimex\_STM32P405\_Keil/Prog/timer.c:



## Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*
- void [SysTick\\_Handler](#) (void)  
*Interrupt service routine of the timer.*

### 7.1260.1 Detailed Description

Timer driver source file.

### 7.1260.2 Function Documentation

#### 7.1260.2.1 SysTick\_Handler()

```
void SysTick_Handler (
 void)
```

Interrupt service routine of the timer.

#### Returns

none.

#### 7.1260.2.2 TimerGet()

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

#### Returns

Current value of the millisecond timer.

### 7.1260.2.3 TimerInit()

```
void TimerInit (
 void)
```

Initializes the timer.

#### Returns

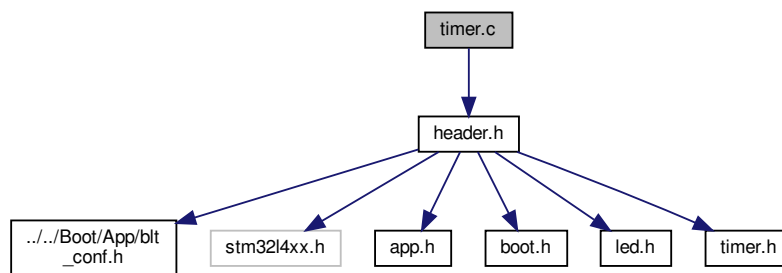
none.

## 7.1261 timer.c File Reference

Timer driver source file.

```
#include "header.h"
```

Include dependency graph for Demo/ARMCM4\_STM32L4\_Nucleo\_L476RG\_CubeIDE/Prog/App/timer.c:



## Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*

### 7.1261.1 Detailed Description

Timer driver source file.

### 7.1261.2 Function Documentation



### 7.1261.2.1 TimerGet()

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

#### Returns

Current value of the millisecond timer.

### 7.1261.2.2 TimerInit()

```
void TimerInit (
 void)
```

Initializes the timer.

#### Returns

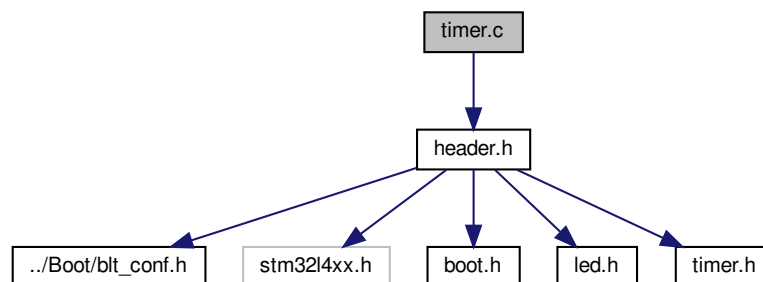
none.

## 7.1262 timer.c File Reference

Timer driver source file.

```
#include "header.h"
```

Include dependency graph for Demo/ARMCM4\_STM32L4\_Nucleo\_L476RG\_GCC/Prog/timer.c:



## Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*
- void [SysTick\\_Handler](#) (void)  
*Interrupt service routine of the timer.*

### 7.1262.1 Detailed Description

Timer driver source file.

### 7.1262.2 Function Documentation

#### 7.1262.2.1 SysTick\_Handler()

```
void SysTick_Handler (
 void)
```

Interrupt service routine of the timer.

##### Returns

none.

#### 7.1262.2.2 TimerGet()

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

##### Returns

Current value of the millisecond timer.

#### 7.1262.2.3 TimerInit()

```
void TimerInit (
 void)
```

Initializes the timer.

##### Returns

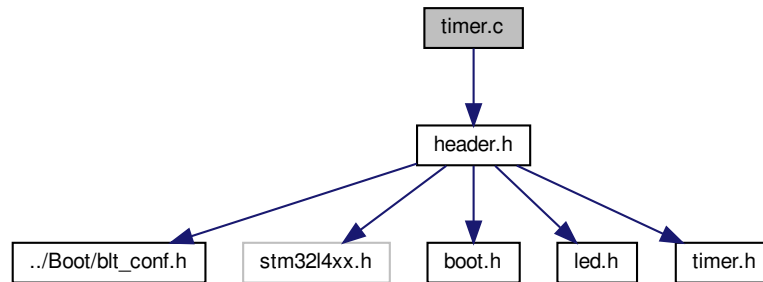
none.

## 7.1263 timer.c File Reference

Timer driver source file.

```
#include "header.h"
```

Include dependency graph for Demo/ARMCM4\_STM32L4\_Nucleo\_L476RG\_IAR/Prog/timer.c:



### Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*
- void [SysTick\\_Handler](#) (void)  
*Interrupt service routine of the timer.*

### 7.1263.1 Detailed Description

Timer driver source file.

### 7.1263.2 Function Documentation

#### 7.1263.2.1 SysTick\_Handler()

```
void SysTick_Handler (
 void)
```

Interrupt service routine of the timer.

#### Returns

none.

### 7.1263.2.2 TimerGet()

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

#### Returns

Current value of the millisecond timer.

### 7.1263.2.3 TimerInit()

```
void TimerInit (
 void)
```

Initializes the timer.

#### Returns

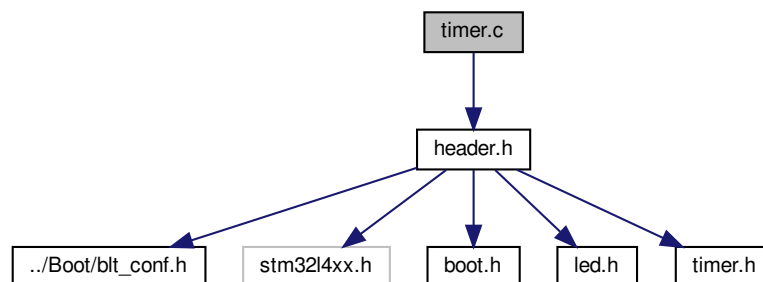
none.

## 7.1264 timer.c File Reference

Timer driver source file.

```
#include "header.h"
```

Include dependency graph for Demo/ARMCM4\_STM32L4\_Nucleo\_L476RG\_Keil/Prog/timer.c:



## Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*
- void [SysTick\\_Handler](#) (void)  
*Interrupt service routine of the timer.*

### 7.1264.1 Detailed Description

Timer driver source file.

### 7.1264.2 Function Documentation

#### 7.1264.2.1 SysTick\_Handler()

```
void SysTick_Handler (
 void)
```

Interrupt service routine of the timer.

##### Returns

none.

#### 7.1264.2.2 TimerGet()

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

##### Returns

Current value of the millisecond timer.

#### 7.1264.2.3 TimerInit()

```
void TimerInit (
 void)
```

Initializes the timer.

##### Returns

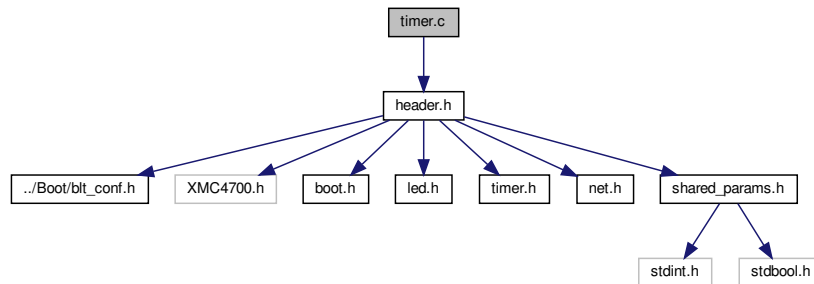
none.

## 7.1265 timer.c File Reference

Timer driver source file.

```
#include "header.h"
```

Include dependency graph for Demo/ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_GCC/Prog/timer.c:



### Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- void [TimerDeinit](#) (void)  
*Stops and disables the timer.*
- void [TimerSet](#) (unsigned long timer\_value)  
*Sets the initial counter value of the millisecond timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*
- void [SysTick\\_Handler](#) (void)  
*Interrupt service routine of the timer.*

### Variables

- static unsigned long [millisecond\\_counter](#)  
*Local variable for storing the number of milliseconds that have elapsed since startup.*

#### 7.1265.1 Detailed Description

Timer driver source file.

#### 7.1265.2 Function Documentation

### 7.1265.2.1 SysTick\_Handler()

```
void SysTick_Handler (
 void)
```

Interrupt service routine of the timer.

#### Returns

none.

### 7.1265.2.2 TimerDeinit()

```
void TimerDeinit (
 void)
```

Stops and disables the timer.

#### Returns

none.

### 7.1265.2.3 TimerGet()

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

#### Returns

Current value of the millisecond timer.

### 7.1265.2.4 TimerInit()

```
void TimerInit (
 void)
```

Initializes the timer.

#### Returns

none.

### 7.1265.2.5 TimerSet()

```
void TimerSet (
 unsigned long timer_value)
```

Sets the initial counter value of the millisecond timer.

## Parameters

|                    |                                            |
|--------------------|--------------------------------------------|
| <i>timer_value</i> | initialize value of the millisecond timer. |
|--------------------|--------------------------------------------|

## Returns

none.

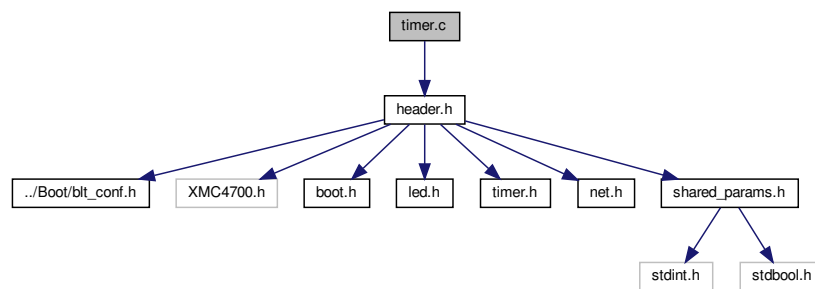
Referenced by TimerInit().

## 7.1266 timer.c File Reference

Timer driver source file.

```
#include "header.h"
```

Include dependency graph for Demo/ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_IAR/Prog/timer.c:



## Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- void [TimerDeinit](#) (void)  
*Stops and disables the timer.*
- void [TimerSet](#) (unsigned long timer\_value)  
*Sets the initial counter value of the millisecond timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*
- void [SysTick\\_Handler](#) (void)  
*Interrupt service routine of the timer.*

## Variables

- static unsigned long [millisecond\\_counter](#)  
*Local variable for storing the number of milliseconds that have elapsed since startup.*



## 7.1266.1 Detailed Description

Timer driver source file.

## 7.1266.2 Function Documentation

### 7.1266.2.1 SysTick\_Handler()

```
void SysTick_Handler (
 void)
```

Interrupt service routine of the timer.

#### Returns

none.

### 7.1266.2.2 TimerDeinit()

```
void TimerDeinit (
 void)
```

Stops and disables the timer.

#### Returns

none.

### 7.1266.2.3 TimerGet()

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

#### Returns

Current value of the millisecond timer.

#### 7.1266.2.4 TimerInit()

```
void TimerInit (
 void)
```

Initializes the timer.

##### Returns

none.

#### 7.1266.2.5 TimerSet()

```
void TimerSet (
 unsigned long timer_value)
```

Sets the initial counter value of the millisecond timer.

##### Parameters

|                    |                                            |
|--------------------|--------------------------------------------|
| <i>timer_value</i> | initialize value of the millisecond timer. |
|--------------------|--------------------------------------------|

##### Returns

none.

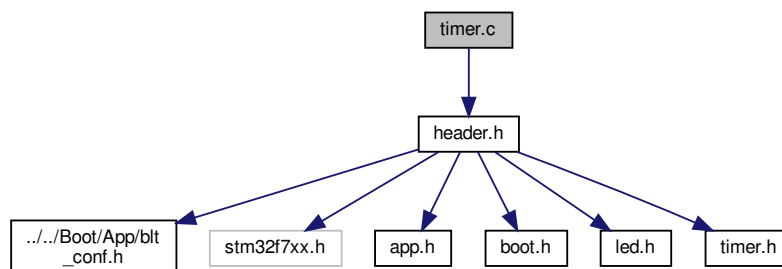
Referenced by TimerInit().

## 7.1267 timer.c File Reference

Timer driver source file.

```
#include "header.h"
```

Include dependency graph for Demo/ARMCM7\_STM32F7\_Nucleo\_F746ZG\_CubeIDE/Prog/App/timer.c:



## Functions

- void `TimerInit` (void)  
*Initializes the timer.*
- unsigned long `TimerGet` (void)  
*Obtains the counter value of the millisecond timer.*

### 7.1267.1 Detailed Description

Timer driver source file.

### 7.1267.2 Function Documentation

#### 7.1267.2.1 `TimerGet()`

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

#### Returns

Current value of the millisecond timer.

#### 7.1267.2.2 `TimerInit()`

```
void TimerInit (
 void)
```

Initializes the timer.

#### Returns

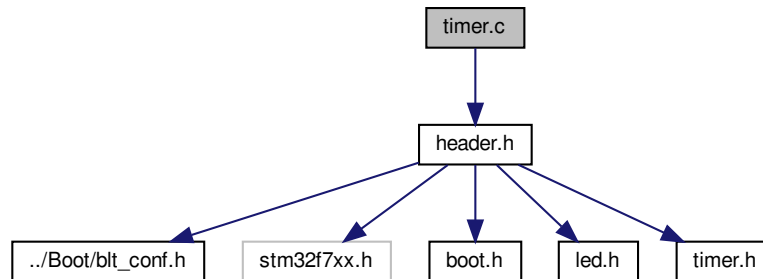
none.

## 7.1268 timer.c File Reference

Timer driver source file.

```
#include "header.h"
```

Include dependency graph for Demo/ARMCM7\_STM32F7\_Nucleo\_F746ZG\_GCC/Prog/timer.c:



### Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*
- void [SysTick\\_Handler](#) (void)  
*Interrupt service routine of the timer.*

### 7.1268.1 Detailed Description

Timer driver source file.

### 7.1268.2 Function Documentation

#### 7.1268.2.1 SysTick\_Handler()

```
void SysTick_Handler (
 void)
```

Interrupt service routine of the timer.

#### Returns

none.

### 7.1268.2.2 TimerGet()

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

#### Returns

Current value of the millisecond timer.

### 7.1268.2.3 TimerInit()

```
void TimerInit (
 void)
```

Initializes the timer.

#### Returns

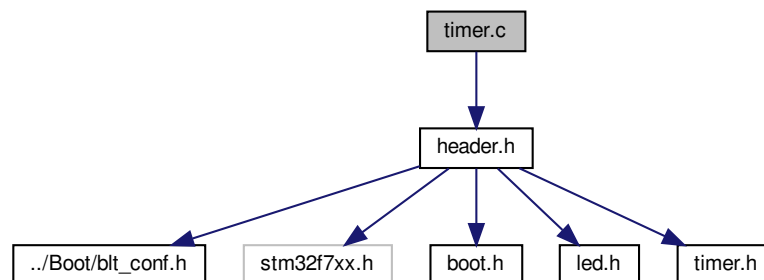
none.

## 7.1269 timer.c File Reference

Timer driver source file.

```
#include "header.h"
```

Include dependency graph for Demo/ARMCM7\_STM32F7\_Nucleo\_F746ZG\_IAR/Prog/timer.c:



## Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*
- void [SysTick\\_Handler](#) (void)  
*Interrupt service routine of the timer.*

### 7.1269.1 Detailed Description

Timer driver source file.

### 7.1269.2 Function Documentation

#### 7.1269.2.1 SysTick\_Handler()

```
void SysTick_Handler (
 void)
```

Interrupt service routine of the timer.

##### Returns

none.

#### 7.1269.2.2 TimerGet()

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

##### Returns

Current value of the millisecond timer.

#### 7.1269.2.3 TimerInit()

```
void TimerInit (
 void)
```

Initializes the timer.

##### Returns

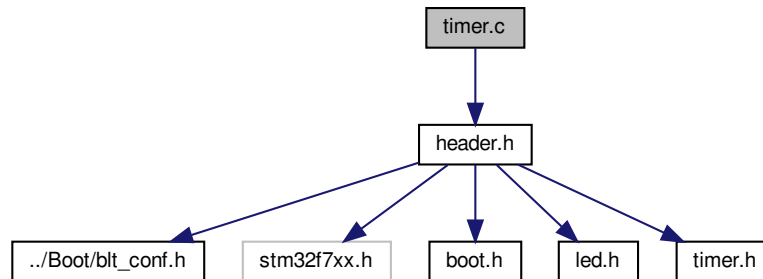
none.

## 7.1270 timer.c File Reference

Timer driver source file.

```
#include "header.h"
```

Include dependency graph for Demo/ARMCM7\_STM32F7\_Nucleo\_F746ZG\_Keil/Prog/timer.c:



### Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*
- void [SysTick\\_Handler](#) (void)  
*Interrupt service routine of the timer.*

### 7.1270.1 Detailed Description

Timer driver source file.

### 7.1270.2 Function Documentation

#### 7.1270.2.1 SysTick\_Handler()

```
void SysTick_Handler (
 void)
```

Interrupt service routine of the timer.

#### Returns

none.

### 7.1270.2.2 TimerGet()

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

#### Returns

Current value of the millisecond timer.

### 7.1270.2.3 TimerInit()

```
void TimerInit (
 void)
```

Initializes the timer.

#### Returns

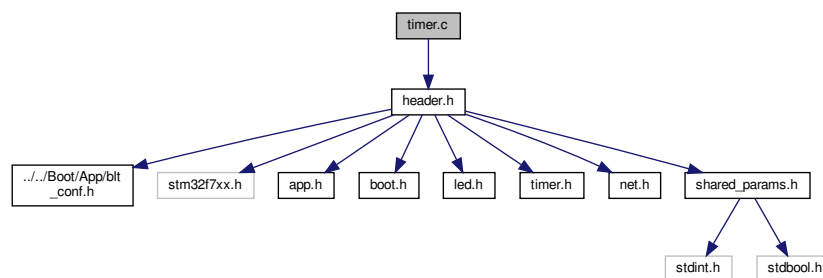
none.

## 7.1271 timer.c File Reference

Timer driver source file.

```
#include "header.h"
```

Include dependency graph for Demo/ARMCM7\_STM32F7\_Nucleo\_F767ZI\_CubeIDE/Prog/App/timer.c:



## Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*



### 7.1271.1 Detailed Description

Timer driver source file.

### 7.1271.2 Function Documentation

#### 7.1271.2.1 TimerGet()

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

##### Returns

Current value of the millisecond timer.

#### 7.1271.2.2 TimerInit()

```
void TimerInit (
 void)
```

Initializes the timer.

##### Returns

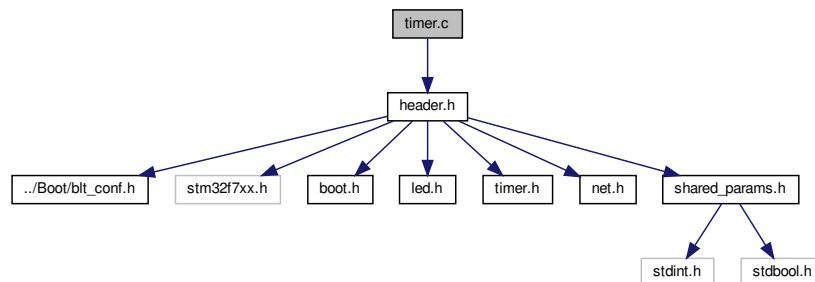
none.

## 7.1272 timer.c File Reference

Timer driver source file.

```
#include "header.h"
```

Include dependency graph for Demo/ARMCM7\_STM32F7\_Nucleo\_F767ZI\_GCC/Prog/timer.c:



## Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*
- void [SysTick\\_Handler](#) (void)  
*Interrupt service routine of the timer.*

### 7.1272.1 Detailed Description

Timer driver source file.

### 7.1272.2 Function Documentation

#### 7.1272.2.1 SysTick\_Handler()

```
void SysTick_Handler (
 void)
```

Interrupt service routine of the timer.

#### Returns

none.

#### 7.1272.2.2 TimerGet()

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

#### Returns

Current value of the millisecond timer.

## 7.1272.2.3 TimerInit()

```
void TimerInit (
 void)
```

Initializes the timer.

## Returns

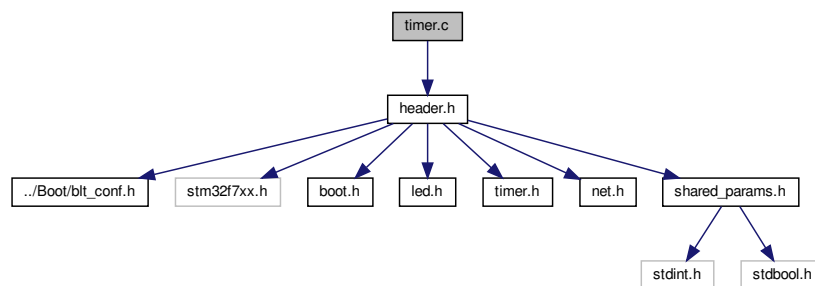
none.

## 7.1273 timer.c File Reference

Timer driver source file.

```
#include "header.h"
```

Include dependency graph for Demo/ARMCM7\_STM32F7\_Nucleo\_F767ZI\_IAR/Prog/timer.c:



## Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*
- void [SysTick\\_Handler](#) (void)  
*Interrupt service routine of the timer.*

## 7.1273.1 Detailed Description

Timer driver source file.

## 7.1273.2 Function Documentation

### 7.1273.2.1 SysTick\_Handler()

```
void SysTick_Handler (
 void)
```

Interrupt service routine of the timer.

#### Returns

none.

### 7.1273.2.2 TimerGet()

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

#### Returns

Current value of the millisecond timer.

### 7.1273.2.3 TimerInit()

```
void TimerInit (
 void)
```

Initializes the timer.

#### Returns

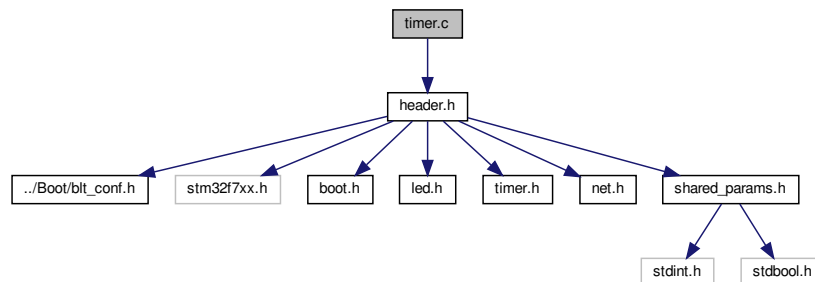
none.

## 7.1274 timer.c File Reference

Timer driver source file.

```
#include "header.h"
```

Include dependency graph for Demo/ARMCM7\_STM32F7\_Nucleo\_F767ZI\_Keil/Prog/timer.c:



## Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*
- void [SysTick\\_Handler](#) (void)  
*Interrupt service routine of the timer.*

### 7.1274.1 Detailed Description

Timer driver source file.

### 7.1274.2 Function Documentation

#### 7.1274.2.1 SysTick\_Handler()

```
void SysTick_Handler (
 void)
```

Interrupt service routine of the timer.

#### Returns

none.

#### 7.1274.2.2 TimerGet()

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

#### Returns

Current value of the millisecond timer.

### 7.1274.2.3 TimerInit()

```
void TimerInit (
 void)
```

Initializes the timer.

#### Returns

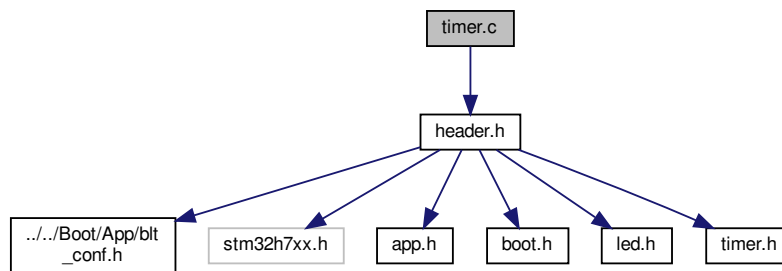
none.

## 7.1275 timer.c File Reference

Timer driver source file.

```
#include "header.h"
```

Include dependency graph for Demo/ARMCM7\_STM32H7\_Nucleo\_H743ZI\_CubeIDE/Prog/App/timer.c:



## Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*

### 7.1275.1 Detailed Description

Timer driver source file.

### 7.1275.2 Function Documentation

### 7.1275.2.1 TimerGet()

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

#### Returns

Current value of the millisecond timer.

### 7.1275.2.2 TimerInit()

```
void TimerInit (
 void)
```

Initializes the timer.

#### Returns

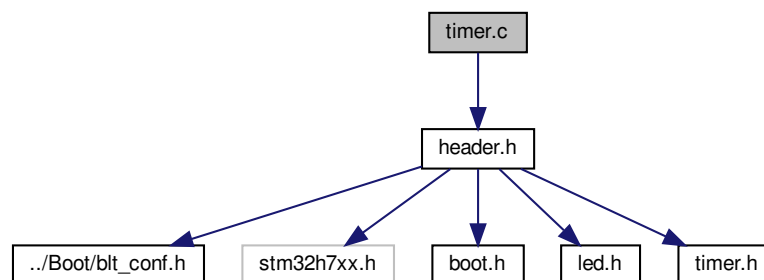
none.

## 7.1276 timer.c File Reference

Timer driver source file.

```
#include "header.h"
```

Include dependency graph for Demo/ARMCM7\_STM32H7\_Nucleo\_H743ZI\_GCC/Prog/timer.c:



## Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*
- void [SysTick\\_Handler](#) (void)  
*Interrupt service routine of the timer.*

### 7.1276.1 Detailed Description

Timer driver source file.

### 7.1276.2 Function Documentation

#### 7.1276.2.1 SysTick\_Handler()

```
void SysTick_Handler (
 void)
```

Interrupt service routine of the timer.

##### Returns

none.

#### 7.1276.2.2 TimerGet()

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

##### Returns

Current value of the millisecond timer.

#### 7.1276.2.3 TimerInit()

```
void TimerInit (
 void)
```

Initializes the timer.

##### Returns

none.

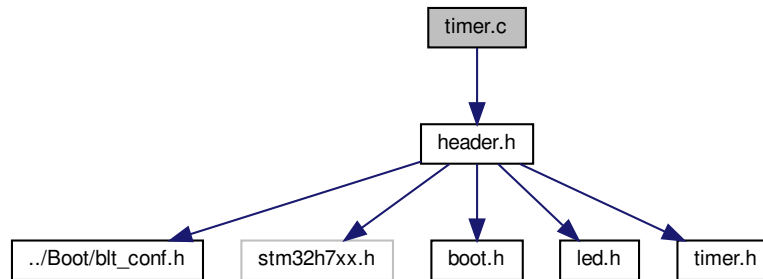


## 7.1277 timer.c File Reference

Timer driver source file.

```
#include "header.h"
```

Include dependency graph for Demo/ARMCM7\_STM32H7\_Nucleo\_H743ZI\_IAR/Prog/timer.c:



### Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*
- void [SysTick\\_Handler](#) (void)  
*Interrupt service routine of the timer.*

### 7.1277.1 Detailed Description

Timer driver source file.

### 7.1277.2 Function Documentation

#### 7.1277.2.1 SysTick\_Handler()

```
void SysTick_Handler (
 void)
```

Interrupt service routine of the timer.

#### Returns

none.

### 7.1277.2.2 TimerGet()

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

#### Returns

Current value of the millisecond timer.

### 7.1277.2.3 TimerInit()

```
void TimerInit (
 void)
```

Initializes the timer.

#### Returns

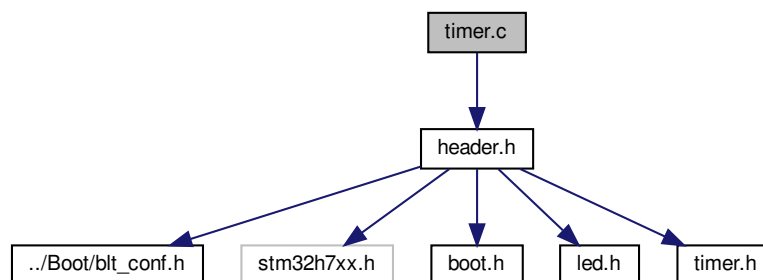
none.

## 7.1278 timer.c File Reference

Timer driver source file.

```
#include "header.h"
```

Include dependency graph for Demo/ARMCM7\_STM32H7\_Nucleo\_H743ZI\_Keil/Prog/timer.c:



## Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*
- void [SysTick\\_Handler](#) (void)  
*Interrupt service routine of the timer.*

## 7.1278.1 Detailed Description

Timer driver source file.

## 7.1278.2 Function Documentation

### 7.1278.2.1 SysTick\_Handler()

```
void SysTick_Handler (
 void)
```

Interrupt service routine of the timer.

#### Returns

none.

### 7.1278.2.2 TimerGet()

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

#### Returns

Current value of the millisecond timer.

### 7.1278.2.3 TimerInit()

```
void TimerInit (
 void)
```

Initializes the timer.

#### Returns

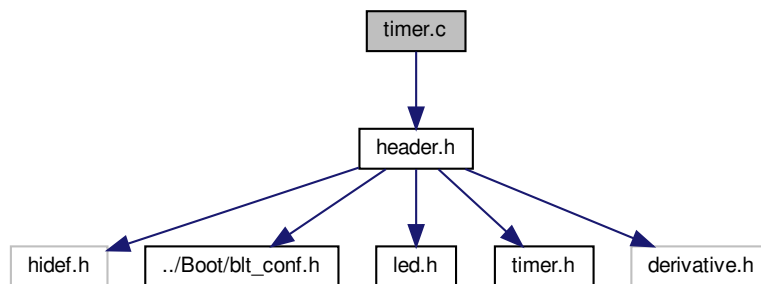
none.

## 7.1279 timer.c File Reference

Timer driver source file.

```
#include "header.h"
```

Include dependency graph for Demo/HCS12\_DevKit\_S12G128\_CodeWarrior/Prog/timer.c:



### Macros

- `#define` [TIMER\\_COUNTS\\_PER\\_MS](#) (24000)  
*Number of free running timer counts in 1 millisecond.*

### Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- void [TimerDeinit](#) (void)  
*Stops and disables the timer.*
- void [TimerSet](#) (unsigned long timer\_value)  
*Sets the initial counter value of the millisecond timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*
- `__interrupt` void [TimerISRHandler](#) (void)  
*Interrupt service routine of the timer.*

### Variables

- static unsigned long [millisecond\\_counter](#)  
*Local variable for storing the number of milliseconds that have elapsed since startup.*

#### 7.1279.1 Detailed Description

Timer driver source file.

## 7.1279.2 Function Documentation

### 7.1279.2.1 TimerDeinit()

```
void TimerDeinit (
 void)
```

Stops and disables the timer.

#### Returns

none.

Referenced by TimerInit().

### 7.1279.2.2 TimerGet()

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

#### Returns

Current value of the millisecond timer.

### 7.1279.2.3 TimerInit()

```
void TimerInit (
 void)
```

Initializes the timer.

#### Returns

none.

### 7.1279.2.4 TimerISRHandler()

```
__interrupt void TimerISRHandler (
 void)
```

Interrupt service routine of the timer.

#### Returns

none.

### 7.1279.2.5 TimerSet()

```
void TimerSet (
 unsigned long timer_value)
```

Sets the initial counter value of the millisecond timer.

**Parameters**

|                          |                                            |
|--------------------------|--------------------------------------------|
| <code>timer_value</code> | initialize value of the millisecond timer. |
|--------------------------|--------------------------------------------|

**Returns**

none.

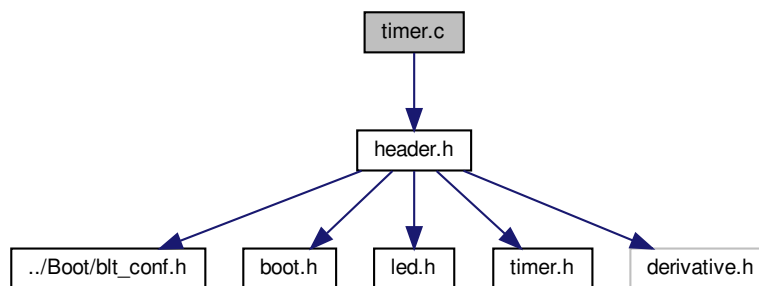
Referenced by TimerInit().

**7.1280 timer.c File Reference**

Timer driver source file.

```
#include "header.h"
```

Include dependency graph for Demo/HCS12\_Evbplus\_Dragon12p\_CodeWarrior/Prog/timer.c:

**Macros**

- `#define` [TIMER\\_COUNTS\\_PER\\_MS](#) ([BOOT\\_CPU\\_SYSTEM\\_SPEED\\_KHZ](#))  
*Number of free running timer counts in 1 millisecond.*

**Functions**

- void [TimerInit](#) (void)  
*Initializes the timer.*
- void [TimerDeinit](#) (void)  
*Stops and disables the timer.*
- void [TimerSet](#) (unsigned long timer\_value)  
*Sets the initial counter value of the millisecond timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*
- `__interrupt` void [TimerISRHandler](#) (void)  
*Interrupt service routine of the timer.*

## Variables

- static unsigned long [millisecond\\_counter](#)  
*Local variable for storing the number of milliseconds that have elapsed since startup.*

## 7.1280.1 Detailed Description

Timer driver source file.

## 7.1280.2 Function Documentation

### 7.1280.2.1 TimerDeinit()

```
void TimerDeinit (
 void)
```

Stops and disables the timer.

#### Returns

none.

Referenced by TimerInit().

### 7.1280.2.2 TimerGet()

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

#### Returns

Current value of the millisecond timer.

### 7.1280.2.3 TimerInit()

```
void TimerInit (
 void)
```

Initializes the timer.

#### Returns

none.

#### 7.1280.2.4 TimerISRHandler()

```
__interrupt void TimerISRHandler (
 void)
```

Interrupt service routine of the timer.

##### Returns

none.

#### 7.1280.2.5 TimerSet()

```
void TimerSet (
 unsigned long timer_value)
```

Sets the initial counter value of the millisecond timer.

##### Parameters

|                    |                                            |
|--------------------|--------------------------------------------|
| <i>timer_value</i> | initialize value of the millisecond timer. |
|--------------------|--------------------------------------------|

##### Returns

none.

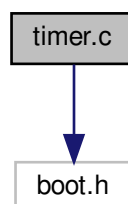
Referenced by TimerInit().

### 7.1281 timer.c File Reference

Bootloader timer driver source file.

```
#include "boot.h"
```

Include dependency graph for Source/\_template/timer.c:





## Functions

- void [TimerInit](#) (void)  
*Initializes the polling based millisecond timer driver.*
- void [TimerReset](#) (void)  
*Reset the timer by placing the timer back into it's default reset configuration.*
- void [TimerUpdate](#) (void)  
*Updates the millisecond timer.*
- [blt\\_int32u](#) [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*

## Variables

- static [blt\\_int32u](#) [millisecond\\_counter](#)  
*Local variable for storing the number of milliseconds that have elapsed since startup.*

### 7.1281.1 Detailed Description

Bootloader timer driver source file.

### 7.1281.2 Function Documentation

#### 7.1281.2.1 TimerGet()

```
blt_int32u TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

#### Returns

Current value of the millisecond timer.

#### 7.1281.2.2 TimerInit()

```
void TimerInit (
 void)
```

Initializes the polling based millisecond timer driver.

Initializes the timer.

#### Returns

none.

### 7.1281.2.3 TimerReset()

```
void TimerReset (
 void)
```

Reset the timer by placing the timer back into it's default reset configuration.

#### Returns

none.

Referenced by CpuStartUserProgram(), and TimerInit().

### 7.1281.2.4 TimerUpdate()

```
void TimerUpdate (
 void)
```

Updates the millisecond timer.

#### Returns

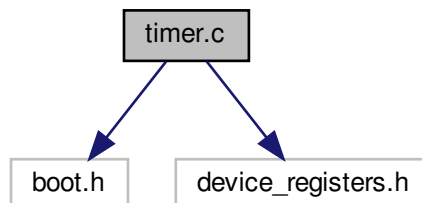
none.

Referenced by BootTask(), and TimerGet().

## 7.1282 timer.c File Reference

Bootloader timer driver source file.

```
#include "boot.h"
#include "device_registers.h"
Include dependency graph for Source/ARMCM0_S32K11/timer.c:
```



## Functions

- void [TimerInit](#) (void)  
*Initializes the polling based millisecond timer driver.*
- void [TimerReset](#) (void)  
*Reset the timer by placing the timer back into it's default reset configuration.*
- void [TimerUpdate](#) (void)  
*Updates the millisecond timer.*
- [blt\\_int32u](#) [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*

## Variables

- static [blt\\_int32u](#) [millisecond\\_counter](#)  
*Local variable for storing the number of milliseconds that have elapsed since startup.*

### 7.1282.1 Detailed Description

Bootloader timer driver source file.

### 7.1282.2 Function Documentation

#### 7.1282.2.1 [TimerGet\(\)](#)

```
blt_int32u TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

#### Returns

Current value of the millisecond timer.

#### 7.1282.2.2 [TimerInit\(\)](#)

```
void TimerInit (
 void)
```

Initializes the polling based millisecond timer driver.

Initializes the timer.

#### Returns

none.

### 7.1282.2.3 TimerReset()

```
void TimerReset (
 void)
```

Reset the timer by placing the timer back into it's default reset configuration.

#### Returns

none.

Referenced by TimerInit().

### 7.1282.2.4 TimerUpdate()

```
void TimerUpdate (
 void)
```

Updates the millisecond timer.

#### Returns

none.

Referenced by TimerGet().

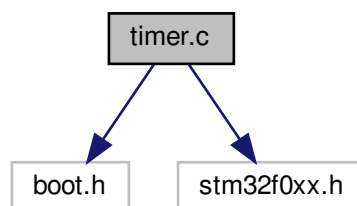
## 7.1283 timer.c File Reference

Bootloader timer driver source file.

```
#include "boot.h"
```

```
#include "stm32f0xx.h"
```

Include dependency graph for Source/ARMCM0\_STM32F0/timer.c:



## Functions

- void [TimerInit](#) (void)  
*Initializes the polling based millisecond timer driver.*
- void [TimerReset](#) (void)  
*Reset the timer by placing the timer back into it's default reset configuration.*
- void [TimerUpdate](#) (void)  
*Updates the millisecond timer.*
- [blt\\_int32u](#) [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*
- [uint32\\_t](#) [HAL\\_GetTick](#) (void)  
*Override for the HAL driver's GetTick() functionality. This is needed because the bootloader doesn't use interrupts, but the HAL's tick functionality assumes that it does. This will cause the HAL\_Delay() function to not work properly. As a result of this override, the HAL's tick functionality works in polling mode.*
- [\\_\\_weak](#) void [SysTick\\_Handler](#) (void)  
*This function handles the SysTick interrupt. The HAL driver is initialized before this timer driver. The HAL driver configures the SysTick for interrupt driven mode, which is afterwards disabled by the timer driver initialization. It is theoretically possible that the SysTick interrupt still fires before the timer driver disables it. Therefore the handler is implemented here. If not, then the default handler from cstart.s is used, which hangs the system.*

## Variables

- static [blt\\_int32u](#) [millisecond\\_counter](#)  
*Local variable for storing the number of milliseconds that have elapsed since startup.*

### 7.1283.1 Detailed Description

Bootloader timer driver source file.

### 7.1283.2 Function Documentation

#### 7.1283.2.1 HAL\_GetTick()

```
uint32_t HAL_GetTick (
 void)
```

Override for the HAL driver's GetTick() functionality. This is needed because the bootloader doesn't use interrupts, but the HAL's tick functionality assumes that it does. This will cause the HAL\_Delay() function to not work properly. As a result of this override, the HAL's tick functionality works in polling mode.

#### Returns

Current value of the millisecond timer.

Referenced by [TimerGet](#)().

### 7.1283.2.2 SysTick\_Handler()

```
__weak void SysTick_Handler (
 void)
```

This function handles the SysTick interrupt. The HAL driver is initialized before this timer driver. The HAL driver configures the SysTick for interrupt driven mode, which is afterwards disabled by the timer driver initialization. It is theoretically possible that the SysTick interrupt still fires before the timer driver disables it. Therefore the handler is implemented here. If not, then the default handler from cstart.s is used, which hangs the system.

#### Returns

none.

### 7.1283.2.3 TimerGet()

```
blt_int32u TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

#### Returns

Current value of the millisecond timer.

Referenced by HAL\_GetTick().

### 7.1283.2.4 TimerInit()

```
void TimerInit (
 void)
```

Initializes the polling based millisecond timer driver.

Initializes the timer.

#### Returns

none.

### 7.1283.2.5 TimerReset()

```
void TimerReset (
 void)
```

Reset the timer by placing the timer back into it's default reset configuration.

#### Returns

none.

Referenced by TimerInit().

### 7.1283.2.6 TimerUpdate()

```
void TimerUpdate (
 void)
```

Updates the millisecond timer.

#### Returns

none.

Referenced by TimerGet().

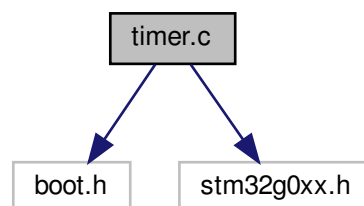
## 7.1284 timer.c File Reference

Bootloader timer driver source file.

```
#include "boot.h"
```

```
#include "stm32g0xx.h"
```

Include dependency graph for Source/ARMCM0\_STM32G0/timer.c:



## Functions

- void [TimerInit](#) (void)  
*Initializes the polling based millisecond timer driver.*
- void [TimerReset](#) (void)  
*Reset the timer by placing the timer back into it's default reset configuration.*
- void [TimerUpdate](#) (void)  
*Updates the millisecond timer.*
- [blt\\_int32u](#) [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*
- [uint32\\_t](#) [HAL\\_GetTick](#) (void)  
*Override for the HAL driver's GetTick() functionality. This is needed because the bootloader doesn't use interrupts, but the HAL's tick functionality assumes that it does. This will cause the HAL\_Delay() function to not work properly. As a result of this override, the HAL's tick functionality works in polling mode.*
- [\\_\\_weak](#) void [SysTick\\_Handler](#) (void)  
*This function handles the SysTick interrupt. The HAL driver is initialized before this timer driver. The HAL driver configures the SysTick for interrupt driven mode, which is afterwards disabled by the timer driver initialization. It is theoretically possible that the SysTick interrupt still fires before the timer driver disables it. Therefore the handler is implemented here. If not, then the default handler from cstart.s is used, which hangs the system.*

## Variables

- static [blt\\_int32u](#) [millisecond\\_counter](#)  
*Local variable for storing the number of milliseconds that have elapsed since startup.*

### 7.1284.1 Detailed Description

Bootloader timer driver source file.

### 7.1284.2 Function Documentation

#### 7.1284.2.1 HAL\_GetTick()

```
uint32_t HAL_GetTick (
 void)
```

Override for the HAL driver's GetTick() functionality. This is needed because the bootloader doesn't use interrupts, but the HAL's tick functionality assumes that it does. This will cause the HAL\_Delay() function to not work properly. As a result of this override, the HAL's tick functionality works in polling mode.

#### Returns

Current value of the millisecond timer.



### 7.1284.2.2 SysTick\_Handler()

```
__weak void SysTick_Handler (
 void)
```

This function handles the SysTick interrupt. The HAL driver is initialized before this timer driver. The HAL driver configures the SysTick for interrupt driven mode, which is afterwards disabled by the timer driver initialization. It is theoretically possible that the SysTick interrupt still fires before the timer driver disables it. Therefore the handler is implemented here. If not, then the default handler from cstart.s is used, which hangs the system.

#### Returns

none.

### 7.1284.2.3 TimerGet()

```
blt_int32u TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

#### Returns

Current value of the millisecond timer.

Referenced by HAL\_GetTick().

### 7.1284.2.4 TimerInit()

```
void TimerInit (
 void)
```

Initializes the polling based millisecond timer driver.

Initializes the timer.

#### Returns

none.

### 7.1284.2.5 TimerReset()

```
void TimerReset (
 void)
```

Reset the timer by placing the timer back into it's default reset configuration.

#### Returns

none.

Referenced by TimerInit().

### 7.1284.2.6 TimerUpdate()

```
void TimerUpdate (
 void)
```

Updates the millisecond timer.

#### Returns

none.

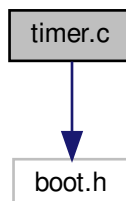
Referenced by TimerGet().

## 7.1285 timer.c File Reference

Bootloader timer driver source file.

```
#include "boot.h"
```

Include dependency graph for Source/ARMCM0\_XMC1/timer.c:



## Data Structures

- struct [tSysTickRegs](#)

*Systick registers.*

## Macros

- #define [SYSTICK\\_BIT\\_CLKSOURCE](#) (([blt\\_int32u](#))0x00000004)  
*CLKSOURCE bit of the system tick.*
- #define [SYSTICK\\_BIT\\_ENABLE](#) (([blt\\_int32u](#))0x00000001)  
*ENABLE bit of the system tick.*
- #define [SYSTICK\\_BIT\\_COUNTERFLAG](#) (([blt\\_int32u](#))0x00010000)  
*COUNTERFLAG bit of the system tick.*
- #define [SYSTICK](#) (([tSysTickRegs](#) \*) ([blt\\_int32u](#))0xE000E010)  
*Macro to access the system tick registers.*

## Functions

- void [TimerInit](#) (void)  
*Initializes the polling based millisecond timer driver.*
- void [TimerReset](#) (void)  
*Reset the timer by placing the timer back into it's default reset configuration.*
- void [TimerUpdate](#) (void)  
*Updates the millisecond timer.*
- [blt\\_int32u](#) [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*

## Variables

- static [blt\\_int32u](#) [millisecond\\_counter](#)  
*Local variable for storing the number of milliseconds that have elapsed since startup.*

### 7.1285.1 Detailed Description

Bootloader timer driver source file.

### 7.1285.2 Function Documentation

#### 7.1285.2.1 TimerGet()

```
blt_int32u TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

##### Returns

Current value of the millisecond timer.

#### 7.1285.2.2 TimerInit()

```
void TimerInit (
 void)
```

Initializes the polling based millisecond timer driver.

Initializes the timer.

##### Returns

none.

#### 7.1285.2.3 TimerReset()

```
void TimerReset (
 void)
```

Reset the timer by placing the timer back into it's default reset configuration.

##### Returns

none.

Referenced by TimerInit().

#### 7.1285.2.4 TimerUpdate()

```
void TimerUpdate (
 void)
```

Updates the millisecond timer.

##### Returns

none.

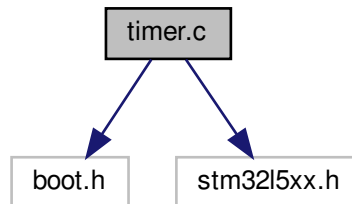
Referenced by TimerGet().

## 7.1286 timer.c File Reference

Bootloader timer driver source file.

```
#include "boot.h"
#include "stm32l5xx.h"
```

Include dependency graph for Source/ARMCM33\_STM32L5/timer.c:



### Functions

- void [TimerInit](#) (void)  
*Initializes the polling based millisecond timer driver.*
- void [TimerReset](#) (void)  
*Reset the timer by placing the timer back into it's default reset configuration.*
- void [TimerUpdate](#) (void)  
*Updates the millisecond timer.*
- [blt\\_int32u](#) [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*
- [uint32\\_t](#) [HAL\\_GetTick](#) (void)  
*Override for the HAL driver's GetTick() functionality. This is needed because the bootloader doesn't use interrupts, but the HAL's tick functionality assumes that it does. This will cause the HAL\_Delay() function to not work properly. As a result of this override, the HAL's tick functionality works in polling mode.*
- [\\_\\_weak](#) void [SysTick\\_Handler](#) (void)  
*This function handles the SysTick interrupt. The HAL driver is initialized before this timer driver. The HAL driver configures the SysTick for interrupt driven mode, which is afterwards disabled by the timer driver initialization. It is theoretically possible that the SysTick interrupt still fires before the timer driver disables it. Therefore the handler is implemented here. If not, then the default handler from cstart.s is used, which hangs the system.*

### Variables

- static [blt\\_int32u](#) [millisecond\\_counter](#)  
*Local variable for storing the number of milliseconds that have elapsed since startup.*

#### 7.1286.1 Detailed Description

Bootloader timer driver source file.

## 7.1286.2 Function Documentation

### 7.1286.2.1 HAL\_GetTick()

```
uint32_t HAL_GetTick (
 void)
```

Override for the HAL driver's GetTick() functionality. This is needed because the bootloader doesn't use interrupts, but the HAL's tick functionality assumes that it does. This will cause the HAL\_Delay() function to not work properly. As a result of this override, the HAL's tick functionality works in polling mode.

#### Returns

Current value of the millisecond timer.

### 7.1286.2.2 SysTick\_Handler()

```
__weak void SysTick_Handler (
 void)
```

This function handles the SysTick interrupt. The HAL driver is initialized before this timer driver. The HAL driver configures the SysTick for interrupt driven mode, which is afterwards disabled by the timer driver initialization. It is theoretically possible that the SysTick interrupt still fires before the timer driver disables it. Therefore the handler is implemented here. If not, then the default handler from cstart.s is used, which hangs the system.

#### Returns

none.

### 7.1286.2.3 TimerGet()

```
blt_int32u TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

#### Returns

Current value of the millisecond timer.

Referenced by HAL\_GetTick().

#### 7.1286.2.4 TimerInit()

```
void TimerInit (
 void)
```

Initializes the polling based millisecond timer driver.

Initializes the timer.

##### Returns

none.

#### 7.1286.2.5 TimerReset()

```
void TimerReset (
 void)
```

Reset the timer by placing the timer back into it's default reset configuration.

##### Returns

none.

Referenced by TimerInit().

#### 7.1286.2.6 TimerUpdate()

```
void TimerUpdate (
 void)
```

Updates the millisecond timer.

##### Returns

none.

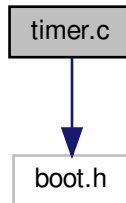
Referenced by TimerGet().

## 7.1287 timer.c File Reference

Bootloader timer driver source file.

```
#include "boot.h"
```

Include dependency graph for Source/ARMCM3\_EFM32/timer.c:



### Data Structures

- struct `tSysTickRegs`  
*Systick registers.*

### Macros

- #define `SYSTICK_BIT_CLKSOURCE` `((blt_int32u)0x00000004)`  
*CLKSOURCE bit of the system tick.*
- #define `SYSTICK_BIT_ENABLE` `((blt_int32u)0x00000001)`  
*ENABLE bit of the system tick.*
- #define `SYSTICK_BIT_COUNTERFLAG` `((blt_int32u)0x00010000)`  
*COUNTERFLAG bit of the system tick.*
- #define `SYSTICK` `((tSysTickRegs *) (blt_int32u)0xE00E010)`  
*Macro to access the system tick registers.*

### Functions

- void `TimerInit` (void)  
*Initializes the polling based millisecond timer driver.*
- void `TimerReset` (void)  
*Reset the timer by placing the timer back into it's default reset configuration.*
- void `TimerUpdate` (void)  
*Updates the millisecond timer.*
- `blt_int32u` `TimerGet` (void)  
*Obtains the counter value of the millisecond timer.*



## Variables

- static `blt_int32u millisecond_counter`

*Local variable for storing the number of milliseconds that have elapsed since startup.*

### 7.1287.1 Detailed Description

Bootloader timer driver source file.

### 7.1287.2 Function Documentation

#### 7.1287.2.1 TimerGet()

```
blt_int32u TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

#### Returns

Current value of the millisecond timer.

#### 7.1287.2.2 TimerInit()

```
void TimerInit (
 void)
```

Initializes the polling based millisecond timer driver.

Initializes the timer.

#### Returns

none.

### 7.1287.2.3 TimerReset()

```
void TimerReset (
 void)
```

Reset the timer by placing the timer back into it's default reset configuration.

#### Returns

none.

Referenced by TimerInit().

### 7.1287.2.4 TimerUpdate()

```
void TimerUpdate (
 void)
```

Updates the millisecond timer.

#### Returns

none.

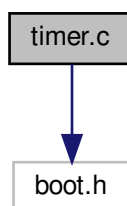
Referenced by TimerGet().

## 7.1288 timer.c File Reference

Bootloader timer driver source file.

```
#include "boot.h"
```

Include dependency graph for Source/ARMCM3\_LM3S/timer.c:



## Data Structures

- struct [tSysTickRegs](#)  
*Systick registers.*

## Macros

- #define [SYSTICK\\_BIT\\_CLKSOURCE](#) (([blt\\_int32u](#))0x00000004)  
*CLKSOURCE bit of the system tick.*
- #define [SYSTICK\\_BIT\\_ENABLE](#) (([blt\\_int32u](#))0x00000001)  
*ENABLE bit of the system tick.*
- #define [SYSTICK\\_BIT\\_COUNTERFLAG](#) (([blt\\_int32u](#))0x00010000)  
*COUNTERFLAG bit of the system tick.*
- #define [SYSTICK](#) (([tSysTickRegs](#) \*) ([blt\\_int32u](#))0xE000E010)  
*Macro to access the system tick registers.*

## Functions

- void [TimerInit](#) (void)  
*Initializes the polling based millisecond timer driver.*
- void [TimerReset](#) (void)  
*Reset the timer by placing the timer back into it's default reset configuration.*
- void [TimerUpdate](#) (void)  
*Updates the millisecond timer.*
- [blt\\_int32u](#) [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*

## Variables

- static [blt\\_int32u](#) [millisecond\\_counter](#)  
*Local variable for storing the number of milliseconds that have elapsed since startup.*

### 7.1288.1 Detailed Description

Bootloader timer driver source file.

### 7.1288.2 Function Documentation

#### 7.1288.2.1 TimerGet()

```
blt_int32u TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

##### Returns

Current value of the millisecond timer.

#### 7.1288.2.2 TimerInit()

```
void TimerInit (
 void)
```

Initializes the polling based millisecond timer driver.

Initializes the timer.

##### Returns

none.

#### 7.1288.2.3 TimerReset()

```
void TimerReset (
 void)
```

Reset the timer by placing the timer back into it's default reset configuration.

##### Returns

none.

Referenced by TimerInit().

#### 7.1288.2.4 TimerUpdate()

```
void TimerUpdate (
 void)
```

Updates the millisecond timer.

##### Returns

none.

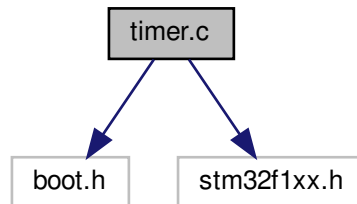
Referenced by TimerGet().

## 7.1289 timer.c File Reference

Bootloader timer driver source file.

```
#include "boot.h"
#include "stm32f1xx.h"
```

Include dependency graph for Source/ARMCM3\_STM32F1/timer.c:



### Functions

- void [TimerInit](#) (void)  
*Initializes the polling based millisecond timer driver.*
- void [TimerReset](#) (void)  
*Reset the timer by placing the timer back into it's default reset configuration.*
- void [TimerUpdate](#) (void)  
*Updates the millisecond timer.*
- [blt\\_int32u](#) [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*
- [uint32\\_t](#) [HAL\\_GetTick](#) (void)  
*Override for the HAL driver's GetTick() functionality. This is needed because the bootloader doesn't use interrupts, but the HAL's tick functionality assumes that it does. This will cause the HAL\_Delay() function to not work properly. As a result of this override, the HAL's tick functionality works in polling mode.*
- [\\_\\_weak](#) void [SysTick\\_Handler](#) (void)  
*This function handles the SysTick interrupt. The HAL driver is initialized before this timer driver. The HAL driver configures the SysTick for interrupt driven mode, which is afterwards disabled by the timer driver initialization. It is theoretically possible that the SysTick interrupt still fires before the timer driver disables it. Therefore the handler is implemented here. If not, then the default handler from cstart.s is used, which hangs the system.*

### Variables

- static [blt\\_int32u](#) [millisecond\\_counter](#)  
*Local variable for storing the number of milliseconds that have elapsed since startup.*

#### 7.1289.1 Detailed Description

Bootloader timer driver source file.

## 7.1289.2 Function Documentation

### 7.1289.2.1 HAL\_GetTick()

```
uint32_t HAL_GetTick (
 void)
```

Override for the HAL driver's GetTick() functionality. This is needed because the bootloader doesn't use interrupts, but the HAL's tick functionality assumes that it does. This will cause the HAL\_Delay() function to not work properly. As a result of this override, the HAL's tick functionality works in polling mode.

#### Returns

Current value of the millisecond timer.

### 7.1289.2.2 SysTick\_Handler()

```
__weak void SysTick_Handler (
 void)
```

This function handles the SysTick interrupt. The HAL driver is initialized before this timer driver. The HAL driver configures the SysTick for interrupt driven mode, which is afterwards disabled by the timer driver initialization. It is theoretically possible that the SysTick interrupt still fires before the timer driver disables it. Therefore the handler is implemented here. If not, then the default handler from cstart.s is used, which hangs the system.

#### Returns

none.

### 7.1289.2.3 TimerGet()

```
blt_int32u TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

#### Returns

Current value of the millisecond timer.

Referenced by HAL\_GetTick().

#### 7.1289.2.4 TimerInit()

```
void TimerInit (
 void)
```

Initializes the polling based millisecond timer driver.

Initializes the timer.

##### Returns

none.

#### 7.1289.2.5 TimerReset()

```
void TimerReset (
 void)
```

Reset the timer by placing the timer back into it's default reset configuration.

##### Returns

none.

Referenced by TimerInit().

#### 7.1289.2.6 TimerUpdate()

```
void TimerUpdate (
 void)
```

Updates the millisecond timer.

##### Returns

none.

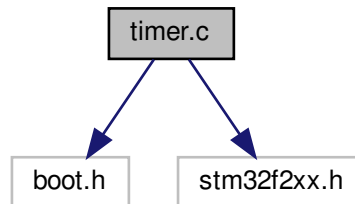
Referenced by TimerGet().

## 7.1290 timer.c File Reference

Bootloader timer driver source file.

```
#include "boot.h"
#include "stm32f2xx.h"
```

Include dependency graph for Source/ARMCM3\_STM32F2/timer.c:



### Functions

- void [TimerInit](#) (void)  
*Initializes the polling based millisecond timer driver.*
- void [TimerReset](#) (void)  
*Reset the timer by placing the timer back into it's default reset configuration.*
- void [TimerUpdate](#) (void)  
*Updates the millisecond timer.*
- [blt\\_int32u](#) [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*
- [uint32\\_t](#) [HAL\\_GetTick](#) (void)  
*Override for the HAL driver's GetTick() functionality. This is needed because the bootloader doesn't use interrupts, but the HAL's tick functionality assumes that it does. This will cause the HAL\_Delay() function to not work properly. As a result of this override, the HAL's tick functionality works in polling mode.*
- [\\_\\_weak](#) void [SysTick\\_Handler](#) (void)  
*This function handles the SysTick interrupt. The HAL driver is initialized before this timer driver. The HAL driver configures the SysTick for interrupt driven mode, which is afterwards disabled by the timer driver initialization. It is theoretically possible that the SysTick interrupt still fires before the timer driver disables it. Therefore the handler is implemented here. If not, then the default handler from cstart.s is used, which hangs the system.*

### Variables

- static [blt\\_int32u](#) [millisecond\\_counter](#)  
*Local variable for storing the number of milliseconds that have elapsed since startup.*

#### 7.1290.1 Detailed Description

Bootloader timer driver source file.



## 7.1290.2 Function Documentation

### 7.1290.2.1 HAL\_GetTick()

```
uint32_t HAL_GetTick (
 void)
```

Override for the HAL driver's GetTick() functionality. This is needed because the bootloader doesn't use interrupts, but the HAL's tick functionality assumes that it does. This will cause the HAL\_Delay() function to not work properly. As a result of this override, the HAL's tick functionality works in polling mode.

#### Returns

Current value of the millisecond timer.

### 7.1290.2.2 SysTick\_Handler()

```
__weak void SysTick_Handler (
 void)
```

This function handles the SysTick interrupt. The HAL driver is initialized before this timer driver. The HAL driver configures the SysTick for interrupt driven mode, which is afterwards disabled by the timer driver initialization. It is theoretically possible that the SysTick interrupt still fires before the timer driver disables it. Therefore the handler is implemented here. If not, then the default handler from cstart.s is used, which hangs the system.

#### Returns

none.

### 7.1290.2.3 TimerGet()

```
blt_int32u TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

#### Returns

Current value of the millisecond timer.

Referenced by HAL\_GetTick().

#### 7.1290.2.4 TimerInit()

```
void TimerInit (
 void)
```

Initializes the polling based millisecond timer driver.

Initializes the timer.

##### Returns

none.

#### 7.1290.2.5 TimerReset()

```
void TimerReset (
 void)
```

Reset the timer by placing the timer back into it's default reset configuration.

##### Returns

none.

Referenced by TimerInit().

#### 7.1290.2.6 TimerUpdate()

```
void TimerUpdate (
 void)
```

Updates the millisecond timer.

##### Returns

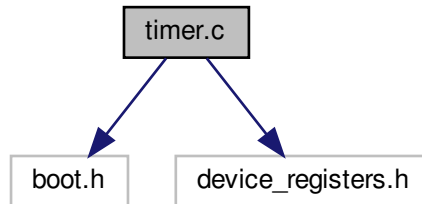
none.

Referenced by TimerGet().

## 7.1291 timer.c File Reference

Bootloader timer driver source file.

```
#include "boot.h"
#include "device_registers.h"
Include dependency graph for Source/ARMCM4_S32K14/timer.c:
```



### Functions

- void [TimerInit](#) (void)  
*Initializes the polling based millisecond timer driver.*
- void [TimerReset](#) (void)  
*Reset the timer by placing the timer back into it's default reset configuration.*
- void [TimerUpdate](#) (void)  
*Updates the millisecond timer.*
- [blt\\_int32u](#) [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*

### Variables

- static [blt\\_int32u](#) [millisecond\\_counter](#)  
*Local variable for storing the number of milliseconds that have elapsed since startup.*

#### 7.1291.1 Detailed Description

Bootloader timer driver source file.

#### 7.1291.2 Function Documentation

#### 7.1291.2.1 TimerGet()

```
blt_int32u TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

##### Returns

Current value of the millisecond timer.

#### 7.1291.2.2 TimerInit()

```
void TimerInit (
 void)
```

Initializes the polling based millisecond timer driver.

Initializes the timer.

##### Returns

none.

#### 7.1291.2.3 TimerReset()

```
void TimerReset (
 void)
```

Reset the timer by placing the timer back into it's default reset configuration.

##### Returns

none.

Referenced by TimerInit().

#### 7.1291.2.4 TimerUpdate()

```
void TimerUpdate (
 void)
```

Updates the millisecond timer.

##### Returns

none.

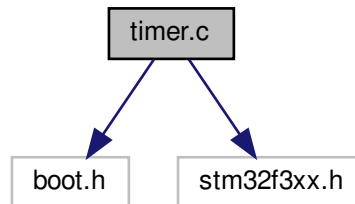
Referenced by TimerGet().

## 7.1292 timer.c File Reference

Bootloader timer driver source file.

```
#include "boot.h"
#include "stm32f3xx.h"
```

Include dependency graph for Source/ARMCM4\_STM32F3/timer.c:



### Functions

- void [TimerInit](#) (void)  
*Initializes the polling based millisecond timer driver.*
- void [TimerReset](#) (void)  
*Reset the timer by placing the timer back into it's default reset configuration.*
- void [TimerUpdate](#) (void)  
*Updates the millisecond timer.*
- [blt\\_int32u](#) [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*
- [uint32\\_t](#) [HAL\\_GetTick](#) (void)  
*Override for the HAL driver's GetTick() functionality. This is needed because the bootloader doesn't use interrupts, but the HAL's tick functionality assumes that it does. This will cause the HAL\_Delay() function to not work properly. As a result of this override, the HAL's tick functionality works in polling mode.*
- [\\_\\_weak](#) void [SysTick\\_Handler](#) (void)  
*This function handles the SysTick interrupt. The HAL driver is initialized before this timer driver. The HAL driver configures the SysTick for interrupt driven mode, which is afterwards disabled by the timer driver initialization. It is theoretically possible that the SysTick interrupt still fires before the timer driver disables it. Therefore the handler is implemented here. If not, then the default handler from cstart.s is used, which hangs the system.*

### Variables

- static [blt\\_int32u](#) [millisecond\\_counter](#)  
*Local variable for storing the number of milliseconds that have elapsed since startup.*

#### 7.1292.1 Detailed Description

Bootloader timer driver source file.

## 7.1292.2 Function Documentation

### 7.1292.2.1 HAL\_GetTick()

```
uint32_t HAL_GetTick (
 void)
```

Override for the HAL driver's GetTick() functionality. This is needed because the bootloader doesn't use interrupts, but the HAL's tick functionality assumes that it does. This will cause the HAL\_Delay() function to not work properly. As a result of this override, the HAL's tick functionality works in polling mode.

#### Returns

Current value of the millisecond timer.

### 7.1292.2.2 SysTick\_Handler()

```
__weak void SysTick_Handler (
 void)
```

This function handles the SysTick interrupt. The HAL driver is initialized before this timer driver. The HAL driver configures the SysTick for interrupt driven mode, which is afterwards disabled by the timer driver initialization. It is theoretically possible that the SysTick interrupt still fires before the timer driver disables it. Therefore the handler is implemented here. If not, then the default handler from cstart.s is used, which hangs the system.

#### Returns

none.

### 7.1292.2.3 TimerGet()

```
blt_int32u TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

#### Returns

Current value of the millisecond timer.

Referenced by HAL\_GetTick().

#### 7.1292.2.4 TimerInit()

```
void TimerInit (
 void)
```

Initializes the polling based millisecond timer driver.

Initializes the timer.

##### Returns

none.

#### 7.1292.2.5 TimerReset()

```
void TimerReset (
 void)
```

Reset the timer by placing the timer back into it's default reset configuration.

##### Returns

none.

Referenced by TimerInit().

#### 7.1292.2.6 TimerUpdate()

```
void TimerUpdate (
 void)
```

Updates the millisecond timer.

##### Returns

none.

Referenced by TimerGet().

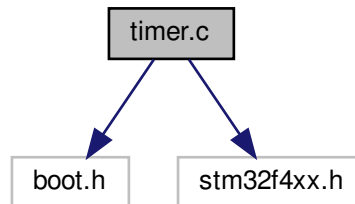
## 7.1293 timer.c File Reference

Bootloader timer driver source file.

```
#include "boot.h"
```

```
#include "stm32f4xx.h"
```

Include dependency graph for Source/ARMCM4\_STM32F4/timer.c:



### Functions

- void [TimerInit](#) (void)  
*Initializes the polling based millisecond timer driver.*
- void [TimerReset](#) (void)  
*Reset the timer by placing the timer back into it's default reset configuration.*
- void [TimerUpdate](#) (void)  
*Updates the millisecond timer.*
- [blt\\_int32u](#) [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*
- [uint32\\_t](#) [HAL\\_GetTick](#) (void)  
*Override for the HAL driver's GetTick() functionality. This is needed because the bootloader doesn't use interrupts, but the HAL's tick functionality assumes that it does. This will cause the HAL\_Delay() function to not work properly. As a result of this override, the HAL's tick functionality works in polling mode.*
- [\\_\\_weak](#) void [SysTick\\_Handler](#) (void)  
*This function handles the SysTick interrupt. The HAL driver is initialized before this timer driver. The HAL driver configures the SysTick for interrupt driven mode, which is afterwards disabled by the timer driver initialization. It is theoretically possible that the SysTick interrupt still fires before the timer driver disables it. Therefore the handler is implemented here. If not, then the default handler from cstart.s is used, which hangs the system.*

### Variables

- static [blt\\_int32u](#) [millisecond\\_counter](#)  
*Local variable for storing the number of milliseconds that have elapsed since startup.*

#### 7.1293.1 Detailed Description

Bootloader timer driver source file.



## 7.1293.2 Function Documentation

### 7.1293.2.1 HAL\_GetTick()

```
uint32_t HAL_GetTick (
 void)
```

Override for the HAL driver's GetTick() functionality. This is needed because the bootloader doesn't use interrupts, but the HAL's tick functionality assumes that it does. This will cause the HAL\_Delay() function to not work properly. As a result of this override, the HAL's tick functionality works in polling mode.

#### Returns

Current value of the millisecond timer.

### 7.1293.2.2 SysTick\_Handler()

```
__weak void SysTick_Handler (
 void)
```

This function handles the SysTick interrupt. The HAL driver is initialized before this timer driver. The HAL driver configures the SysTick for interrupt driven mode, which is afterwards disabled by the timer driver initialization. It is theoretically possible that the SysTick interrupt still fires before the timer driver disables it. Therefore the handler is implemented here. If not, then the default handler from cstart.s is used, which hangs the system.

#### Returns

none.

### 7.1293.2.3 TimerGet()

```
blt_int32u TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

#### Returns

Current value of the millisecond timer.

Referenced by HAL\_GetTick().

#### 7.1293.2.4 TimerInit()

```
void TimerInit (
 void)
```

Initializes the polling based millisecond timer driver.

Initializes the timer.

##### Returns

none.

#### 7.1293.2.5 TimerReset()

```
void TimerReset (
 void)
```

Reset the timer by placing the timer back into it's default reset configuration.

##### Returns

none.

Referenced by TimerInit().

#### 7.1293.2.6 TimerUpdate()

```
void TimerUpdate (
 void)
```

Updates the millisecond timer.

##### Returns

none.

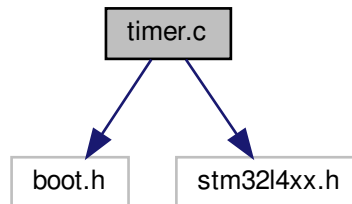
Referenced by TimerGet().

## 7.1294 timer.c File Reference

Bootloader timer driver source file.

```
#include "boot.h"
#include "stm32l4xx.h"
```

Include dependency graph for Source/ARMCM4\_STM32L4/timer.c:



### Functions

- void [TimerInit](#) (void)  
*Initializes the polling based millisecond timer driver.*
- void [TimerReset](#) (void)  
*Reset the timer by placing the timer back into it's default reset configuration.*
- void [TimerUpdate](#) (void)  
*Updates the millisecond timer.*
- `blt_int32u` [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*
- `uint32_t` [HAL\\_GetTick](#) (void)  
*Override for the HAL driver's GetTick() functionality. This is needed because the bootloader doesn't use interrupts, but the HAL's tick functionality assumes that it does. This will cause the HAL\_Delay() function to not work properly. As a result of this override, the HAL's tick functionality works in polling mode.*
- `__weak` void [SysTick\\_Handler](#) (void)  
*This function handles the SysTick interrupt. The HAL driver is initialized before this timer driver. The HAL driver configures the SysTick for interrupt driven mode, which is afterwards disabled by the timer driver initialization. It is theoretically possible that the SysTick interrupt still fires before the timer driver disables it. Therefore the handler is implemented here. If not, then the default handler from cstart.s is used, which hangs the system.*

### Variables

- static `blt_int32u` [millisecond\\_counter](#)  
*Local variable for storing the number of milliseconds that have elapsed since startup.*

#### 7.1294.1 Detailed Description

Bootloader timer driver source file.

## 7.1294.2 Function Documentation

### 7.1294.2.1 HAL\_GetTick()

```
uint32_t HAL_GetTick (
 void)
```

Override for the HAL driver's GetTick() functionality. This is needed because the bootloader doesn't use interrupts, but the HAL's tick functionality assumes that it does. This will cause the HAL\_Delay() function to not work properly. As a result of this override, the HAL's tick functionality works in polling mode.

#### Returns

Current value of the millisecond timer.

### 7.1294.2.2 SysTick\_Handler()

```
__weak void SysTick_Handler (
 void)
```

This function handles the SysTick interrupt. The HAL driver is initialized before this timer driver. The HAL driver configures the SysTick for interrupt driven mode, which is afterwards disabled by the timer driver initialization. It is theoretically possible that the SysTick interrupt still fires before the timer driver disables it. Therefore the handler is implemented here. If not, then the default handler from cstart.s is used, which hangs the system.

#### Returns

none.

### 7.1294.2.3 TimerGet()

```
blt_int32u TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

#### Returns

Current value of the millisecond timer.

Referenced by HAL\_GetTick().

#### 7.1294.2.4 TimerInit()

```
void TimerInit (
 void)
```

Initializes the polling based millisecond timer driver.

Initializes the timer.

##### Returns

none.

#### 7.1294.2.5 TimerReset()

```
void TimerReset (
 void)
```

Reset the timer by placing the timer back into it's default reset configuration.

##### Returns

none.

Referenced by TimerInit().

#### 7.1294.2.6 TimerUpdate()

```
void TimerUpdate (
 void)
```

Updates the millisecond timer.

##### Returns

none.

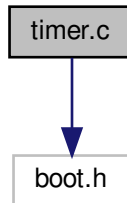
Referenced by TimerGet().

## 7.1295 timer.c File Reference

Bootloader timer driver source file.

```
#include "boot.h"
```

Include dependency graph for Source/ARMCM4\_TM4C/timer.c:



### Data Structures

- struct `tSysTickRegs`  
*Systick registers.*

### Macros

- #define `SYSTICK_BIT_CLKSOURCE` `((blt_int32u)0x00000004)`  
*CLKSOURCE bit of the system tick.*
- #define `SYSTICK_BIT_ENABLE` `((blt_int32u)0x00000001)`  
*ENABLE bit of the system tick.*
- #define `SYSTICK_BIT_COUNTERFLAG` `((blt_int32u)0x00010000)`  
*COUNTERFLAG bit of the system tick.*
- #define `SYSTICK` `((tSysTickRegs *) (blt_int32u)0xE00E010)`  
*Macro to access the system tick registers.*

### Functions

- void `TimerInit` (void)  
*Initializes the polling based millisecond timer driver.*
- void `TimerReset` (void)  
*Reset the timer by placing the timer back into it's default reset configuration.*
- void `TimerUpdate` (void)  
*Updates the millisecond timer.*
- `blt_int32u` `TimerGet` (void)  
*Obtains the counter value of the millisecond timer.*

## Variables

- static `blt_int32u millisecond_counter`

*Local variable for storing the number of milliseconds that have elapsed since startup.*

### 7.1295.1 Detailed Description

Bootloader timer driver source file.

### 7.1295.2 Function Documentation

#### 7.1295.2.1 TimerGet()

```
blt_int32u TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

#### Returns

Current value of the millisecond timer.

#### 7.1295.2.2 TimerInit()

```
void TimerInit (
 void)
```

Initializes the polling based millisecond timer driver.

Initializes the timer.

#### Returns

none.

### 7.1295.2.3 TimerReset()

```
void TimerReset (
 void)
```

Reset the timer by placing the timer back into it's default reset configuration.

#### Returns

none.

Referenced by TimerInit().

### 7.1295.2.4 TimerUpdate()

```
void TimerUpdate (
 void)
```

Updates the millisecond timer.

#### Returns

none.

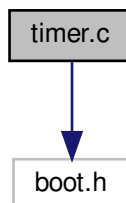
Referenced by TimerGet().

## 7.1296 timer.c File Reference

Bootloader timer driver source file.

```
#include "boot.h"
```

Include dependency graph for Source/ARMCM4\_XMC4/timer.c:





## Data Structures

- struct [tSysTickRegs](#)  
*Systick registers.*

## Macros

- #define [SYSTICK\\_BIT\\_CLKSOURCE](#) (([blt\\_int32u](#))0x00000004)  
*CLKSOURCE bit of the system tick.*
- #define [SYSTICK\\_BIT\\_ENABLE](#) (([blt\\_int32u](#))0x00000001)  
*ENABLE bit of the system tick.*
- #define [SYSTICK\\_BIT\\_COUNTERFLAG](#) (([blt\\_int32u](#))0x00010000)  
*COUNTERFLAG bit of the system tick.*
- #define [SYSTICK](#) (([tSysTickRegs](#) \*) ([blt\\_int32u](#))0xE000E010)  
*Macro to access the system tick registers.*

## Functions

- void [TimerInit](#) (void)  
*Initializes the polling based millisecond timer driver.*
- void [TimerReset](#) (void)  
*Reset the timer by placing the timer back into it's default reset configuration.*
- void [TimerUpdate](#) (void)  
*Updates the millisecond timer.*
- [blt\\_int32u](#) [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*

## Variables

- static [blt\\_int32u](#) [millisecond\\_counter](#)  
*Local variable for storing the number of milliseconds that have elapsed since startup.*

### 7.1296.1 Detailed Description

Bootloader timer driver source file.

### 7.1296.2 Function Documentation

#### 7.1296.2.1 TimerGet()

```
blt_int32u TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

##### Returns

Current value of the millisecond timer.

#### 7.1296.2.2 TimerInit()

```
void TimerInit (
 void)
```

Initializes the polling based millisecond timer driver.

Initializes the timer.

##### Returns

none.

#### 7.1296.2.3 TimerReset()

```
void TimerReset (
 void)
```

Reset the timer by placing the timer back into it's default reset configuration.

##### Returns

none.

Referenced by TimerInit().

#### 7.1296.2.4 TimerUpdate()

```
void TimerUpdate (
 void)
```

Updates the millisecond timer.

##### Returns

none.

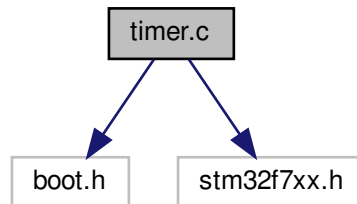
Referenced by TimerGet().

## 7.1297 timer.c File Reference

Bootloader timer driver source file.

```
#include "boot.h"
#include "stm32f7xx.h"
```

Include dependency graph for Source/ARMCM7\_STM32F7/timer.c:



### Functions

- void [TimerInit](#) (void)  
*Initializes the polling based millisecond timer driver.*
- void [TimerReset](#) (void)  
*Reset the timer by placing the timer back into it's default reset configuration.*
- void [TimerUpdate](#) (void)  
*Updates the millisecond timer.*
- [blt\\_int32u](#) [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*
- [uint32\\_t](#) [HAL\\_GetTick](#) (void)  
*Override for the HAL driver's GetTick() functionality. This is needed because the bootloader doesn't use interrupts, but the HAL's tick functionality assumes that it does. This will cause the HAL\_Delay() function to not work properly. As a result of this override, the HAL's tick functionality works in polling mode.*
- [\\_\\_weak](#) void [SysTick\\_Handler](#) (void)  
*This function handles the SysTick interrupt. The HAL driver is initialized before this timer driver. The HAL driver configures the SysTick for interrupt driven mode, which is afterwards disabled by the timer driver initialization. It is theoretically possible that the SysTick interrupt still fires before the timer driver disables it. Therefore the handler is implemented here. If not, then the default handler from cstart.s is used, which hangs the system.*

### Variables

- static [blt\\_int32u](#) [millisecond\\_counter](#)  
*Local variable for storing the number of milliseconds that have elapsed since startup.*

#### 7.1297.1 Detailed Description

Bootloader timer driver source file.

## 7.1297.2 Function Documentation

### 7.1297.2.1 HAL\_GetTick()

```
uint32_t HAL_GetTick (
 void)
```

Override for the HAL driver's GetTick() functionality. This is needed because the bootloader doesn't use interrupts, but the HAL's tick functionality assumes that it does. This will cause the HAL\_Delay() function to not work properly. As a result of this override, the HAL's tick functionality works in polling mode.

#### Returns

Current value of the millisecond timer.

### 7.1297.2.2 SysTick\_Handler()

```
__weak void SysTick_Handler (
 void)
```

This function handles the SysTick interrupt. The HAL driver is initialized before this timer driver. The HAL driver configures the SysTick for interrupt driven mode, which is afterwards disabled by the timer driver initialization. It is theoretically possible that the SysTick interrupt still fires before the timer driver disables it. Therefore the handler is implemented here. If not, then the default handler from cstart.s is used, which hangs the system.

#### Returns

none.

### 7.1297.2.3 TimerGet()

```
blt_int32u TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

#### Returns

Current value of the millisecond timer.

Referenced by HAL\_GetTick().

#### 7.1297.2.4 TimerInit()

```
void TimerInit (
 void)
```

Initializes the polling based millisecond timer driver.

Initializes the timer.

##### Returns

none.

#### 7.1297.2.5 TimerReset()

```
void TimerReset (
 void)
```

Reset the timer by placing the timer back into it's default reset configuration.

##### Returns

none.

Referenced by TimerInit().

#### 7.1297.2.6 TimerUpdate()

```
void TimerUpdate (
 void)
```

Updates the millisecond timer.

##### Returns

none.

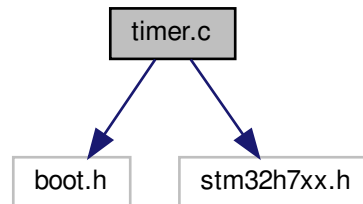
Referenced by TimerGet().

## 7.1298 timer.c File Reference

Bootloader timer driver source file.

```
#include "boot.h"
#include "stm32h7xx.h"
```

Include dependency graph for Source/ARMCM7\_STM32H7/timer.c:



### Functions

- void [TimerInit](#) (void)  
*Initializes the polling based millisecond timer driver.*
- void [TimerReset](#) (void)  
*Reset the timer by placing the timer back into it's default reset configuration.*
- void [TimerUpdate](#) (void)  
*Updates the millisecond timer.*
- [blt\\_int32u](#) [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*
- [uint32\\_t](#) [HAL\\_GetTick](#) (void)  
*Override for the HAL driver's GetTick() functionality. This is needed because the bootloader doesn't use interrupts, but the HAL's tick functionality assumes that it does. This will cause the HAL\_Delay() function to not work properly. As a result of this override, the HAL's tick functionality works in polling mode.*
- [\\_\\_weak](#) void [SysTick\\_Handler](#) (void)  
*This function handles the SysTick interrupt. The HAL driver is initialized before this timer driver. The HAL driver configures the SysTick for interrupt driven mode, which is afterwards disabled by the timer driver initialization. It is theoretically possible that the SysTick interrupt still fires before the timer driver disables it. Therefore the handler is implemented here. If not, then the default handler from cstart.s is used, which hangs the system.*

### Variables

- static [blt\\_int32u](#) [millisecond\\_counter](#)  
*Local variable for storing the number of milliseconds that have elapsed since startup.*

#### 7.1298.1 Detailed Description

Bootloader timer driver source file.

## 7.1298.2 Function Documentation

### 7.1298.2.1 HAL\_GetTick()

```
uint32_t HAL_GetTick (
 void)
```

Override for the HAL driver's GetTick() functionality. This is needed because the bootloader doesn't use interrupts, but the HAL's tick functionality assumes that it does. This will cause the HAL\_Delay() function to not work properly. As a result of this override, the HAL's tick functionality works in polling mode.

#### Returns

Current value of the millisecond timer.

### 7.1298.2.2 SysTick\_Handler()

```
__weak void SysTick_Handler (
 void)
```

This function handles the SysTick interrupt. The HAL driver is initialized before this timer driver. The HAL driver configures the SysTick for interrupt driven mode, which is afterwards disabled by the timer driver initialization. It is theoretically possible that the SysTick interrupt still fires before the timer driver disables it. Therefore the handler is implemented here. If not, then the default handler from cstart.s is used, which hangs the system.

#### Returns

none.

### 7.1298.2.3 TimerGet()

```
blt_int32u TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

#### Returns

Current value of the millisecond timer.

Referenced by HAL\_GetTick().

#### 7.1298.2.4 TimerInit()

```
void TimerInit (
 void)
```

Initializes the polling based millisecond timer driver.

Initializes the timer.

##### Returns

none.

#### 7.1298.2.5 TimerReset()

```
void TimerReset (
 void)
```

Reset the timer by placing the timer back into it's default reset configuration.

##### Returns

none.

Referenced by TimerInit().

#### 7.1298.2.6 TimerUpdate()

```
void TimerUpdate (
 void)
```

Updates the millisecond timer.

##### Returns

none.

Referenced by TimerGet().

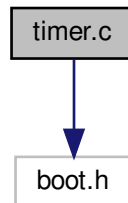


## 7.1299 timer.c File Reference

Bootloader timer driver source file.

```
#include "boot.h"
```

Include dependency graph for Source/HCS12/timer.c:



### Data Structures

- struct [tTimerRegs](#)  
*Structure type with the layout of the timer related control registers.*

### Macros

- #define [TIMER\\_REGS\\_BASE\\_ADDRESS](#) (0x0040)  
*Base address for the timer related control registers.*
- #define [TIMER](#) ((volatile [tTimerRegs](#) \*)[TIMER\\_REGS\\_BASE\\_ADDRESS](#))  
*Macro for accessing the flash related control registers.*
- #define [TIMER\\_COUNTS\\_PER\\_MS](#) ([BOOT\\_CPU\\_SYSTEM\\_SPEED\\_KHZ](#))  
*Number of free running counter ticks in one millisecond.*
- #define [TEN\\_BIT](#) (0x80)  
*TSCR1 - timer enable bit.*
- #define [IOS0\\_BIT](#) (0x01)  
*TIOS - channel 0 ic/oc configuration bit.*
- #define [C0F\\_BIT](#) (0x01)  
*TFLG1 - channel 0 ic/oc event flag bit.*

### Functions

- void [TimerInit](#) (void)  
*Initializes the polling based millisecond timer driver.*
- void [TimerReset](#) (void)  
*Reset the timer by placing the timer back into it's default reset configuration.*
- void [TimerUpdate](#) (void)  
*Updates the millisecond timer.*
- [blt\\_int32u](#) [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*

## Variables

- static `blt_int32u millisecond_counter`

*Local variable for storing the number of milliseconds that have elapsed since startup.*

## 7.1299.1 Detailed Description

Bootloader timer driver source file.

## 7.1299.2 Function Documentation

### 7.1299.2.1 TimerGet()

```
blt_int32u TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

#### Returns

Current value of the millisecond timer.

### 7.1299.2.2 TimerInit()

```
void TimerInit (
 void)
```

Initializes the polling based millisecond timer driver.

Initializes the timer.

#### Returns

none.

### 7.1299.2.3 TimerReset()

```
void TimerReset (
 void)
```

Reset the timer by placing the timer back into it's default reset configuration.

#### Returns

none.

Referenced by TimerInit().

### 7.1299.2.4 TimerUpdate()

```
void TimerUpdate (
 void)
```

Updates the millisecond timer.

#### Returns

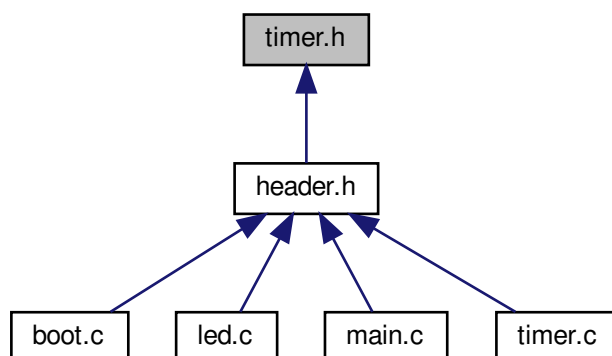
none.

Referenced by TimerGet().

## 7.1300 timer.h File Reference

Timer driver header file.

This graph shows which files directly or indirectly include this file:



## Functions

- void `TimerInit` (void)  
*Initializes the timer.*
- unsigned long `TimerGet` (void)  
*Obtains the counter value of the millisecond timer.*

### 7.1300.1 Detailed Description

Timer driver header file.

### 7.1300.2 Function Documentation

#### 7.1300.2.1 `TimerGet()`

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

##### Returns

Current value of the millisecond timer.

#### 7.1300.2.2 `TimerInit()`

```
void TimerInit (
 void)
```

Initializes the timer.

##### Returns

none.

Initializes the timer.

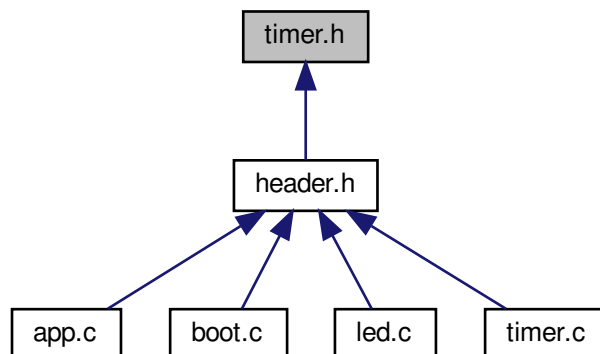
##### Returns

none.

## 7.1301 timer.h File Reference

Timer driver header file.

This graph shows which files directly or indirectly include this file:



### Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*

### 7.1301.1 Detailed Description

Timer driver header file.

### 7.1301.2 Function Documentation

#### 7.1301.2.1 TimerGet()

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

#### Returns

Current value of the millisecond timer.

### 7.1301.2.2 TimerInit()

```
void TimerInit (
 void)
```

Initializes the timer.

#### Returns

none.

Initializes the timer.

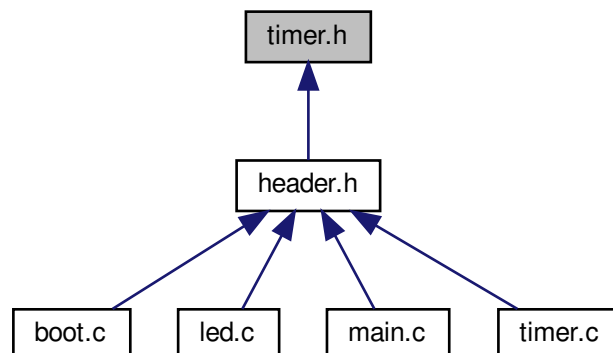
#### Returns

none.

## 7.1302 timer.h File Reference

Timer driver header file.

This graph shows which files directly or indirectly include this file:



## Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*

### 7.1302.1 Detailed Description

Timer driver header file.

## 7.1302.2 Function Documentation

### 7.1302.2.1 TimerGet()

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

#### Returns

Current value of the millisecond timer.

### 7.1302.2.2 TimerInit()

```
void TimerInit (
 void)
```

Initializes the timer.

#### Returns

none.

Initializes the timer.

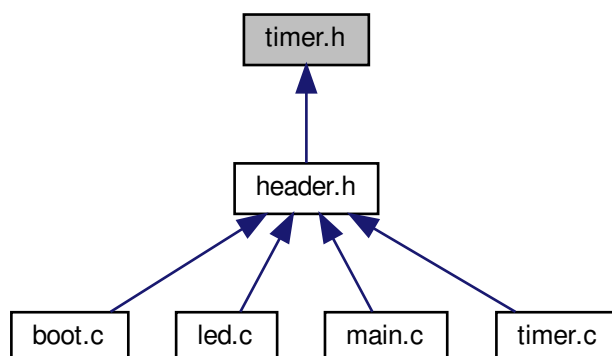
#### Returns

none.

## 7.1303 timer.h File Reference

Timer driver header file.

This graph shows which files directly or indirectly include this file:



## Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*

### 7.1303.1 Detailed Description

Timer driver header file.

### 7.1303.2 Function Documentation

#### 7.1303.2.1 [TimerGet\(\)](#)

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

##### Returns

Current value of the millisecond timer.

#### 7.1303.2.2 [TimerInit\(\)](#)

```
void TimerInit (
 void)
```

Initializes the timer.

##### Returns

none.

Initializes the timer.

##### Returns

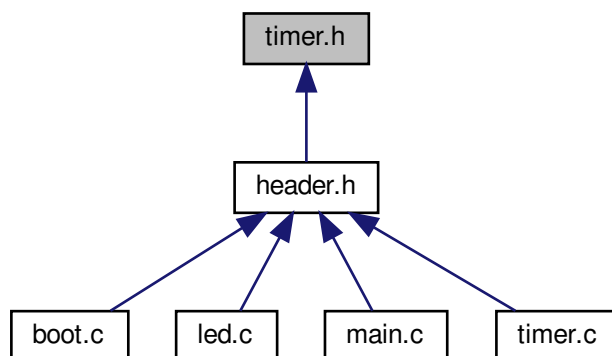
none.



## 7.1304 timer.h File Reference

Timer driver header file.

This graph shows which files directly or indirectly include this file:



### Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*

### 7.1304.1 Detailed Description

Timer driver header file.

### 7.1304.2 Function Documentation

#### 7.1304.2.1 TimerGet()

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

#### Returns

Current value of the millisecond timer.

### 7.1304.2.2 TimerInit()

```
void TimerInit (
 void)
```

Initializes the timer.

#### Returns

none.

Initializes the timer.

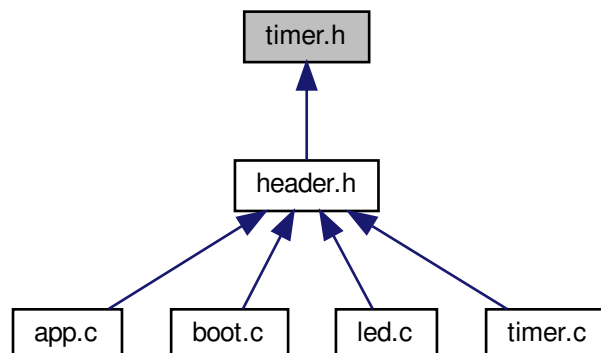
#### Returns

none.

## 7.1305 timer.h File Reference

Timer driver header file.

This graph shows which files directly or indirectly include this file:



## Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*

### 7.1305.1 Detailed Description

Timer driver header file.

## 7.1305.2 Function Documentation

### 7.1305.2.1 TimerGet()

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

#### Returns

Current value of the millisecond timer.

### 7.1305.2.2 TimerInit()

```
void TimerInit (
 void)
```

Initializes the timer.

#### Returns

none.

Initializes the timer.

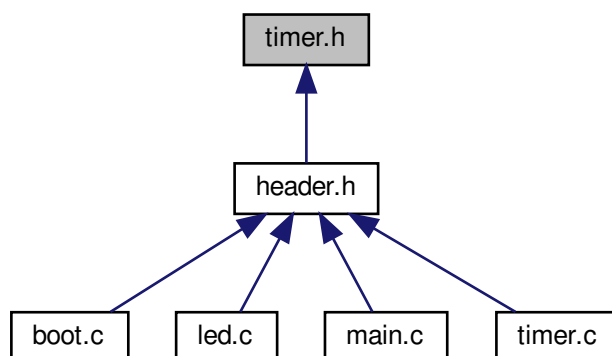
#### Returns

none.

## 7.1306 timer.h File Reference

Timer driver header file.

This graph shows which files directly or indirectly include this file:



## Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*

### 7.1306.1 Detailed Description

Timer driver header file.

### 7.1306.2 Function Documentation

#### 7.1306.2.1 [TimerGet\(\)](#)

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

##### Returns

Current value of the millisecond timer.

#### 7.1306.2.2 [TimerInit\(\)](#)

```
void TimerInit (
 void)
```

Initializes the timer.

##### Returns

none.

Initializes the timer.

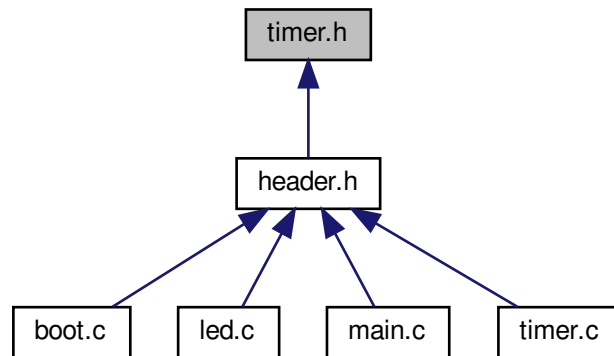
##### Returns

none.

## 7.1307 timer.h File Reference

Timer driver header file.

This graph shows which files directly or indirectly include this file:



### Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*

### 7.1307.1 Detailed Description

Timer driver header file.

### 7.1307.2 Function Documentation

#### 7.1307.2.1 TimerGet()

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

#### Returns

Current value of the millisecond timer.

### 7.1307.2.2 TimerInit()

```
void TimerInit (
 void)
```

Initializes the timer.

#### Returns

none.

Initializes the timer.

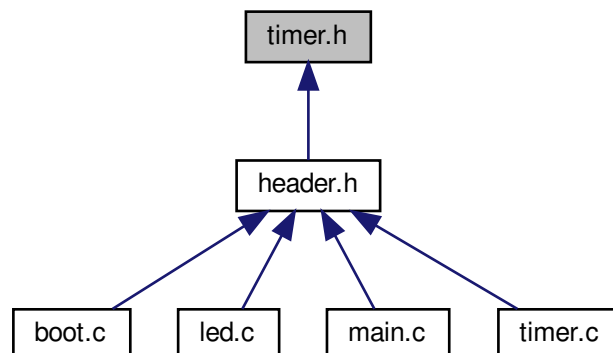
#### Returns

none.

## 7.1308 timer.h File Reference

Timer driver header file.

This graph shows which files directly or indirectly include this file:



## Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*

### 7.1308.1 Detailed Description

Timer driver header file.

## 7.1308.2 Function Documentation

### 7.1308.2.1 TimerGet()

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

#### Returns

Current value of the millisecond timer.

### 7.1308.2.2 TimerInit()

```
void TimerInit (
 void)
```

Initializes the timer.

#### Returns

none.

Initializes the timer.

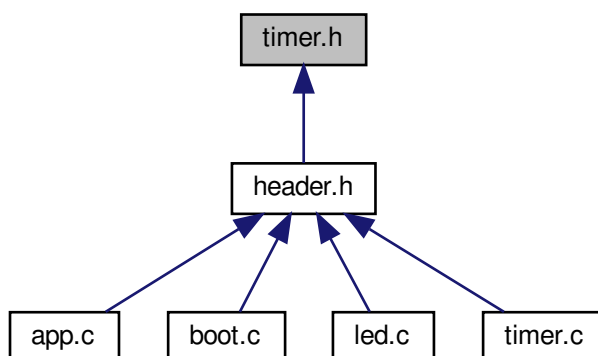
#### Returns

none.

## 7.1309 timer.h File Reference

Timer driver header file.

This graph shows which files directly or indirectly include this file:



## Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*

### 7.1309.1 Detailed Description

Timer driver header file.

### 7.1309.2 Function Documentation

#### 7.1309.2.1 [TimerGet\(\)](#)

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

#### Returns

Current value of the millisecond timer.

#### 7.1309.2.2 [TimerInit\(\)](#)

```
void TimerInit (
 void)
```

Initializes the timer.

#### Returns

none.

Initializes the timer.

#### Returns

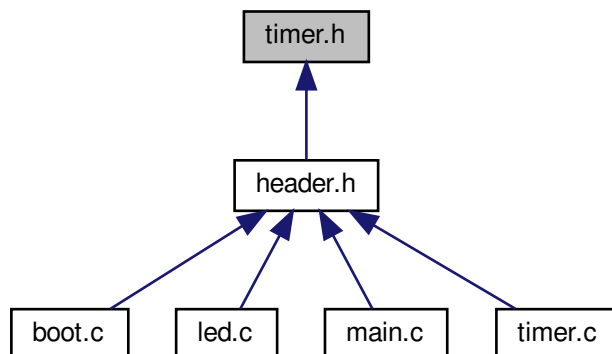
none.



## 7.1310 timer.h File Reference

Timer driver header file.

This graph shows which files directly or indirectly include this file:



### Functions

- void `TimerInit` (void)  
*Initializes the timer.*
- unsigned long `TimerGet` (void)  
*Obtains the counter value of the millisecond timer.*

### 7.1310.1 Detailed Description

Timer driver header file.

### 7.1310.2 Function Documentation

#### 7.1310.2.1 TimerGet()

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

#### Returns

Current value of the millisecond timer.

### 7.1310.2.2 TimerInit()

```
void TimerInit (
 void)
```

Initializes the timer.

#### Returns

none.

Initializes the timer.

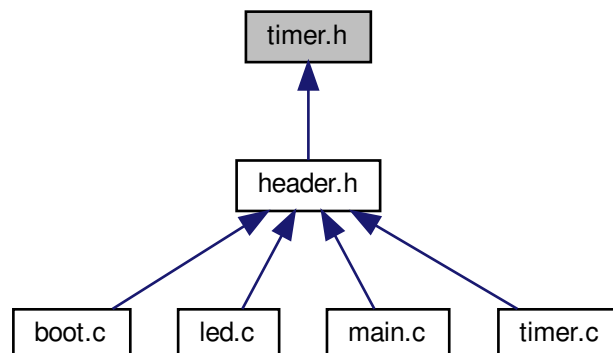
#### Returns

none.

## 7.1311 timer.h File Reference

Timer driver header file.

This graph shows which files directly or indirectly include this file:



## Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*

### 7.1311.1 Detailed Description

Timer driver header file.

## 7.1311.2 Function Documentation

### 7.1311.2.1 TimerGet()

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

#### Returns

Current value of the millisecond timer.

### 7.1311.2.2 TimerInit()

```
void TimerInit (
 void)
```

Initializes the timer.

#### Returns

none.

Initializes the timer.

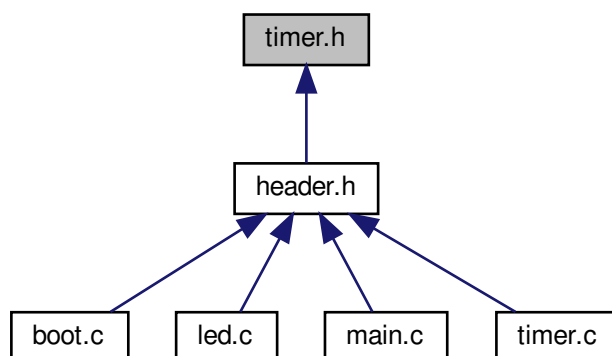
#### Returns

none.

## 7.1312 timer.h File Reference

Timer driver header file.

This graph shows which files directly or indirectly include this file:



## Functions

- void `TimerInit` (void)  
*Initializes the timer.*
- unsigned long `TimerGet` (void)  
*Obtains the counter value of the millisecond timer.*

### 7.1312.1 Detailed Description

Timer driver header file.

### 7.1312.2 Function Documentation

#### 7.1312.2.1 `TimerGet()`

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

##### Returns

Current value of the millisecond timer.

#### 7.1312.2.2 `TimerInit()`

```
void TimerInit (
 void)
```

Initializes the timer.

##### Returns

none.

Initializes the timer.

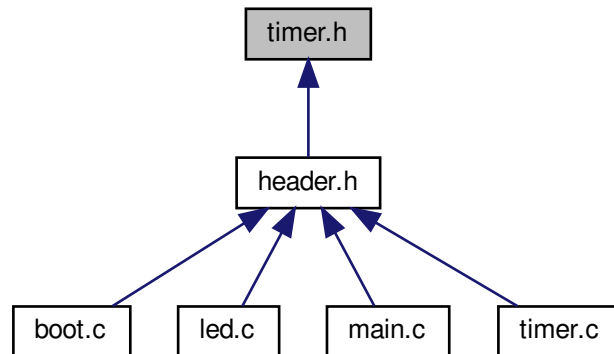
##### Returns

none.

## 7.1313 timer.h File Reference

Timer driver header file.

This graph shows which files directly or indirectly include this file:



### Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- void [TimerDeinit](#) (void)  
*Stops and disables the timer.*
- void [TimerSet](#) (unsigned long timer\_value)  
*Sets the initial counter value of the millisecond timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*

### 7.1313.1 Detailed Description

Timer driver header file.

### 7.1313.2 Function Documentation

#### 7.1313.2.1 TimerDeinit()

```
void TimerDeinit (
 void)
```

Stops and disables the timer.

##### Returns

none.

#### 7.1313.2.2 TimerGet()

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

##### Returns

Current value of the millisecond timer.

#### 7.1313.2.3 TimerInit()

```
void TimerInit (
 void)
```

Initializes the timer.

##### Returns

none.

Initializes the timer.

##### Returns

none.

#### 7.1313.2.4 TimerSet()

```
void TimerSet (
 unsigned long timer_value)
```

Sets the initial counter value of the millisecond timer.

## Parameters

|                          |                                            |
|--------------------------|--------------------------------------------|
| <code>timer_value</code> | initialize value of the millisecond timer. |
|--------------------------|--------------------------------------------|

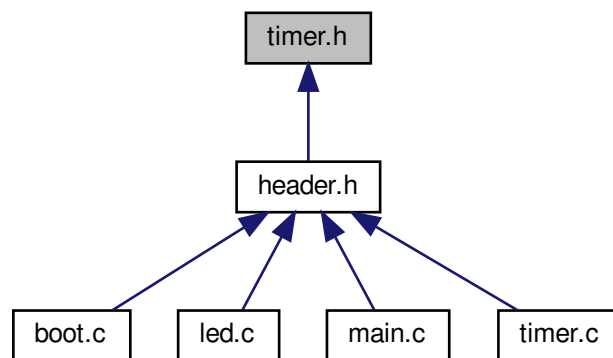
## Returns

none.

## 7.1314 timer.h File Reference

Timer driver header file.

This graph shows which files directly or indirectly include this file:



## Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- void [TimerDeinit](#) (void)  
*Stops and disables the timer.*
- void [TimerSet](#) (unsigned long timer\_value)  
*Sets the initial counter value of the millisecond timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*

### 7.1314.1 Detailed Description

Timer driver header file.

## 7.1314.2 Function Documentation

### 7.1314.2.1 TimerDeinit()

```
void TimerDeinit (
 void)
```

Stops and disables the timer.

#### Returns

none.

### 7.1314.2.2 TimerGet()

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

#### Returns

Current value of the millisecond timer.

### 7.1314.2.3 TimerInit()

```
void TimerInit (
 void)
```

Initializes the timer.

#### Returns

none.

Initializes the timer.

#### Returns

none.

### 7.1314.2.4 TimerSet()

```
void TimerSet (
 unsigned long timer_value)
```

Sets the initial counter value of the millisecond timer.



## Parameters

|                          |                                            |
|--------------------------|--------------------------------------------|
| <code>timer_value</code> | initialize value of the millisecond timer. |
|--------------------------|--------------------------------------------|

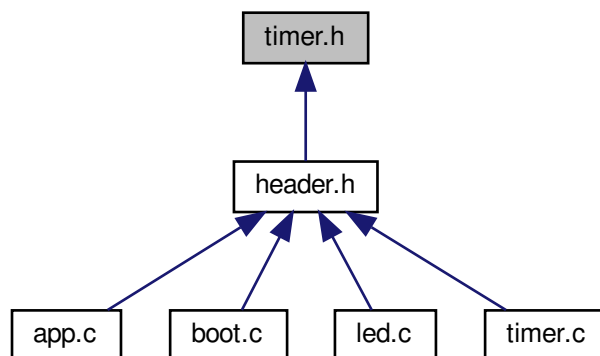
## Returns

none.

## 7.1315 timer.h File Reference

Timer driver header file.

This graph shows which files directly or indirectly include this file:



## Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*

### 7.1315.1 Detailed Description

Timer driver header file.

### 7.1315.2 Function Documentation

### 7.1315.2.1 TimerGet()

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

#### Returns

Current value of the millisecond timer.

### 7.1315.2.2 TimerInit()

```
void TimerInit (
 void)
```

Initializes the timer.

#### Returns

none.

Initializes the timer.

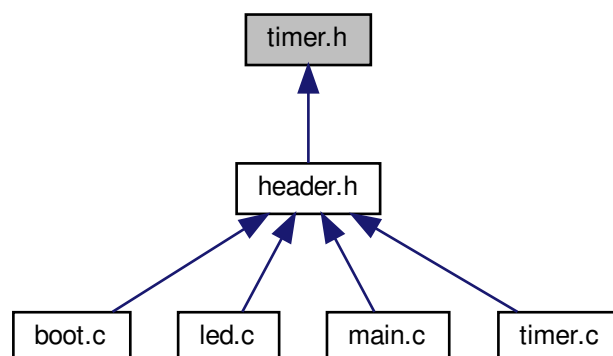
#### Returns

none.

## 7.1316 timer.h File Reference

Timer driver header file.

This graph shows which files directly or indirectly include this file:



## Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*

### 7.1316.1 Detailed Description

Timer driver header file.

### 7.1316.2 Function Documentation

#### 7.1316.2.1 [TimerGet\(\)](#)

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

##### Returns

Current value of the millisecond timer.

#### 7.1316.2.2 [TimerInit\(\)](#)

```
void TimerInit (
 void)
```

Initializes the timer.

##### Returns

none.

Initializes the timer.

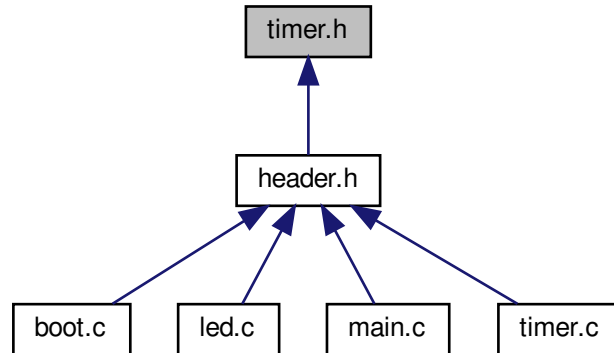
##### Returns

none.

## 7.1317 timer.h File Reference

Timer driver header file.

This graph shows which files directly or indirectly include this file:



### Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*

### 7.1317.1 Detailed Description

Timer driver header file.

### 7.1317.2 Function Documentation

#### 7.1317.2.1 TimerGet()

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

#### Returns

Current value of the millisecond timer.

### 7.1317.2.2 TimerInit()

```
void TimerInit (
 void)
```

Initializes the timer.

#### Returns

none.

Initializes the timer.

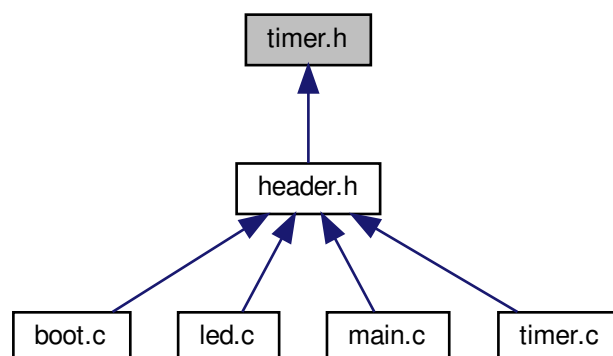
#### Returns

none.

## 7.1318 timer.h File Reference

Timer driver header file.

This graph shows which files directly or indirectly include this file:



### Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*

### 7.1318.1 Detailed Description

Timer driver header file.

## 7.1318.2 Function Documentation

### 7.1318.2.1 TimerGet()

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

#### Returns

Current value of the millisecond timer.

### 7.1318.2.2 TimerInit()

```
void TimerInit (
 void)
```

Initializes the timer.

#### Returns

none.

Initializes the timer.

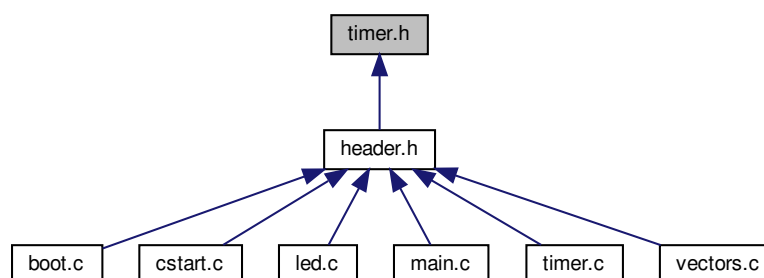
#### Returns

none.

## 7.1319 timer.h File Reference

Timer driver header file.

This graph shows which files directly or indirectly include this file:



## Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- void [TimerDeinit](#) (void)  
*Stops and disables the timer.*
- void [TimerSet](#) (unsigned long timer\_value)  
*Sets the initial counter value of the millisecond timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*
- void [TimerISRHandler](#) (void)  
*Interrupt service routine of the timer.*

### 7.1319.1 Detailed Description

Timer driver header file.

### 7.1319.2 Function Documentation

#### 7.1319.2.1 [TimerDeinit\(\)](#)

```
void TimerDeinit (
 void)
```

Stops and disables the timer.

##### Returns

none.

#### 7.1319.2.2 [TimerGet\(\)](#)

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

##### Returns

Current value of the millisecond timer.

### 7.1319.2.3 TimerInit()

```
void TimerInit (
 void)
```

Initializes the timer.

#### Returns

none.

Initializes the timer.

#### Returns

none.

### 7.1319.2.4 TimerISRHandler()

```
void TimerISRHandler (
 void)
```

Interrupt service routine of the timer.

#### Returns

none.

### 7.1319.2.5 TimerSet()

```
void TimerSet (
 unsigned long timer_value)
```

Sets the initial counter value of the millisecond timer.

#### Parameters

|                    |                                            |
|--------------------|--------------------------------------------|
| <i>timer_value</i> | initialize value of the millisecond timer. |
|--------------------|--------------------------------------------|

#### Returns

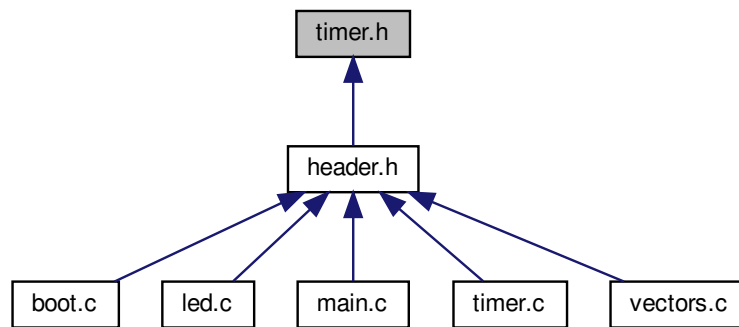
none.



## 7.1320 timer.h File Reference

Timer driver header file.

This graph shows which files directly or indirectly include this file:



### Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- void [TimerDeinit](#) (void)  
*Stops and disables the timer.*
- void [TimerSet](#) (unsigned long timer\_value)  
*Sets the initial counter value of the millisecond timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*
- void [TimerISRHandler](#) (void)  
*Interrupt service routine of the timer.*

### 7.1320.1 Detailed Description

Timer driver header file.

### 7.1320.2 Function Documentation

#### 7.1320.2.1 TimerDeinit()

```
void TimerDeinit (
 void)
```

Stops and disables the timer.

##### Returns

none.

#### 7.1320.2.2 TimerGet()

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

##### Returns

Current value of the millisecond timer.

#### 7.1320.2.3 TimerInit()

```
void TimerInit (
 void)
```

Initializes the timer.

##### Returns

none.

Initializes the timer.

##### Returns

none.

#### 7.1320.2.4 TimerISRHandler()

```
void TimerISRHandler (
 void)
```

Interrupt service routine of the timer.

##### Returns

none.

#### 7.1320.2.5 TimerSet()

```
void TimerSet (
 unsigned long timer_value)
```

Sets the initial counter value of the millisecond timer.

## Parameters

|                          |                                            |
|--------------------------|--------------------------------------------|
| <code>timer_value</code> | initialize value of the millisecond timer. |
|--------------------------|--------------------------------------------|

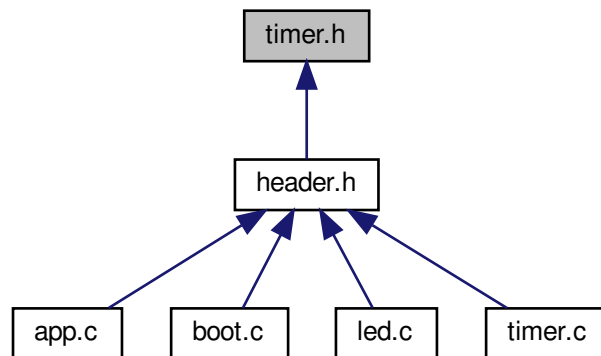
## Returns

none.

## 7.1321 timer.h File Reference

Timer driver header file.

This graph shows which files directly or indirectly include this file:



## Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*

### 7.1321.1 Detailed Description

Timer driver header file.

### 7.1321.2 Function Documentation

### 7.1321.2.1 TimerGet()

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

#### Returns

Current value of the millisecond timer.

### 7.1321.2.2 TimerInit()

```
void TimerInit (
 void)
```

Initializes the timer.

#### Returns

none.

Initializes the timer.

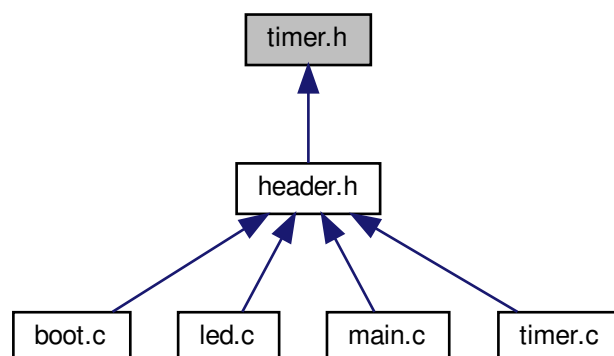
#### Returns

none.

## 7.1322 timer.h File Reference

Timer driver header file.

This graph shows which files directly or indirectly include this file:



## Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*

### 7.1322.1 Detailed Description

Timer driver header file.

### 7.1322.2 Function Documentation

#### 7.1322.2.1 [TimerGet\(\)](#)

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

##### Returns

Current value of the millisecond timer.

#### 7.1322.2.2 [TimerInit\(\)](#)

```
void TimerInit (
 void)
```

Initializes the timer.

##### Returns

none.

Initializes the timer.

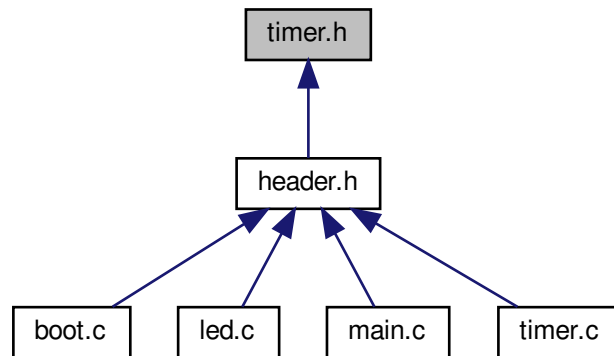
##### Returns

none.

## 7.1323 timer.h File Reference

Timer driver header file.

This graph shows which files directly or indirectly include this file:



### Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*

### 7.1323.1 Detailed Description

Timer driver header file.

### 7.1323.2 Function Documentation

#### 7.1323.2.1 TimerGet()

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

#### Returns

Current value of the millisecond timer.

### 7.1323.2.2 TimerInit()

```
void TimerInit (
 void)
```

Initializes the timer.

#### Returns

none.

Initializes the timer.

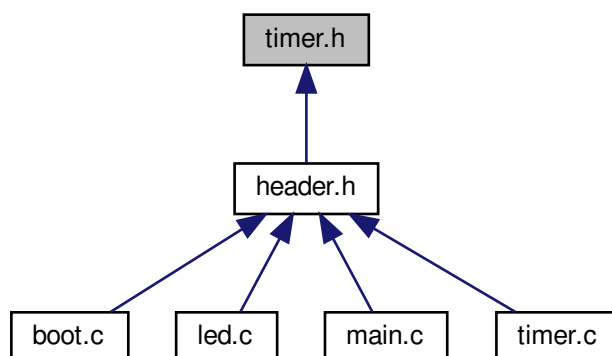
#### Returns

none.

## 7.1324 timer.h File Reference

Timer driver header file.

This graph shows which files directly or indirectly include this file:



## Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*

### 7.1324.1 Detailed Description

Timer driver header file.

## 7.1324.2 Function Documentation

### 7.1324.2.1 TimerGet()

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

#### Returns

Current value of the millisecond timer.

### 7.1324.2.2 TimerInit()

```
void TimerInit (
 void)
```

Initializes the timer.

#### Returns

none.

Initializes the timer.

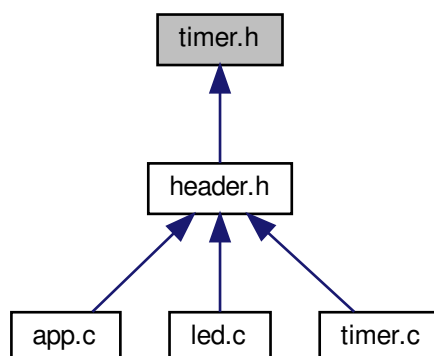
#### Returns

none.

## 7.1325 timer.h File Reference

Timer driver header file.

This graph shows which files directly or indirectly include this file:





## Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*

### 7.1325.1 Detailed Description

Timer driver header file.

### 7.1325.2 Function Documentation

#### 7.1325.2.1 [TimerGet\(\)](#)

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

##### Returns

Current value of the millisecond timer.

#### 7.1325.2.2 [TimerInit\(\)](#)

```
void TimerInit (
 void)
```

Initializes the timer.

##### Returns

none.

Initializes the timer.

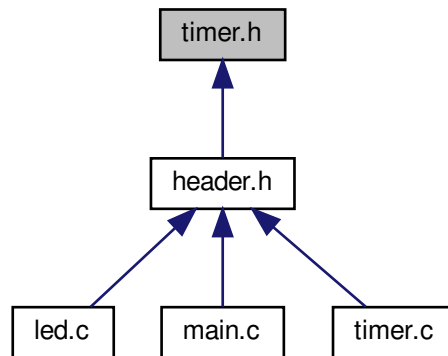
##### Returns

none.

## 7.1326 timer.h File Reference

Timer driver header file.

This graph shows which files directly or indirectly include this file:



### Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*

### 7.1326.1 Detailed Description

Timer driver header file.

### 7.1326.2 Function Documentation

#### 7.1326.2.1 TimerGet()

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

#### Returns

Current value of the millisecond timer.

### 7.1326.2.2 TimerInit()

```
void TimerInit (
 void)
```

Initializes the timer.

#### Returns

none.

Initializes the timer.

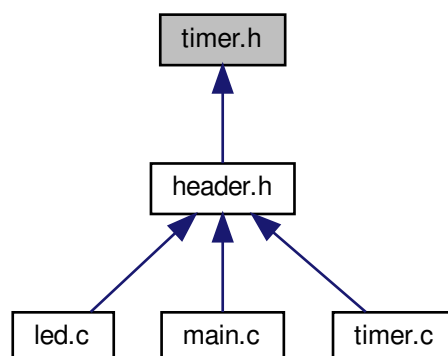
#### Returns

none.

## 7.1327 timer.h File Reference

Timer driver header file.

This graph shows which files directly or indirectly include this file:



## Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*

### 7.1327.1 Detailed Description

Timer driver header file.

## 7.1327.2 Function Documentation

### 7.1327.2.1 TimerGet()

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

#### Returns

Current value of the millisecond timer.

### 7.1327.2.2 TimerInit()

```
void TimerInit (
 void)
```

Initializes the timer.

#### Returns

none.

Initializes the timer.

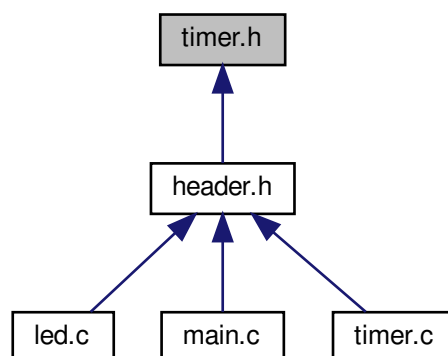
#### Returns

none.

## 7.1328 timer.h File Reference

Timer driver header file.

This graph shows which files directly or indirectly include this file:



## Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*

### 7.1328.1 Detailed Description

Timer driver header file.

### 7.1328.2 Function Documentation

#### 7.1328.2.1 [TimerGet\(\)](#)

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

#### Returns

Current value of the millisecond timer.

#### 7.1328.2.2 [TimerInit\(\)](#)

```
void TimerInit (
 void)
```

Initializes the timer.

#### Returns

none.

Initializes the timer.

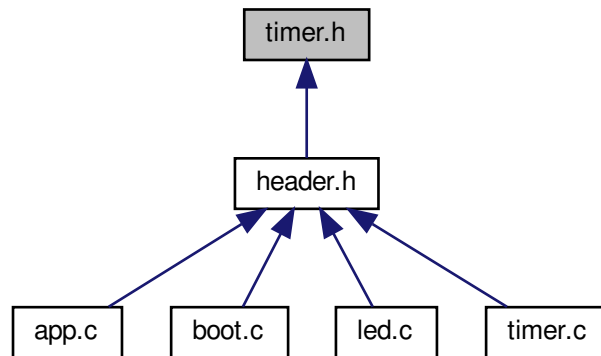
#### Returns

none.

## 7.1329 timer.h File Reference

Timer driver header file.

This graph shows which files directly or indirectly include this file:



### Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*

### 7.1329.1 Detailed Description

Timer driver header file.

### 7.1329.2 Function Documentation

#### 7.1329.2.1 TimerGet()

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

#### Returns

Current value of the millisecond timer.

### 7.1329.2.2 TimerInit()

```
void TimerInit (
 void)
```

Initializes the timer.

#### Returns

none.

Initializes the timer.

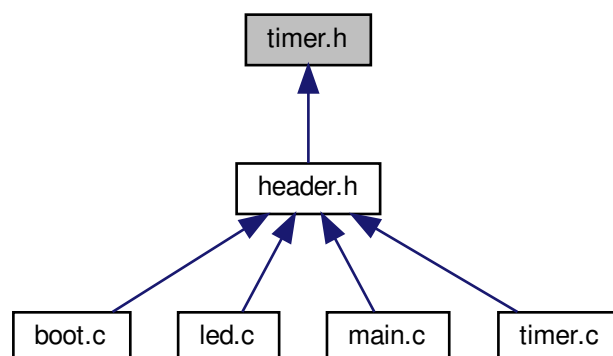
#### Returns

none.

## 7.1330 timer.h File Reference

Timer driver header file.

This graph shows which files directly or indirectly include this file:



## Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*

### 7.1330.1 Detailed Description

Timer driver header file.

## 7.1330.2 Function Documentation

### 7.1330.2.1 TimerGet()

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

#### Returns

Current value of the millisecond timer.

### 7.1330.2.2 TimerInit()

```
void TimerInit (
 void)
```

Initializes the timer.

#### Returns

none.

Initializes the timer.

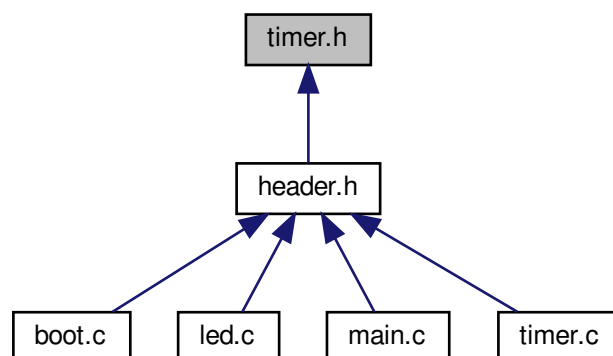
#### Returns

none.

## 7.1331 timer.h File Reference

Timer driver header file.

This graph shows which files directly or indirectly include this file:





## Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*

### 7.1331.1 Detailed Description

Timer driver header file.

### 7.1331.2 Function Documentation

#### 7.1331.2.1 TimerGet()

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

##### Returns

Current value of the millisecond timer.

#### 7.1331.2.2 TimerInit()

```
void TimerInit (
 void)
```

Initializes the timer.

##### Returns

none.

Initializes the timer.

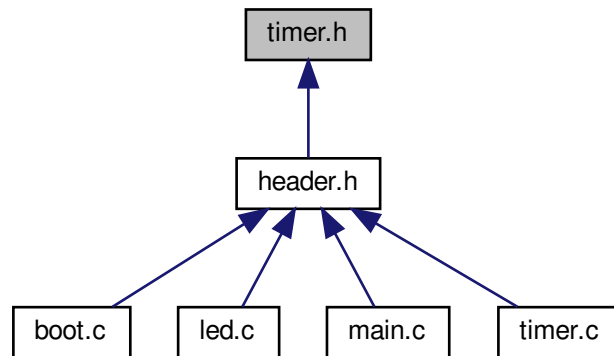
##### Returns

none.

## 7.1332 timer.h File Reference

Timer driver header file.

This graph shows which files directly or indirectly include this file:



### Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*

### 7.1332.1 Detailed Description

Timer driver header file.

### 7.1332.2 Function Documentation

#### 7.1332.2.1 TimerGet()

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

#### Returns

Current value of the millisecond timer.

### 7.1332.2.2 TimerInit()

```
void TimerInit (
 void)
```

Initializes the timer.

#### Returns

none.

Initializes the timer.

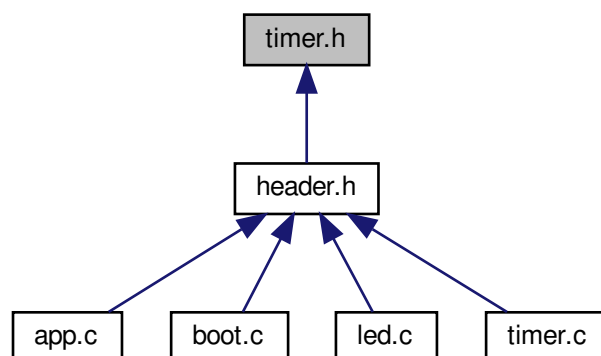
#### Returns

none.

## 7.1333 timer.h File Reference

Timer driver header file.

This graph shows which files directly or indirectly include this file:



## Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*

### 7.1333.1 Detailed Description

Timer driver header file.

## 7.1333.2 Function Documentation

### 7.1333.2.1 TimerGet()

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

#### Returns

Current value of the millisecond timer.

### 7.1333.2.2 TimerInit()

```
void TimerInit (
 void)
```

Initializes the timer.

#### Returns

none.

Initializes the timer.

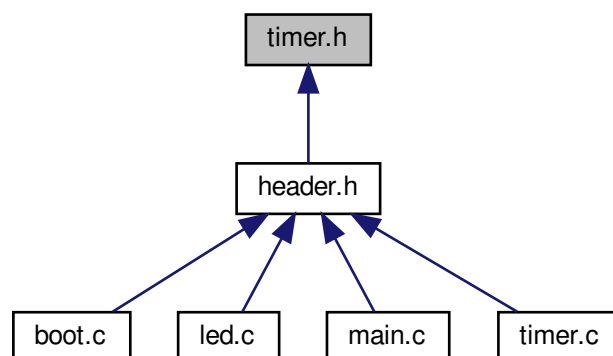
#### Returns

none.

## 7.1334 timer.h File Reference

Timer driver header file.

This graph shows which files directly or indirectly include this file:



## Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*

### 7.1334.1 Detailed Description

Timer driver header file.

### 7.1334.2 Function Documentation

#### 7.1334.2.1 TimerGet()

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

##### Returns

Current value of the millisecond timer.

#### 7.1334.2.2 TimerInit()

```
void TimerInit (
 void)
```

Initializes the timer.

##### Returns

none.

Initializes the timer.

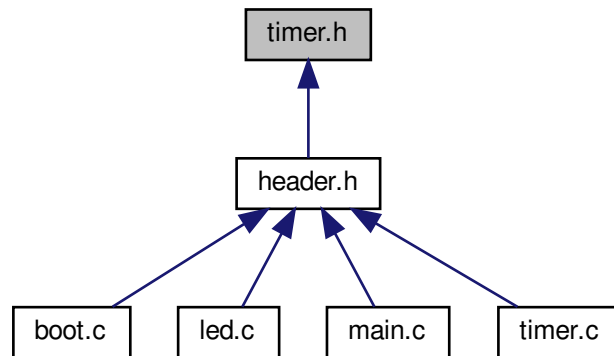
##### Returns

none.

## 7.1335 timer.h File Reference

Timer driver header file.

This graph shows which files directly or indirectly include this file:



### Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*

### 7.1335.1 Detailed Description

Timer driver header file.

### 7.1335.2 Function Documentation

#### 7.1335.2.1 TimerGet()

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

#### Returns

Current value of the millisecond timer.

### 7.1335.2.2 TimerInit()

```
void TimerInit (
 void)
```

Initializes the timer.

#### Returns

none.

Initializes the timer.

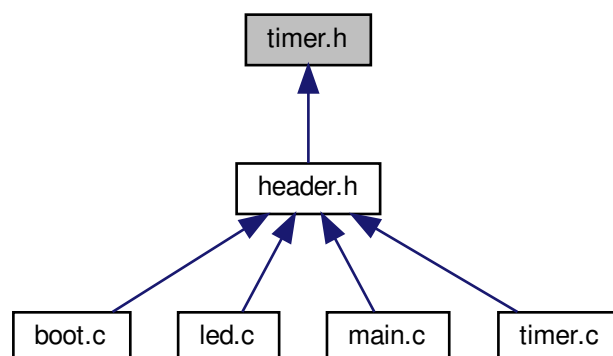
#### Returns

none.

## 7.1336 timer.h File Reference

Timer driver header file.

This graph shows which files directly or indirectly include this file:



## Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*

### 7.1336.1 Detailed Description

Timer driver header file.

## 7.1336.2 Function Documentation

### 7.1336.2.1 TimerGet()

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

#### Returns

Current value of the millisecond timer.

### 7.1336.2.2 TimerInit()

```
void TimerInit (
 void)
```

Initializes the timer.

#### Returns

none.

Initializes the timer.

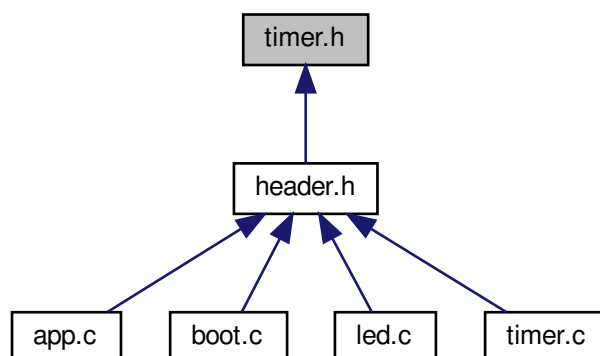
#### Returns

none.

## 7.1337 timer.h File Reference

Timer driver header file.

This graph shows which files directly or indirectly include this file:





## Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*

### 7.1337.1 Detailed Description

Timer driver header file.

### 7.1337.2 Function Documentation

#### 7.1337.2.1 [TimerGet\(\)](#)

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

##### Returns

Current value of the millisecond timer.

#### 7.1337.2.2 [TimerInit\(\)](#)

```
void TimerInit (
 void)
```

Initializes the timer.

##### Returns

none.

Initializes the timer.

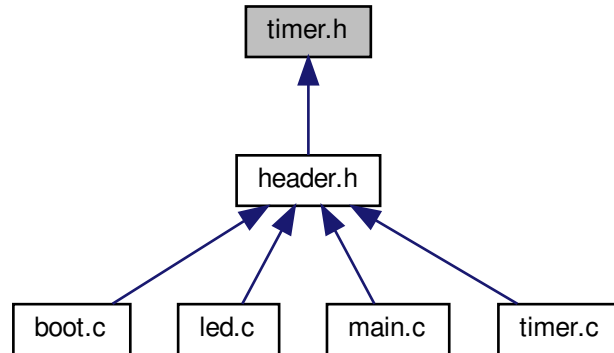
##### Returns

none.

## 7.1338 timer.h File Reference

Timer driver header file.

This graph shows which files directly or indirectly include this file:



### Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*

### 7.1338.1 Detailed Description

Timer driver header file.

### 7.1338.2 Function Documentation

#### 7.1338.2.1 TimerGet()

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

#### Returns

Current value of the millisecond timer.

### 7.1338.2.2 TimerInit()

```
void TimerInit (
 void)
```

Initializes the timer.

#### Returns

none.

Initializes the timer.

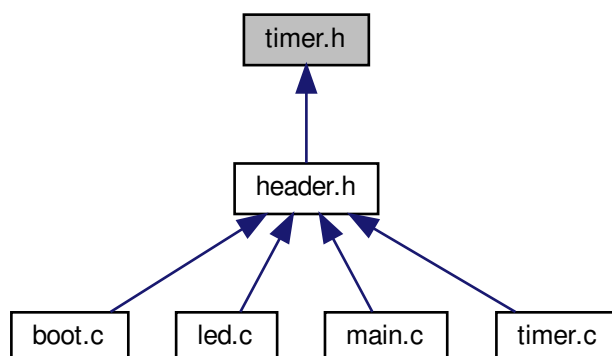
#### Returns

none.

## 7.1339 timer.h File Reference

Timer driver header file.

This graph shows which files directly or indirectly include this file:



## Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*

### 7.1339.1 Detailed Description

Timer driver header file.

## 7.1339.2 Function Documentation

### 7.1339.2.1 TimerGet()

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

#### Returns

Current value of the millisecond timer.

### 7.1339.2.2 TimerInit()

```
void TimerInit (
 void)
```

Initializes the timer.

#### Returns

none.

Initializes the timer.

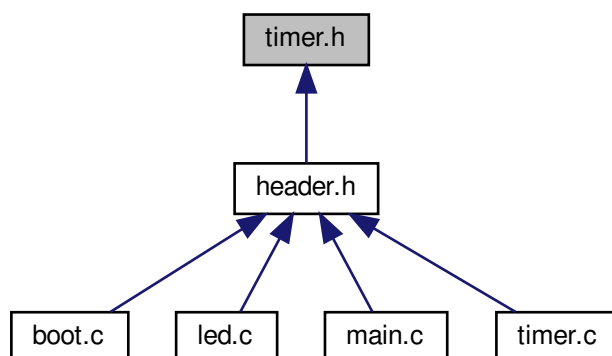
#### Returns

none.

## 7.1340 timer.h File Reference

Timer driver header file.

This graph shows which files directly or indirectly include this file:



## Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*

### 7.1340.1 Detailed Description

Timer driver header file.

### 7.1340.2 Function Documentation

#### 7.1340.2.1 [TimerGet\(\)](#)

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

##### Returns

Current value of the millisecond timer.

#### 7.1340.2.2 [TimerInit\(\)](#)

```
void TimerInit (
 void)
```

Initializes the timer.

##### Returns

none.

Initializes the timer.

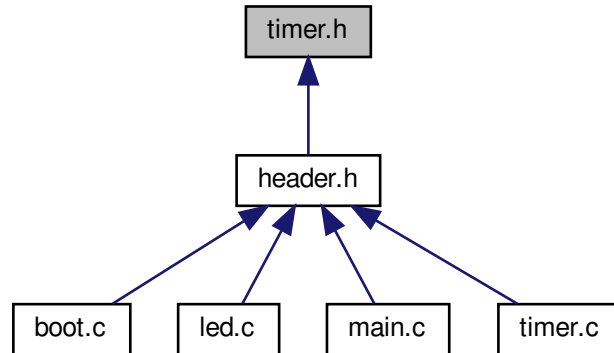
##### Returns

none.

## 7.1341 timer.h File Reference

Timer driver header file.

This graph shows which files directly or indirectly include this file:



### Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*

### 7.1341.1 Detailed Description

Timer driver header file.

### 7.1341.2 Function Documentation

#### 7.1341.2.1 TimerGet()

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

#### Returns

Current value of the millisecond timer.

### 7.1341.2.2 TimerInit()

```
void TimerInit (
 void)
```

Initializes the timer.

#### Returns

none.

Initializes the timer.

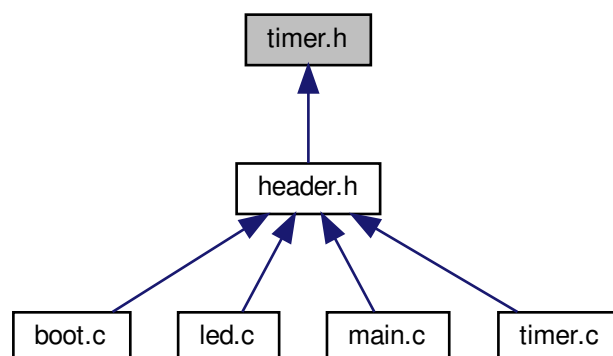
#### Returns

none.

## 7.1342 timer.h File Reference

Timer driver header file.

This graph shows which files directly or indirectly include this file:



### Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*

### 7.1342.1 Detailed Description

Timer driver header file.

## 7.1342.2 Function Documentation

### 7.1342.2.1 TimerGet()

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

#### Returns

Current value of the millisecond timer.

### 7.1342.2.2 TimerInit()

```
void TimerInit (
 void)
```

Initializes the timer.

#### Returns

none.

Initializes the timer.

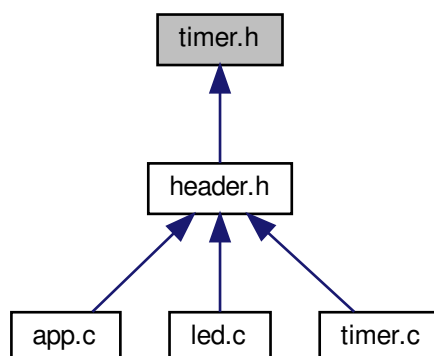
#### Returns

none.

## 7.1343 timer.h File Reference

Timer driver header file.

This graph shows which files directly or indirectly include this file:





## Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*

### 7.1343.1 Detailed Description

Timer driver header file.

### 7.1343.2 Function Documentation

#### 7.1343.2.1 [TimerGet\(\)](#)

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

##### Returns

Current value of the millisecond timer.

#### 7.1343.2.2 [TimerInit\(\)](#)

```
void TimerInit (
 void)
```

Initializes the timer.

##### Returns

none.

Initializes the timer.

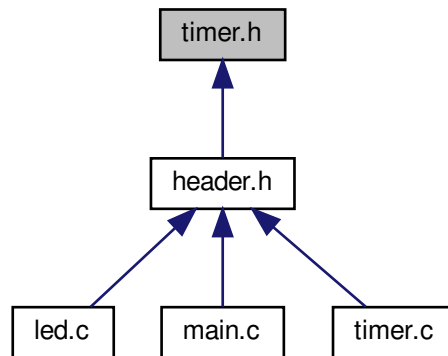
##### Returns

none.

## 7.1344 timer.h File Reference

Timer driver header file.

This graph shows which files directly or indirectly include this file:



### Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*

### 7.1344.1 Detailed Description

Timer driver header file.

### 7.1344.2 Function Documentation

#### 7.1344.2.1 TimerGet()

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

#### Returns

Current value of the millisecond timer.

### 7.1344.2.2 TimerInit()

```
void TimerInit (
 void)
```

Initializes the timer.

#### Returns

none.

Initializes the timer.

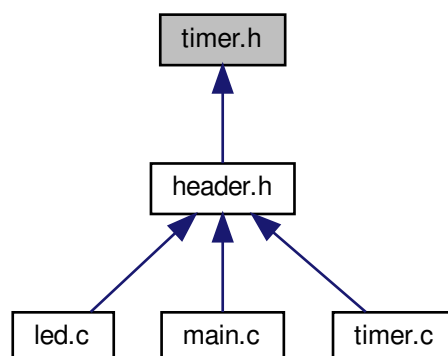
#### Returns

none.

## 7.1345 timer.h File Reference

Timer driver header file.

This graph shows which files directly or indirectly include this file:



## Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*

### 7.1345.1 Detailed Description

Timer driver header file.

## 7.1345.2 Function Documentation

### 7.1345.2.1 TimerGet()

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

#### Returns

Current value of the millisecond timer.

### 7.1345.2.2 TimerInit()

```
void TimerInit (
 void)
```

Initializes the timer.

#### Returns

none.

Initializes the timer.

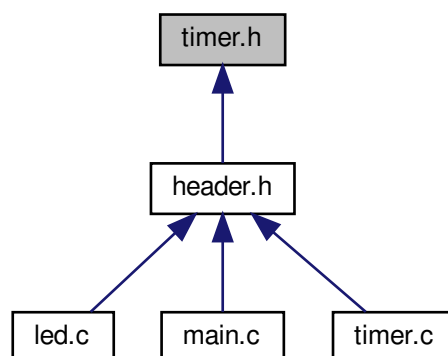
#### Returns

none.

## 7.1346 timer.h File Reference

Timer driver header file.

This graph shows which files directly or indirectly include this file:



## Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*

### 7.1346.1 Detailed Description

Timer driver header file.

### 7.1346.2 Function Documentation

#### 7.1346.2.1 [TimerGet\(\)](#)

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

##### Returns

Current value of the millisecond timer.

#### 7.1346.2.2 [TimerInit\(\)](#)

```
void TimerInit (
 void)
```

Initializes the timer.

##### Returns

none.

Initializes the timer.

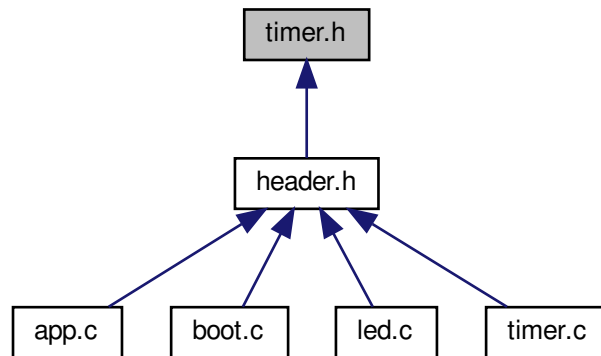
##### Returns

none.

## 7.1347 timer.h File Reference

Timer driver header file.

This graph shows which files directly or indirectly include this file:



### Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*

### 7.1347.1 Detailed Description

Timer driver header file.

### 7.1347.2 Function Documentation

#### 7.1347.2.1 TimerGet()

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

#### Returns

Current value of the millisecond timer.

### 7.1347.2.2 TimerInit()

```
void TimerInit (
 void)
```

Initializes the timer.

#### Returns

none.

Initializes the timer.

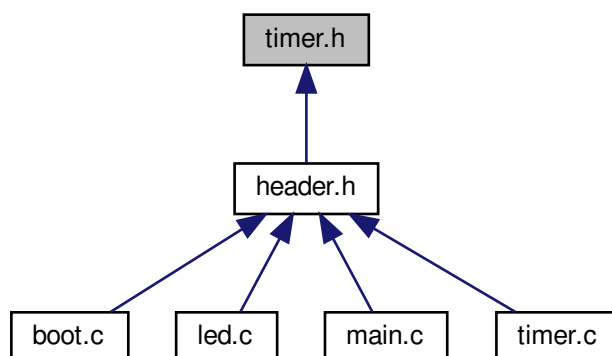
#### Returns

none.

## 7.1348 timer.h File Reference

Timer driver header file.

This graph shows which files directly or indirectly include this file:



### Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*

### 7.1348.1 Detailed Description

Timer driver header file.

## 7.1348.2 Function Documentation

### 7.1348.2.1 TimerGet()

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

#### Returns

Current value of the millisecond timer.

### 7.1348.2.2 TimerInit()

```
void TimerInit (
 void)
```

Initializes the timer.

#### Returns

none.

Initializes the timer.

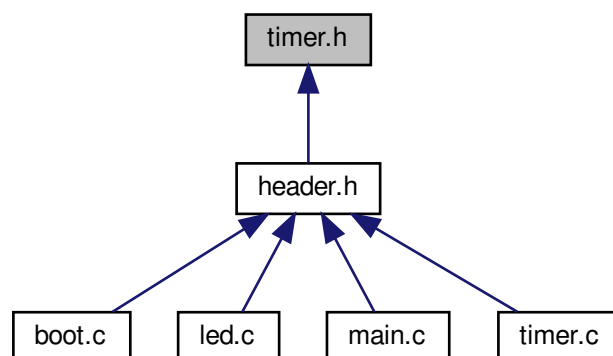
#### Returns

none.

## 7.1349 timer.h File Reference

Timer driver header file.

This graph shows which files directly or indirectly include this file:





## Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*

### 7.1349.1 Detailed Description

Timer driver header file.

### 7.1349.2 Function Documentation

#### 7.1349.2.1 [TimerGet\(\)](#)

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

##### Returns

Current value of the millisecond timer.

#### 7.1349.2.2 [TimerInit\(\)](#)

```
void TimerInit (
 void)
```

Initializes the timer.

##### Returns

none.

Initializes the timer.

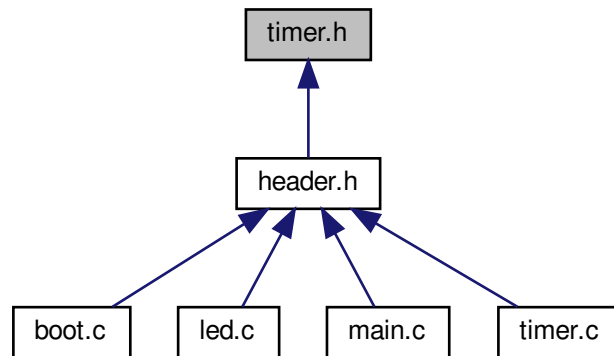
##### Returns

none.

## 7.1350 timer.h File Reference

Timer driver header file.

This graph shows which files directly or indirectly include this file:



### Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*

### 7.1350.1 Detailed Description

Timer driver header file.

### 7.1350.2 Function Documentation

#### 7.1350.2.1 TimerGet()

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

#### Returns

Current value of the millisecond timer.

### 7.1350.2.2 TimerInit()

```
void TimerInit (
 void)
```

Initializes the timer.

#### Returns

none.

Initializes the timer.

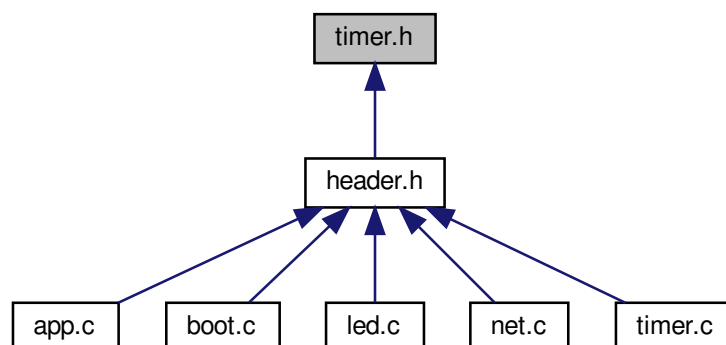
#### Returns

none.

## 7.1351 timer.h File Reference

Timer driver header file.

This graph shows which files directly or indirectly include this file:



### Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*

### 7.1351.1 Detailed Description

Timer driver header file.

## 7.1351.2 Function Documentation

### 7.1351.2.1 TimerGet()

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

#### Returns

Current value of the millisecond timer.

### 7.1351.2.2 TimerInit()

```
void TimerInit (
 void)
```

Initializes the timer.

#### Returns

none.

Initializes the timer.

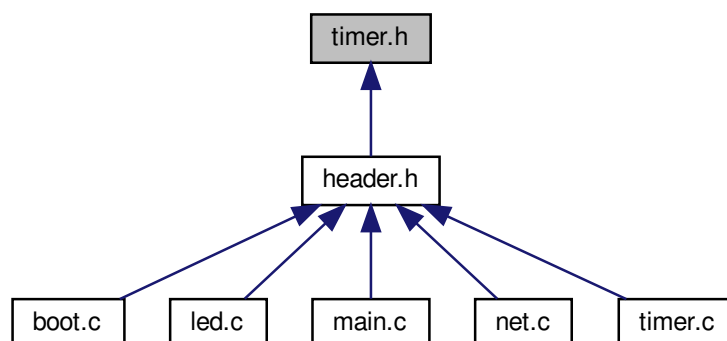
#### Returns

none.

## 7.1352 timer.h File Reference

Timer driver header file.

This graph shows which files directly or indirectly include this file:



## Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*

### 7.1352.1 Detailed Description

Timer driver header file.

### 7.1352.2 Function Documentation

#### 7.1352.2.1 [TimerGet\(\)](#)

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

##### Returns

Current value of the millisecond timer.

#### 7.1352.2.2 [TimerInit\(\)](#)

```
void TimerInit (
 void)
```

Initializes the timer.

##### Returns

none.

Initializes the timer.

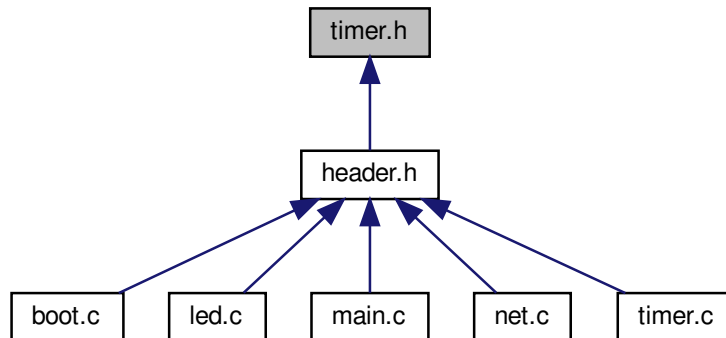
##### Returns

none.

## 7.1353 timer.h File Reference

Timer driver header file.

This graph shows which files directly or indirectly include this file:



### Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*

### 7.1353.1 Detailed Description

Timer driver header file.

### 7.1353.2 Function Documentation

#### 7.1353.2.1 TimerGet()

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

#### Returns

Current value of the millisecond timer.

### 7.1353.2.2 TimerInit()

```
void TimerInit (
 void)
```

Initializes the timer.

#### Returns

none.

Initializes the timer.

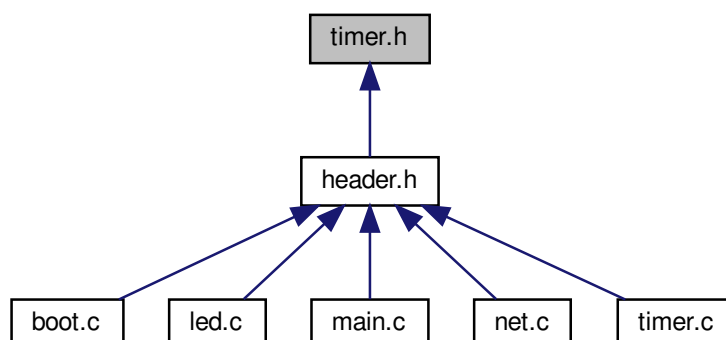
#### Returns

none.

## 7.1354 timer.h File Reference

Timer driver header file.

This graph shows which files directly or indirectly include this file:



## Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*

### 7.1354.1 Detailed Description

Timer driver header file.

## 7.1354.2 Function Documentation

### 7.1354.2.1 TimerGet()

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

#### Returns

Current value of the millisecond timer.

### 7.1354.2.2 TimerInit()

```
void TimerInit (
 void)
```

Initializes the timer.

#### Returns

none.

Initializes the timer.

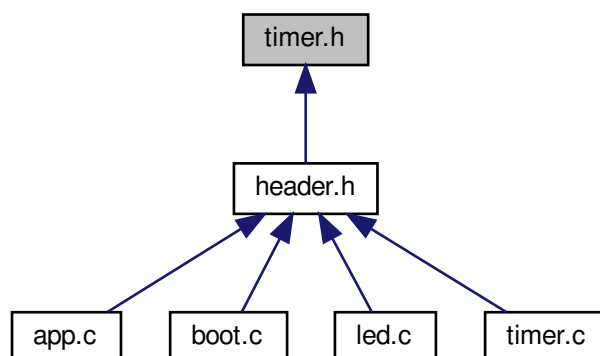
#### Returns

none.

## 7.1355 timer.h File Reference

Timer driver header file.

This graph shows which files directly or indirectly include this file:





## Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*

### 7.1355.1 Detailed Description

Timer driver header file.

### 7.1355.2 Function Documentation

#### 7.1355.2.1 [TimerGet\(\)](#)

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

##### Returns

Current value of the millisecond timer.

#### 7.1355.2.2 [TimerInit\(\)](#)

```
void TimerInit (
 void)
```

Initializes the timer.

##### Returns

none.

Initializes the timer.

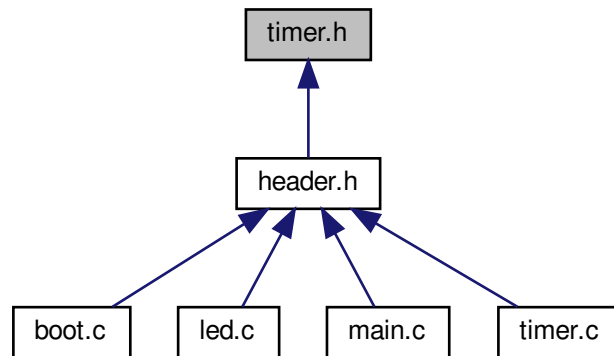
##### Returns

none.

## 7.1356 timer.h File Reference

Timer driver header file.

This graph shows which files directly or indirectly include this file:



### Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*

### 7.1356.1 Detailed Description

Timer driver header file.

### 7.1356.2 Function Documentation

#### 7.1356.2.1 TimerGet()

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

#### Returns

Current value of the millisecond timer.

### 7.1356.2.2 TimerInit()

```
void TimerInit (
 void)
```

Initializes the timer.

#### Returns

none.

Initializes the timer.

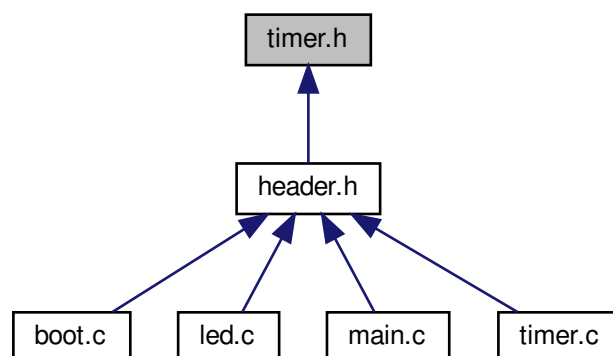
#### Returns

none.

## 7.1357 timer.h File Reference

Timer driver header file.

This graph shows which files directly or indirectly include this file:



## Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*

### 7.1357.1 Detailed Description

Timer driver header file.

## 7.1357.2 Function Documentation

### 7.1357.2.1 TimerGet()

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

#### Returns

Current value of the millisecond timer.

### 7.1357.2.2 TimerInit()

```
void TimerInit (
 void)
```

Initializes the timer.

#### Returns

none.

Initializes the timer.

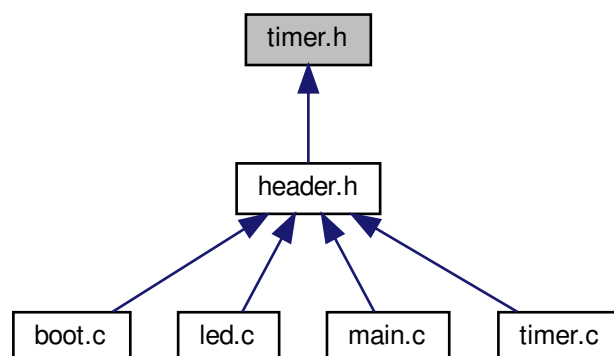
#### Returns

none.

## 7.1358 timer.h File Reference

Timer driver header file.

This graph shows which files directly or indirectly include this file:



## Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*

### 7.1358.1 Detailed Description

Timer driver header file.

### 7.1358.2 Function Documentation

#### 7.1358.2.1 TimerGet()

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

##### Returns

Current value of the millisecond timer.

#### 7.1358.2.2 TimerInit()

```
void TimerInit (
 void)
```

Initializes the timer.

##### Returns

none.

Initializes the timer.

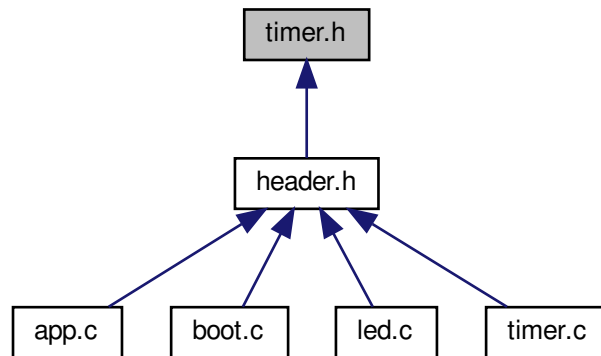
##### Returns

none.

## 7.1359 timer.h File Reference

Timer driver header file.

This graph shows which files directly or indirectly include this file:



### Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*

### 7.1359.1 Detailed Description

Timer driver header file.

### 7.1359.2 Function Documentation

#### 7.1359.2.1 TimerGet()

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

#### Returns

Current value of the millisecond timer.

### 7.1359.2.2 TimerInit()

```
void TimerInit (
 void)
```

Initializes the timer.

#### Returns

none.

Initializes the timer.

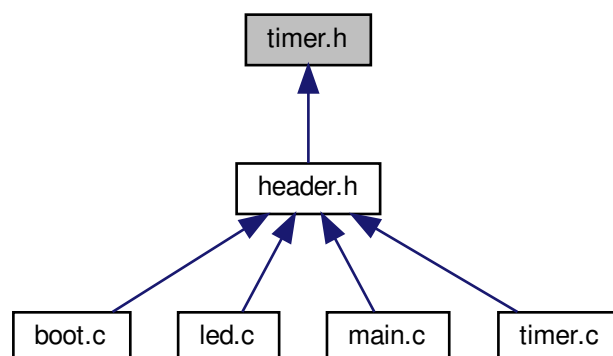
#### Returns

none.

## 7.1360 timer.h File Reference

Timer driver header file.

This graph shows which files directly or indirectly include this file:



## Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*

### 7.1360.1 Detailed Description

Timer driver header file.

## 7.1360.2 Function Documentation

### 7.1360.2.1 TimerGet()

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

#### Returns

Current value of the millisecond timer.

### 7.1360.2.2 TimerInit()

```
void TimerInit (
 void)
```

Initializes the timer.

#### Returns

none.

Initializes the timer.

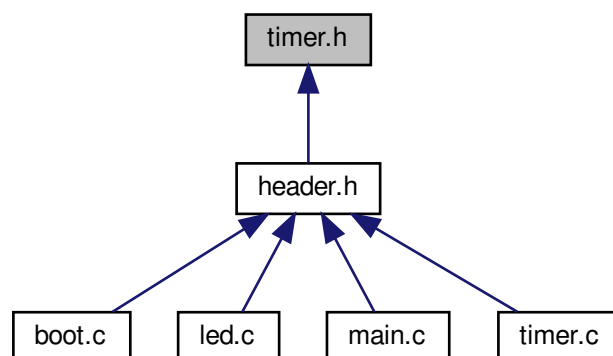
#### Returns

none.

## 7.1361 timer.h File Reference

Timer driver header file.

This graph shows which files directly or indirectly include this file:





## Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*

### 7.1361.1 Detailed Description

Timer driver header file.

### 7.1361.2 Function Documentation

#### 7.1361.2.1 [TimerGet\(\)](#)

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

##### Returns

Current value of the millisecond timer.

#### 7.1361.2.2 [TimerInit\(\)](#)

```
void TimerInit (
 void)
```

Initializes the timer.

##### Returns

none.

Initializes the timer.

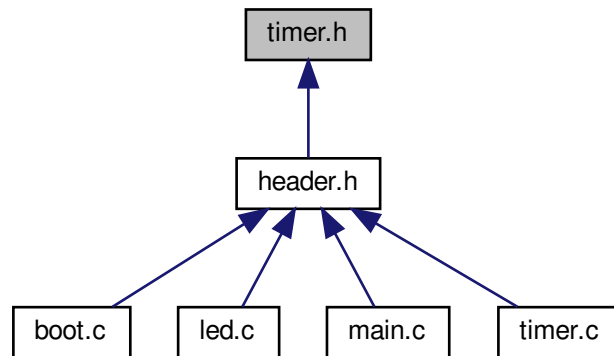
##### Returns

none.

## 7.1362 timer.h File Reference

Timer driver header file.

This graph shows which files directly or indirectly include this file:



### Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*

### 7.1362.1 Detailed Description

Timer driver header file.

### 7.1362.2 Function Documentation

#### 7.1362.2.1 TimerGet()

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

#### Returns

Current value of the millisecond timer.

### 7.1362.2.2 TimerInit()

```
void TimerInit (
 void)
```

Initializes the timer.

#### Returns

none.

Initializes the timer.

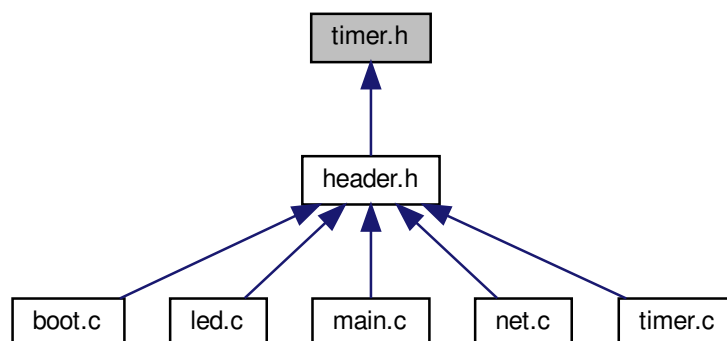
#### Returns

none.

## 7.1363 timer.h File Reference

Timer driver header file.

This graph shows which files directly or indirectly include this file:



## Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- void [TimerDeinit](#) (void)  
*Stops and disables the timer.*
- void [TimerSet](#) (unsigned long timer\_value)  
*Sets the initial counter value of the millisecond timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*

### 7.1363.1 Detailed Description

Timer driver header file.

### 7.1363.2 Function Documentation

#### 7.1363.2.1 TimerDeinit()

```
void TimerDeinit (
 void)
```

Stops and disables the timer.

##### Returns

none.

#### 7.1363.2.2 TimerGet()

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

##### Returns

Current value of the millisecond timer.

#### 7.1363.2.3 TimerInit()

```
void TimerInit (
 void)
```

Initializes the timer.

##### Returns

none.

Initializes the timer.

##### Returns

none.

#### 7.1363.2.4 TimerSet()

```
void TimerSet (
 unsigned long timer_value)
```

Sets the initial counter value of the millisecond timer.

## Parameters

|                          |                                            |
|--------------------------|--------------------------------------------|
| <code>timer_value</code> | initialize value of the millisecond timer. |
|--------------------------|--------------------------------------------|

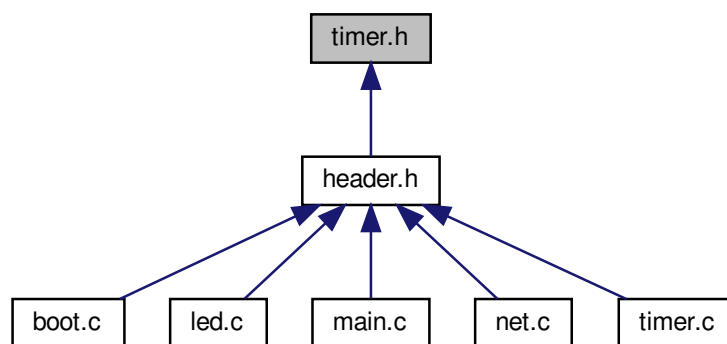
## Returns

none.

## 7.1364 timer.h File Reference

Timer driver header file.

This graph shows which files directly or indirectly include this file:



## Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- void [TimerDeinit](#) (void)  
*Stops and disables the timer.*
- void [TimerSet](#) (unsigned long timer\_value)  
*Sets the initial counter value of the millisecond timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*

### 7.1364.1 Detailed Description

Timer driver header file.

## 7.1364.2 Function Documentation

### 7.1364.2.1 TimerDeinit()

```
void TimerDeinit (
 void)
```

Stops and disables the timer.

#### Returns

none.

### 7.1364.2.2 TimerGet()

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

#### Returns

Current value of the millisecond timer.

### 7.1364.2.3 TimerInit()

```
void TimerInit (
 void)
```

Initializes the timer.

#### Returns

none.

Initializes the timer.

#### Returns

none.

### 7.1364.2.4 TimerSet()

```
void TimerSet (
 unsigned long timer_value)
```

Sets the initial counter value of the millisecond timer.

## Parameters

|                          |                                            |
|--------------------------|--------------------------------------------|
| <code>timer_value</code> | initialize value of the millisecond timer. |
|--------------------------|--------------------------------------------|

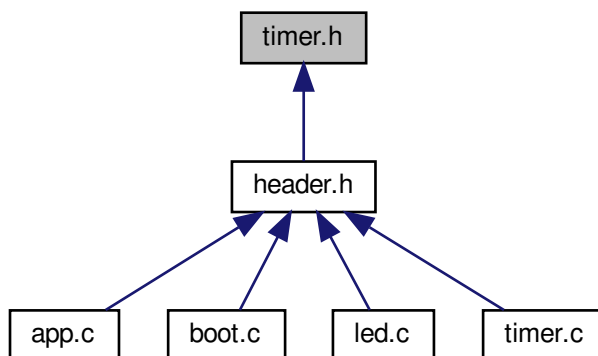
## Returns

none.

## 7.1365 timer.h File Reference

Timer driver header file.

This graph shows which files directly or indirectly include this file:



## Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*

### 7.1365.1 Detailed Description

Timer driver header file.

### 7.1365.2 Function Documentation

### 7.1365.2.1 TimerGet()

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

#### Returns

Current value of the millisecond timer.

### 7.1365.2.2 TimerInit()

```
void TimerInit (
 void)
```

Initializes the timer.

#### Returns

none.

Initializes the timer.

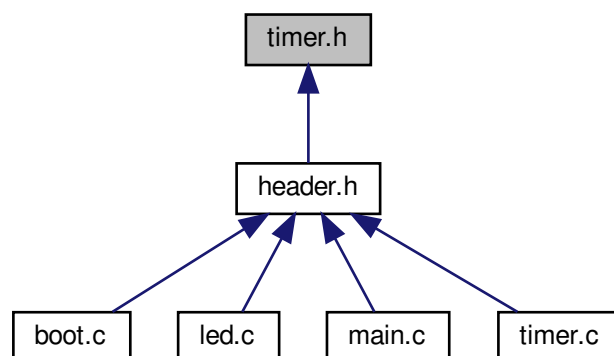
#### Returns

none.

## 7.1366 timer.h File Reference

Timer driver header file.

This graph shows which files directly or indirectly include this file:





## Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*

### 7.1366.1 Detailed Description

Timer driver header file.

### 7.1366.2 Function Documentation

#### 7.1366.2.1 [TimerGet\(\)](#)

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

##### Returns

Current value of the millisecond timer.

#### 7.1366.2.2 [TimerInit\(\)](#)

```
void TimerInit (
 void)
```

Initializes the timer.

##### Returns

none.

Initializes the timer.

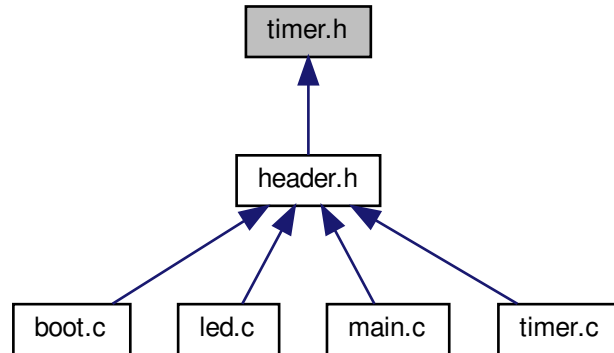
##### Returns

none.

## 7.1367 timer.h File Reference

Timer driver header file.

This graph shows which files directly or indirectly include this file:



### Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*

### 7.1367.1 Detailed Description

Timer driver header file.

### 7.1367.2 Function Documentation

#### 7.1367.2.1 TimerGet()

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

#### Returns

Current value of the millisecond timer.

### 7.1367.2.2 TimerInit()

```
void TimerInit (
 void)
```

Initializes the timer.

#### Returns

none.

Initializes the timer.

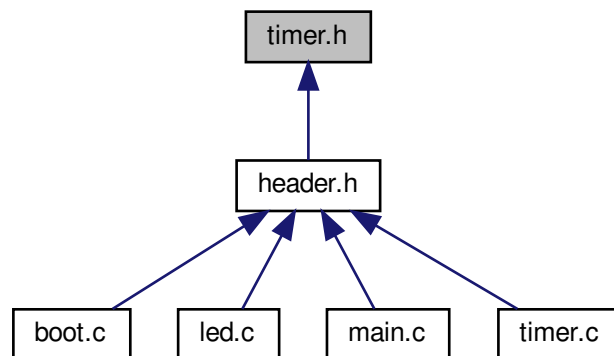
#### Returns

none.

## 7.1368 timer.h File Reference

Timer driver header file.

This graph shows which files directly or indirectly include this file:



## Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*

## 7.1368.1 Detailed Description

Timer driver header file.

## 7.1368.2 Function Documentation

### 7.1368.2.1 TimerGet()

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

#### Returns

Current value of the millisecond timer.

### 7.1368.2.2 TimerInit()

```
void TimerInit (
 void)
```

Initializes the timer.

#### Returns

none.

Initializes the timer.

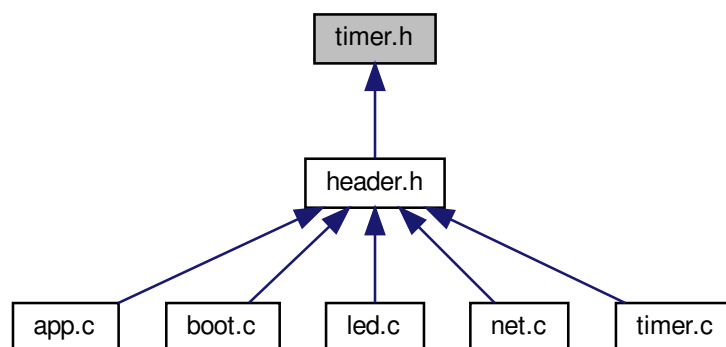
#### Returns

none.

## 7.1369 timer.h File Reference

Timer driver header file.

This graph shows which files directly or indirectly include this file:



## Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*

### 7.1369.1 Detailed Description

Timer driver header file.

### 7.1369.2 Function Documentation

#### 7.1369.2.1 [TimerGet\(\)](#)

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

##### Returns

Current value of the millisecond timer.

#### 7.1369.2.2 [TimerInit\(\)](#)

```
void TimerInit (
 void)
```

Initializes the timer.

##### Returns

none.

Initializes the timer.

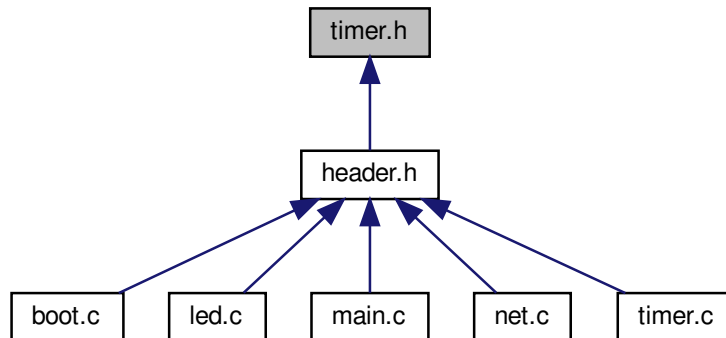
##### Returns

none.

## 7.1370 timer.h File Reference

Timer driver header file.

This graph shows which files directly or indirectly include this file:



### Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*

### 7.1370.1 Detailed Description

Timer driver header file.

### 7.1370.2 Function Documentation

#### 7.1370.2.1 TimerGet()

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

#### Returns

Current value of the millisecond timer.

### 7.1370.2.2 TimerInit()

```
void TimerInit (
 void)
```

Initializes the timer.

#### Returns

none.

Initializes the timer.

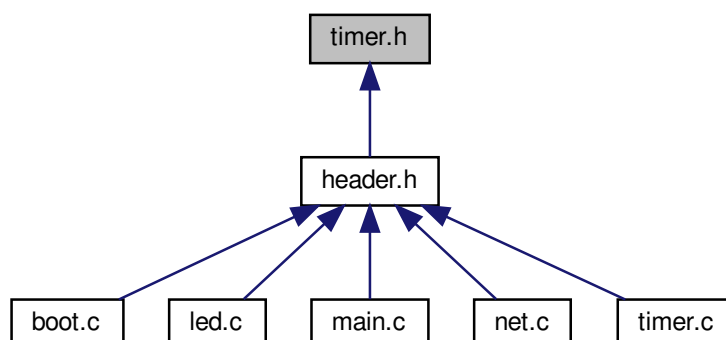
#### Returns

none.

## 7.1371 timer.h File Reference

Timer driver header file.

This graph shows which files directly or indirectly include this file:



## Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*

### 7.1371.1 Detailed Description

Timer driver header file.

## 7.1371.2 Function Documentation

### 7.1371.2.1 TimerGet()

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

#### Returns

Current value of the millisecond timer.

### 7.1371.2.2 TimerInit()

```
void TimerInit (
 void)
```

Initializes the timer.

#### Returns

none.

Initializes the timer.

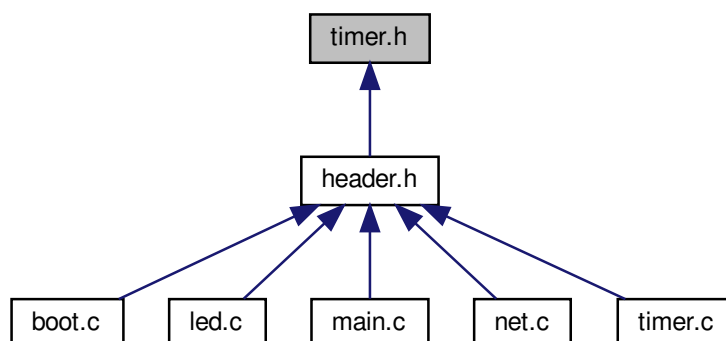
#### Returns

none.

## 7.1372 timer.h File Reference

Timer driver header file.

This graph shows which files directly or indirectly include this file:





## Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*

### 7.1372.1 Detailed Description

Timer driver header file.

### 7.1372.2 Function Documentation

#### 7.1372.2.1 [TimerGet\(\)](#)

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

##### Returns

Current value of the millisecond timer.

#### 7.1372.2.2 [TimerInit\(\)](#)

```
void TimerInit (
 void)
```

Initializes the timer.

##### Returns

none.

Initializes the timer.

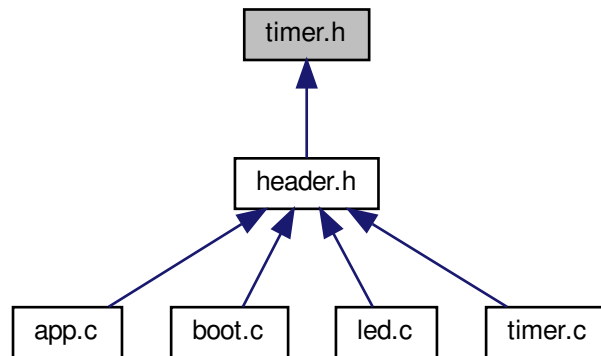
##### Returns

none.

## 7.1373 timer.h File Reference

Timer driver header file.

This graph shows which files directly or indirectly include this file:



### Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*

### 7.1373.1 Detailed Description

Timer driver header file.

### 7.1373.2 Function Documentation

#### 7.1373.2.1 TimerGet()

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

#### Returns

Current value of the millisecond timer.

### 7.1373.2.2 TimerInit()

```
void TimerInit (
 void)
```

Initializes the timer.

#### Returns

none.

Initializes the timer.

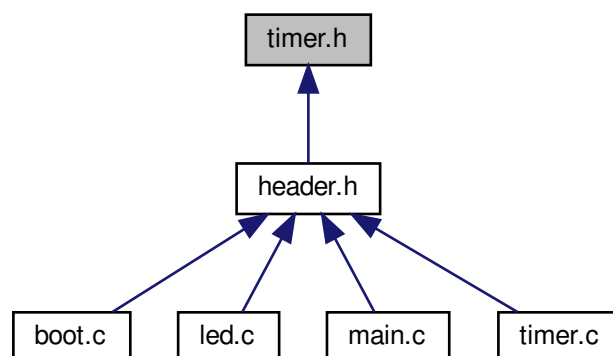
#### Returns

none.

## 7.1374 timer.h File Reference

Timer driver header file.

This graph shows which files directly or indirectly include this file:



## Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*

### 7.1374.1 Detailed Description

Timer driver header file.

## 7.1374.2 Function Documentation

### 7.1374.2.1 TimerGet()

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

#### Returns

Current value of the millisecond timer.

### 7.1374.2.2 TimerInit()

```
void TimerInit (
 void)
```

Initializes the timer.

#### Returns

none.

Initializes the timer.

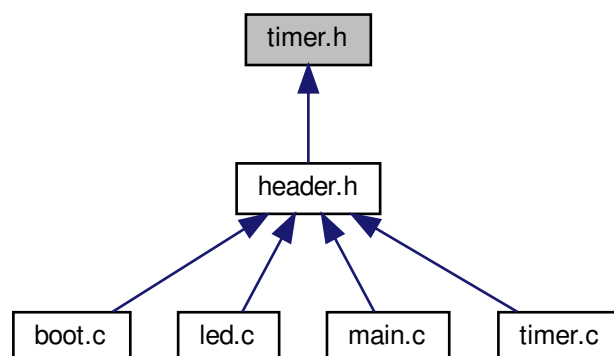
#### Returns

none.

## 7.1375 timer.h File Reference

Timer driver header file.

This graph shows which files directly or indirectly include this file:



## Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*

### 7.1375.1 Detailed Description

Timer driver header file.

### 7.1375.2 Function Documentation

#### 7.1375.2.1 [TimerGet\(\)](#)

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

##### Returns

Current value of the millisecond timer.

#### 7.1375.2.2 [TimerInit\(\)](#)

```
void TimerInit (
 void)
```

Initializes the timer.

##### Returns

none.

Initializes the timer.

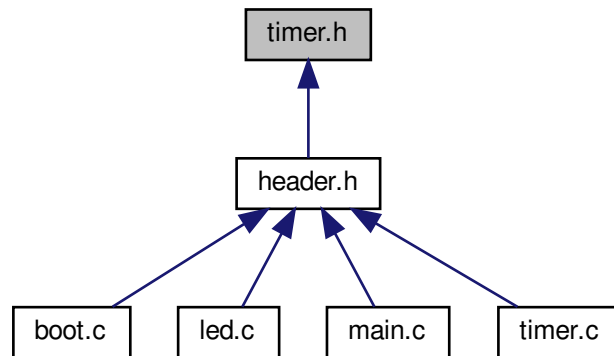
##### Returns

none.

## 7.1376 timer.h File Reference

Timer driver header file.

This graph shows which files directly or indirectly include this file:



### Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*

### 7.1376.1 Detailed Description

Timer driver header file.

### 7.1376.2 Function Documentation

#### 7.1376.2.1 TimerGet()

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

#### Returns

Current value of the millisecond timer.

### 7.1376.2.2 TimerInit()

```
void TimerInit (
 void)
```

Initializes the timer.

#### Returns

none.

Initializes the timer.

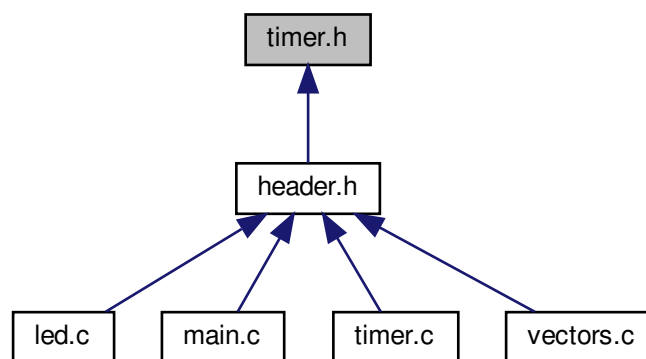
#### Returns

none.

## 7.1377 timer.h File Reference

Timer driver header file.

This graph shows which files directly or indirectly include this file:



## Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- void [TimerDeinit](#) (void)  
*Stops and disables the timer.*
- void [TimerSet](#) (unsigned long timer\_value)  
*Sets the initial counter value of the millisecond timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*
- \_\_interrupt void [TimerISRHandler](#) (void)  
*Interrupt service routine of the timer.*

### 7.1377.1 Detailed Description

Timer driver header file.

### 7.1377.2 Function Documentation

#### 7.1377.2.1 TimerDeinit()

```
void TimerDeinit (
 void)
```

Stops and disables the timer.

##### Returns

none.

#### 7.1377.2.2 TimerGet()

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

##### Returns

Current value of the millisecond timer.

#### 7.1377.2.3 TimerInit()

```
void TimerInit (
 void)
```

Initializes the timer.

##### Returns

none.

Initializes the timer.

##### Returns

none.



#### 7.1377.2.4 TimerISRHandler()

```
__interrupt void TimerISRHandler (
 void)
```

Interrupt service routine of the timer.

##### Returns

none.

#### 7.1377.2.5 TimerSet()

```
void TimerSet (
 unsigned long timer_value)
```

Sets the initial counter value of the millisecond timer.

##### Parameters

|                    |                                            |
|--------------------|--------------------------------------------|
| <i>timer_value</i> | initialize value of the millisecond timer. |
|--------------------|--------------------------------------------|

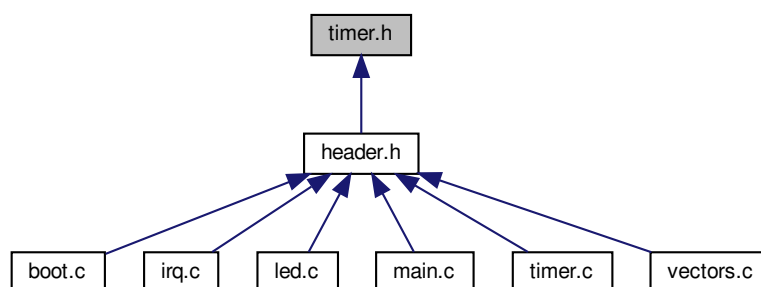
##### Returns

none.

## 7.1378 timer.h File Reference

Timer driver header file.

This graph shows which files directly or indirectly include this file:



## Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- void [TimerDeinit](#) (void)  
*Stops and disables the timer.*
- void [TimerSet](#) (unsigned long timer\_value)  
*Sets the initial counter value of the millisecond timer.*
- unsigned long [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*
- void [TimerISRHandler](#) (void)  
*Interrupt service routine of the timer.*

### 7.1378.1 Detailed Description

Timer driver header file.

### 7.1378.2 Function Documentation

#### 7.1378.2.1 [TimerDeinit\(\)](#)

```
void TimerDeinit (
 void)
```

Stops and disables the timer.

##### Returns

none.

Referenced by [TimerInit\(\)](#).

#### 7.1378.2.2 [TimerGet\(\)](#)

```
unsigned long TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

##### Returns

Current value of the millisecond timer.

### 7.1378.2.3 TimerInit()

```
void TimerInit (
 void)
```

Initializes the timer.

#### Returns

none.

Initializes the timer.

#### Returns

none.

### 7.1378.2.4 TimerISRHandler()

```
void TimerISRHandler (
 void)
```

Interrupt service routine of the timer.

#### Returns

none.

Referenced by `__attribute__()`.

### 7.1378.2.5 TimerSet()

```
void TimerSet (
 unsigned long timer_value)
```

Sets the initial counter value of the millisecond timer.

#### Parameters

|                    |                                            |
|--------------------|--------------------------------------------|
| <i>timer_value</i> | initialize value of the millisecond timer. |
|--------------------|--------------------------------------------|

#### Returns

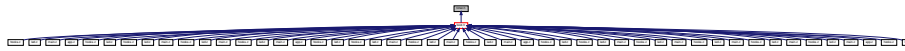
none.

Referenced by `TimerInit()`.

## 7.1379 timer.h File Reference

Bootloader timer driver header file.

This graph shows which files directly or indirectly include this file:



### Functions

- void [TimerInit](#) (void)  
*Initializes the timer.*
- void [TimerUpdate](#) (void)  
*Updates the millisecond timer.*
- [blt\\_int32u](#) [TimerGet](#) (void)  
*Obtains the counter value of the millisecond timer.*
- void [TimerReset](#) (void)  
*Reset the timer by placing the timer back into it's default reset configuration.*

### 7.1379.1 Detailed Description

Bootloader timer driver header file.

### 7.1379.2 Function Documentation

#### 7.1379.2.1 TimerGet()

```
blt_int32u TimerGet (
 void)
```

Obtains the counter value of the millisecond timer.

#### Returns

Current value of the millisecond timer.

Referenced by [BackDoorCheck\(\)](#), [BootComCanInit\(\)](#), [BootComRs232CheckActivationRequest\(\)](#), [CanDisabledModeEnter\(\)](#), [CanDisabledModeExit\(\)](#), [CanFreezeModeEnter\(\)](#), [CanFreezeModeExit\(\)](#), [CanInit\(\)](#), [CanTransmitPacket\(\)](#), [FileFirmwareUpdateCompletedHook\(\)](#), [FileFirmwareUpdateLogHook\(\)](#), [FlashEraseSectors\(\)](#), [FlashWriteBlock\(\)](#), [HAL\\_GetTick\(\)](#), [LedBlinkTask\(\)](#), [LedToggle\(\)](#), [NetInit\(\)](#), [NetServerTask\(\)](#), and [NetTask\(\)](#).

### 7.1379.2.2 TimerInit()

```
void TimerInit (
 void)
```

Initializes the timer.

#### Returns

none.

Initializes the timer.

#### Returns

none.

Referenced by ApplInit(), BootInit(), and Init().

### 7.1379.2.3 TimerReset()

```
void TimerReset (
 void)
```

Reset the timer by placing the timer back into it's default reset configuration.

#### Returns

none.

Referenced by CpuStartUserProgram(), and TimerInit().

### 7.1379.2.4 TimerUpdate()

```
void TimerUpdate (
 void)
```

Updates the millisecond timer.

#### Returns

none.

Referenced by BootTask(), and TimerGet().

## 7.1380 types.h File Reference

Bootloader types header file.

### Macros

- `#define BLT_TRUE (1)`  
*Boolean true value.*
- `#define BLT_FALSE (0)`  
*Boolean false value.*
- `#define BLT_NULL ((void *)0)`  
*NULL pointer value.*

### Typedefs

- `typedef unsigned char blt_bool`
- `typedef char blt_char`
- `typedef unsigned long blt_addr`
- `typedef unsigned char blt_int8u`
- `typedef signed char blt_int8s`
- `typedef unsigned short blt_int16u`
- `typedef signed short blt_int16s`
- `typedef unsigned int blt_int32u`
- `typedef signed int blt_int32s`

### 7.1380.1 Detailed Description

Bootloader types header file.

### 7.1380.2 Typedef Documentation

#### 7.1380.2.1 blt\_addr

```
typedef unsigned long blt_addr
```

memory address type

#### 7.1380.2.2 blt\_bool

```
typedef unsigned char blt_bool
```

boolean type

### 7.1380.2.3 blt\_char

```
typedef char blt_char
```

character type

### 7.1380.2.4 blt\_int16s

```
typedef signed short blt_int16s
```

16-bit signed integer

### 7.1380.2.5 blt\_int16u

```
typedef unsigned short blt_int16u
```

16-bit unsigned integer

### 7.1380.2.6 blt\_int32s

```
typedef signed int blt_int32s
```

32-bit signed integer

### 7.1380.2.7 blt\_int32u

```
typedef unsigned int blt_int32u
```

32-bit unsigned integer

### 7.1380.2.8 blt\_int8s

```
typedef signed char blt_int8s
```

8-bit signed integer

### 7.1380.2.9 blt\_int8u

```
typedef unsigned char blt_int8u
```

8-bit unsigned integer

## 7.1381 types.h File Reference

Bootloader types header file.

## Macros

- `#define BLT_TRUE (1)`  
*Boolean true value.*
- `#define BLT_FALSE (0)`  
*Boolean false value.*
- `#define BLT_NULL ((void *)0)`  
*NULL pointer value.*

## Typedefs

- `typedef unsigned char blt_bool`
- `typedef char blt_char`
- `typedef unsigned long blt_addr`
- `typedef unsigned char blt_int8u`
- `typedef signed char blt_int8s`
- `typedef unsigned short blt_int16u`
- `typedef signed short blt_int16s`
- `typedef unsigned int blt_int32u`
- `typedef signed int blt_int32s`

### 7.1381.1 Detailed Description

Bootloader types header file.

### 7.1381.2 Typedef Documentation

#### 7.1381.2.1 blt\_addr

```
typedef unsigned long blt_addr
```

memory address type

#### 7.1381.2.2 blt\_bool

```
typedef unsigned char blt_bool
```

boolean type

#### 7.1381.2.3 blt\_char

```
typedef char blt_char
```

character type



#### 7.1381.2.4 blt\_int16s

```
typedef signed short blt_int16s
```

16-bit signed integer

#### 7.1381.2.5 blt\_int16u

```
typedef unsigned short blt_int16u
```

16-bit unsigned integer

#### 7.1381.2.6 blt\_int32s

```
typedef signed int blt_int32s
```

32-bit signed integer

#### 7.1381.2.7 blt\_int32u

```
typedef unsigned int blt_int32u
```

32-bit unsigned integer

#### 7.1381.2.8 blt\_int8s

```
typedef signed char blt_int8s
```

8-bit signed integer

#### 7.1381.2.9 blt\_int8u

```
typedef unsigned char blt_int8u
```

8-bit unsigned integer

## 7.1382 types.h File Reference

Bootloader types header file.

## Macros

- `#define BLT_TRUE (1)`  
*Boolean true value.*
- `#define BLT_FALSE (0)`  
*Boolean false value.*
- `#define BLT_NULL ((void *)0)`  
*NULL pointer value.*

## Typedefs

- `typedef unsigned char blt_bool`
- `typedef char blt_char`
- `typedef unsigned long blt_addr`
- `typedef unsigned char blt_int8u`
- `typedef signed char blt_int8s`
- `typedef unsigned short blt_int16u`
- `typedef signed short blt_int16s`
- `typedef unsigned int blt_int32u`
- `typedef signed int blt_int32s`

### 7.1382.1 Detailed Description

Bootloader types header file.

### 7.1382.2 Typedef Documentation

#### 7.1382.2.1 blt\_addr

```
typedef unsigned long blt_addr
```

memory address type

#### 7.1382.2.2 blt\_bool

```
typedef unsigned char blt_bool
```

boolean type

#### 7.1382.2.3 blt\_char

```
typedef char blt_char
```

character type

#### 7.1382.2.4 blt\_int16s

```
typedef signed short blt_int16s
```

16-bit signed integer

#### 7.1382.2.5 blt\_int16u

```
typedef unsigned short blt_int16u
```

16-bit unsigned integer

#### 7.1382.2.6 blt\_int32s

```
typedef signed int blt_int32s
```

32-bit signed integer

#### 7.1382.2.7 blt\_int32u

```
typedef unsigned int blt_int32u
```

32-bit unsigned integer

#### 7.1382.2.8 blt\_int8s

```
typedef signed char blt_int8s
```

8-bit signed integer

#### 7.1382.2.9 blt\_int8u

```
typedef unsigned char blt_int8u
```

8-bit unsigned integer

## 7.1383 types.h File Reference

Bootloader types header file.

## Macros

- `#define BLT_TRUE (1)`  
*Boolean true value.*
- `#define BLT_FALSE (0)`  
*Boolean false value.*
- `#define BLT_NULL ((void *)0)`  
*NULL pointer value.*

## Typedefs

- `typedef unsigned char blt_bool`
- `typedef char blt_char`
- `typedef unsigned long blt_addr`
- `typedef unsigned char blt_int8u`
- `typedef signed char blt_int8s`
- `typedef unsigned short blt_int16u`
- `typedef signed short blt_int16s`
- `typedef unsigned int blt_int32u`
- `typedef signed int blt_int32s`
- `typedef unsigned long long blt_int64u`
- `typedef signed long long blt_int64s`

### 7.1383.1 Detailed Description

Bootloader types header file.

### 7.1383.2 Typedef Documentation

#### 7.1383.2.1 blt\_addr

```
typedef unsigned long blt_addr
```

memory address type

#### 7.1383.2.2 blt\_bool

```
typedef unsigned char blt_bool
```

boolean type

#### 7.1383.2.3 blt\_char

```
typedef char blt_char
```

character type

#### 7.1383.2.4 blt\_int16s

```
typedef signed short blt_int16s
```

16-bit signed integer

#### 7.1383.2.5 blt\_int16u

```
typedef unsigned short blt_int16u
```

16-bit unsigned integer

#### 7.1383.2.6 blt\_int32s

```
typedef signed int blt_int32s
```

32-bit signed integer

#### 7.1383.2.7 blt\_int32u

```
typedef unsigned int blt_int32u
```

32-bit unsigned integer

#### 7.1383.2.8 blt\_int64s

```
typedef signed long long blt_int64s
```

64-bit signed integer

#### 7.1383.2.9 blt\_int64u

```
typedef unsigned long long blt_int64u
```

64-bit unsigned integer

#### 7.1383.2.10 blt\_int8s

```
typedef signed char blt_int8s
```

8-bit signed integer

#### 7.1383.2.11 blt\_int8u

```
typedef unsigned char blt_int8u
```

8-bit unsigned integer

## 7.1384 types.h File Reference

Bootloader types header file.

### Macros

- `#define BLT_TRUE (1)`  
*Boolean true value.*
- `#define BLT_FALSE (0)`  
*Boolean false value.*
- `#define BLT_NULL ((void *)0)`  
*NULL pointer value.*

### Typedefs

- `typedef unsigned char blt_bool`
- `typedef char blt_char`
- `typedef unsigned long blt_addr`
- `typedef unsigned char blt_int8u`
- `typedef signed char blt_int8s`
- `typedef unsigned short blt_int16u`
- `typedef signed short blt_int16s`
- `typedef unsigned int blt_int32u`
- `typedef signed int blt_int32s`

### 7.1384.1 Detailed Description

Bootloader types header file.

### 7.1384.2 Typedef Documentation

#### 7.1384.2.1 blt\_addr

```
typedef unsigned long blt_addr
```

memory address type

#### 7.1384.2.2 blt\_bool

```
typedef unsigned char blt_bool
```

boolean type

### 7.1384.2.3 blt\_char

```
typedef char blt_char
```

character type

### 7.1384.2.4 blt\_int16s

```
typedef signed short blt_int16s
```

16-bit signed integer

### 7.1384.2.5 blt\_int16u

```
typedef unsigned short blt_int16u
```

16-bit unsigned integer

### 7.1384.2.6 blt\_int32s

```
typedef signed int blt_int32s
```

32-bit signed integer

### 7.1384.2.7 blt\_int32u

```
typedef unsigned int blt_int32u
```

32-bit unsigned integer

### 7.1384.2.8 blt\_int8s

```
typedef signed char blt_int8s
```

8-bit signed integer

### 7.1384.2.9 blt\_int8u

```
typedef unsigned char blt_int8u
```

8-bit unsigned integer

## 7.1385 types.h File Reference

Bootloader types header file.

## Macros

- `#define BLT_TRUE (1)`  
*Boolean true value.*
- `#define BLT_FALSE (0)`  
*Boolean false value.*
- `#define BLT_NULL ((void *)0)`  
*NULL pointer value.*

## Typedefs

- `typedef unsigned char blt_bool`
- `typedef char blt_char`
- `typedef unsigned long blt_addr`
- `typedef unsigned char blt_int8u`
- `typedef signed char blt_int8s`
- `typedef unsigned short blt_int16u`
- `typedef signed short blt_int16s`
- `typedef unsigned int blt_int32u`
- `typedef signed int blt_int32s`
- `typedef unsigned long long blt_int64u`
- `typedef signed long long blt_int64s`

### 7.1385.1 Detailed Description

Bootloader types header file.

### 7.1385.2 Typedef Documentation

#### 7.1385.2.1 blt\_addr

```
typedef unsigned long blt_addr
```

memory address type

#### 7.1385.2.2 blt\_bool

```
typedef unsigned char blt_bool
```

boolean type

#### 7.1385.2.3 blt\_char

```
typedef char blt_char
```

character type



#### 7.1385.2.4 blt\_int16s

```
typedef signed short blt_int16s
```

16-bit signed integer

#### 7.1385.2.5 blt\_int16u

```
typedef unsigned short blt_int16u
```

16-bit unsigned integer

#### 7.1385.2.6 blt\_int32s

```
typedef signed int blt_int32s
```

32-bit signed integer

#### 7.1385.2.7 blt\_int32u

```
typedef unsigned int blt_int32u
```

32-bit unsigned integer

#### 7.1385.2.8 blt\_int64s

```
typedef signed long long blt_int64s
```

64-bit signed integer

#### 7.1385.2.9 blt\_int64u

```
typedef unsigned long long blt_int64u
```

64-bit unsigned integer

#### 7.1385.2.10 blt\_int8s

```
typedef signed char blt_int8s
```

8-bit signed integer

#### 7.1385.2.11 blt\_int8u

```
typedef unsigned char blt_int8u
```

8-bit unsigned integer

## 7.1386 types.h File Reference

Bootloader types header file.

### Macros

- `#define BLT_TRUE (1)`  
*Boolean true value.*
- `#define BLT_FALSE (0)`  
*Boolean false value.*
- `#define BLT_NULL ((void *)0)`  
*NULL pointer value.*

### Typedefs

- `typedef unsigned char blt_bool`
- `typedef char blt_char`
- `typedef unsigned long blt_addr`
- `typedef unsigned char blt_int8u`
- `typedef signed char blt_int8s`
- `typedef unsigned short blt_int16u`
- `typedef signed short blt_int16s`
- `typedef unsigned int blt_int32u`
- `typedef signed int blt_int32s`

### 7.1386.1 Detailed Description

Bootloader types header file.

### 7.1386.2 Typedef Documentation

#### 7.1386.2.1 blt\_addr

```
typedef unsigned long blt_addr
```

memory address type

#### 7.1386.2.2 blt\_bool

```
typedef unsigned char blt_bool
```

boolean type

### 7.1386.2.3 blt\_char

```
typedef char blt_char
```

character type

### 7.1386.2.4 blt\_int16s

```
typedef signed short blt_int16s
```

16-bit signed integer

### 7.1386.2.5 blt\_int16u

```
typedef unsigned short blt_int16u
```

16-bit unsigned integer

### 7.1386.2.6 blt\_int32s

```
typedef signed int blt_int32s
```

32-bit signed integer

### 7.1386.2.7 blt\_int32u

```
typedef unsigned int blt_int32u
```

32-bit unsigned integer

### 7.1386.2.8 blt\_int8s

```
typedef signed char blt_int8s
```

8-bit signed integer

### 7.1386.2.9 blt\_int8u

```
typedef unsigned char blt_int8u
```

8-bit unsigned integer

## 7.1387 types.h File Reference

Bootloader types header file.

## Macros

- `#define BLT_TRUE (1)`  
*Boolean true value.*
- `#define BLT_FALSE (0)`  
*Boolean false value.*
- `#define BLT_NULL ((void *)0)`  
*NULL pointer value.*

## Typedefs

- `typedef unsigned char blt_bool`
- `typedef char blt_char`
- `typedef unsigned long blt_addr`
- `typedef unsigned char blt_int8u`
- `typedef signed char blt_int8s`
- `typedef unsigned short blt_int16u`
- `typedef signed short blt_int16s`
- `typedef unsigned int blt_int32u`
- `typedef signed int blt_int32s`

### 7.1387.1 Detailed Description

Bootloader types header file.

### 7.1387.2 Typedef Documentation

#### 7.1387.2.1 blt\_addr

```
typedef unsigned long blt_addr
```

memory address type

#### 7.1387.2.2 blt\_bool

```
typedef unsigned char blt_bool
```

boolean type

#### 7.1387.2.3 blt\_char

```
typedef char blt_char
```

character type

#### 7.1387.2.4 blt\_int16s

```
typedef signed short blt_int16s
```

16-bit signed integer

#### 7.1387.2.5 blt\_int16u

```
typedef unsigned short blt_int16u
```

16-bit unsigned integer

#### 7.1387.2.6 blt\_int32s

```
typedef signed int blt_int32s
```

32-bit signed integer

#### 7.1387.2.7 blt\_int32u

```
typedef unsigned int blt_int32u
```

32-bit unsigned integer

#### 7.1387.2.8 blt\_int8s

```
typedef signed char blt_int8s
```

8-bit signed integer

#### 7.1387.2.9 blt\_int8u

```
typedef unsigned char blt_int8u
```

8-bit unsigned integer

## 7.1388 types.h File Reference

Bootloader types header file.

## Macros

- `#define BLT_TRUE (1)`  
*Boolean true value.*
- `#define BLT_FALSE (0)`  
*Boolean false value.*
- `#define BLT_NULL ((void *)0)`  
*NULL pointer value.*

## Typedefs

- `typedef unsigned char blt_bool`
- `typedef char blt_char`
- `typedef unsigned long blt_addr`
- `typedef unsigned char blt_int8u`
- `typedef signed char blt_int8s`
- `typedef unsigned short blt_int16u`
- `typedef signed short blt_int16s`
- `typedef unsigned int blt_int32u`
- `typedef signed int blt_int32s`

### 7.1388.1 Detailed Description

Bootloader types header file.

### 7.1388.2 Typedef Documentation

#### 7.1388.2.1 blt\_addr

```
typedef unsigned long blt_addr
```

memory address type

#### 7.1388.2.2 blt\_bool

```
typedef unsigned char blt_bool
```

boolean type

#### 7.1388.2.3 blt\_char

```
typedef char blt_char
```

character type

#### 7.1388.2.4 blt\_int16s

```
typedef signed short blt_int16s
```

16-bit signed integer

#### 7.1388.2.5 blt\_int16u

```
typedef unsigned short blt_int16u
```

16-bit unsigned integer

#### 7.1388.2.6 blt\_int32s

```
typedef signed int blt_int32s
```

32-bit signed integer

#### 7.1388.2.7 blt\_int32u

```
typedef unsigned int blt_int32u
```

32-bit unsigned integer

#### 7.1388.2.8 blt\_int8s

```
typedef signed char blt_int8s
```

8-bit signed integer

#### 7.1388.2.9 blt\_int8u

```
typedef unsigned char blt_int8u
```

8-bit unsigned integer

## 7.1389 types.h File Reference

Bootloader types header file.

## Macros

- `#define BLT_TRUE (1)`  
*Boolean true value.*
- `#define BLT_FALSE (0)`  
*Boolean false value.*
- `#define BLT_NULL ((void *)0)`  
*NULL pointer value.*

## Typedefs

- `typedef unsigned char blt_bool`
- `typedef char blt_char`
- `typedef unsigned long blt_addr`
- `typedef unsigned char blt_int8u`
- `typedef signed char blt_int8s`
- `typedef unsigned short blt_int16u`
- `typedef signed short blt_int16s`
- `typedef unsigned int blt_int32u`
- `typedef signed int blt_int32s`

### 7.1389.1 Detailed Description

Bootloader types header file.

### 7.1389.2 Typedef Documentation

#### 7.1389.2.1 blt\_addr

```
typedef unsigned long blt_addr
```

memory address type

#### 7.1389.2.2 blt\_bool

```
typedef unsigned char blt_bool
```

boolean type

#### 7.1389.2.3 blt\_char

```
typedef char blt_char
```

character type



#### 7.1389.2.4 blt\_int16s

```
typedef signed short blt_int16s
```

16-bit signed integer

#### 7.1389.2.5 blt\_int16u

```
typedef unsigned short blt_int16u
```

16-bit unsigned integer

#### 7.1389.2.6 blt\_int32s

```
typedef signed int blt_int32s
```

32-bit signed integer

#### 7.1389.2.7 blt\_int32u

```
typedef unsigned int blt_int32u
```

32-bit unsigned integer

#### 7.1389.2.8 blt\_int8s

```
typedef signed char blt_int8s
```

8-bit signed integer

#### 7.1389.2.9 blt\_int8u

```
typedef unsigned char blt_int8u
```

8-bit unsigned integer

## 7.1390 types.h File Reference

Bootloader types header file.

## Macros

- `#define BLT_TRUE (1)`  
*Boolean true value.*
- `#define BLT_FALSE (0)`  
*Boolean false value.*
- `#define BLT_NULL ((void *)0)`  
*NULL pointer value.*

## Typedefs

- `typedef unsigned char blt_bool`
- `typedef char blt_char`
- `typedef unsigned long blt_addr`
- `typedef unsigned char blt_int8u`
- `typedef signed char blt_int8s`
- `typedef unsigned short blt_int16u`
- `typedef signed short blt_int16s`
- `typedef unsigned int blt_int32u`
- `typedef signed int blt_int32s`

### 7.1390.1 Detailed Description

Bootloader types header file.

### 7.1390.2 Typedef Documentation

#### 7.1390.2.1 blt\_addr

```
typedef unsigned long blt_addr
```

memory address type

#### 7.1390.2.2 blt\_bool

```
typedef unsigned char blt_bool
```

boolean type

#### 7.1390.2.3 blt\_char

```
typedef char blt_char
```

character type

#### 7.1390.2.4 blt\_int16s

```
typedef signed short blt_int16s
```

16-bit signed integer

#### 7.1390.2.5 blt\_int16u

```
typedef unsigned short blt_int16u
```

16-bit unsigned integer

#### 7.1390.2.6 blt\_int32s

```
typedef signed int blt_int32s
```

32-bit signed integer

#### 7.1390.2.7 blt\_int32u

```
typedef unsigned int blt_int32u
```

32-bit unsigned integer

#### 7.1390.2.8 blt\_int8s

```
typedef signed char blt_int8s
```

8-bit signed integer

#### 7.1390.2.9 blt\_int8u

```
typedef unsigned char blt_int8u
```

8-bit unsigned integer

## 7.1391 types.h File Reference

Bootloader types header file.

## Macros

- `#define BLT_TRUE (1)`  
*Boolean true value.*
- `#define BLT_FALSE (0)`  
*Boolean false value.*
- `#define BLT_NULL ((void *)0)`  
*NULL pointer value.*

## Typedefs

- `typedef unsigned char blt_bool`
- `typedef char blt_char`
- `typedef unsigned long blt_addr`
- `typedef unsigned char blt_int8u`
- `typedef signed char blt_int8s`
- `typedef unsigned short blt_int16u`
- `typedef signed short blt_int16s`
- `typedef unsigned int blt_int32u`
- `typedef signed int blt_int32s`

### 7.1391.1 Detailed Description

Bootloader types header file.

### 7.1391.2 Typedef Documentation

#### 7.1391.2.1 blt\_addr

```
typedef unsigned long blt_addr
```

memory address type

#### 7.1391.2.2 blt\_bool

```
typedef unsigned char blt_bool
```

boolean type

#### 7.1391.2.3 blt\_char

```
typedef char blt_char
```

character type

#### 7.1391.2.4 blt\_int16s

```
typedef signed short blt_int16s
```

16-bit signed integer

#### 7.1391.2.5 blt\_int16u

```
typedef unsigned short blt_int16u
```

16-bit unsigned integer

#### 7.1391.2.6 blt\_int32s

```
typedef signed int blt_int32s
```

32-bit signed integer

#### 7.1391.2.7 blt\_int32u

```
typedef unsigned int blt_int32u
```

32-bit unsigned integer

#### 7.1391.2.8 blt\_int8s

```
typedef signed char blt_int8s
```

8-bit signed integer

#### 7.1391.2.9 blt\_int8u

```
typedef unsigned char blt_int8u
```

8-bit unsigned integer

## 7.1392 types.h File Reference

Bootloader types header file.

## Macros

- `#define BLT_TRUE (1)`  
*Boolean true value.*
- `#define BLT_FALSE (0)`  
*Boolean false value.*
- `#define BLT_NULL ((void *)0)`  
*NULL pointer value.*

## Typedefs

- `typedef unsigned char blt_bool`
- `typedef char blt_char`
- `typedef unsigned long blt_addr`
- `typedef unsigned char blt_int8u`
- `typedef signed char blt_int8s`
- `typedef unsigned short blt_int16u`
- `typedef signed short blt_int16s`
- `typedef unsigned int blt_int32u`
- `typedef signed int blt_int32s`

### 7.1392.1 Detailed Description

Bootloader types header file.

### 7.1392.2 Typedef Documentation

#### 7.1392.2.1 blt\_addr

```
typedef unsigned long blt_addr
```

memory address type

#### 7.1392.2.2 blt\_bool

```
typedef unsigned char blt_bool
```

boolean type

#### 7.1392.2.3 blt\_char

```
typedef char blt_char
```

character type

#### 7.1392.2.4 blt\_int16s

```
typedef signed short blt_int16s
```

16-bit signed integer

#### 7.1392.2.5 blt\_int16u

```
typedef unsigned short blt_int16u
```

16-bit unsigned integer

#### 7.1392.2.6 blt\_int32s

```
typedef signed int blt_int32s
```

32-bit signed integer

#### 7.1392.2.7 blt\_int32u

```
typedef unsigned int blt_int32u
```

32-bit unsigned integer

#### 7.1392.2.8 blt\_int8s

```
typedef signed char blt_int8s
```

8-bit signed integer

#### 7.1392.2.9 blt\_int8u

```
typedef unsigned char blt_int8u
```

8-bit unsigned integer

## 7.1393 types.h File Reference

Bootloader types header file.

## Macros

- `#define BLT_TRUE (1)`  
*Boolean true value.*
- `#define BLT_FALSE (0)`  
*Boolean false value.*
- `#define BLT_NULL ((void *)0)`  
*NULL pointer value.*

## Typedefs

- `typedef unsigned char blt_bool`
- `typedef char blt_char`
- `typedef unsigned long blt_addr`
- `typedef unsigned char blt_int8u`
- `typedef signed char blt_int8s`
- `typedef unsigned short blt_int16u`
- `typedef signed short blt_int16s`
- `typedef unsigned int blt_int32u`
- `typedef signed int blt_int32s`

### 7.1393.1 Detailed Description

Bootloader types header file.

### 7.1393.2 Typedef Documentation

#### 7.1393.2.1 blt\_addr

```
typedef unsigned long blt_addr
```

memory address type

#### 7.1393.2.2 blt\_bool

```
typedef unsigned char blt_bool
```

boolean type

#### 7.1393.2.3 blt\_char

```
typedef char blt_char
```

character type



#### 7.1393.2.4 blt\_int16s

```
typedef signed short blt_int16s
```

16-bit signed integer

#### 7.1393.2.5 blt\_int16u

```
typedef unsigned short blt_int16u
```

16-bit unsigned integer

#### 7.1393.2.6 blt\_int32s

```
typedef signed int blt_int32s
```

32-bit signed integer

#### 7.1393.2.7 blt\_int32u

```
typedef unsigned int blt_int32u
```

32-bit unsigned integer

#### 7.1393.2.8 blt\_int8s

```
typedef signed char blt_int8s
```

8-bit signed integer

#### 7.1393.2.9 blt\_int8u

```
typedef unsigned char blt_int8u
```

8-bit unsigned integer

## 7.1394 types.h File Reference

Bootloader types header file.

## Macros

- `#define BLT_TRUE (1)`  
*Boolean true value.*
- `#define BLT_FALSE (0)`  
*Boolean false value.*
- `#define BLT_NULL ((void *)0)`  
*NULL pointer value.*

## Typedefs

- `typedef unsigned char blt_bool`
- `typedef char blt_char`
- `typedef unsigned long blt_addr`
- `typedef unsigned char blt_int8u`
- `typedef signed char blt_int8s`
- `typedef unsigned short blt_int16u`
- `typedef signed short blt_int16s`
- `typedef unsigned int blt_int32u`
- `typedef signed int blt_int32s`

### 7.1394.1 Detailed Description

Bootloader types header file.

### 7.1394.2 Typedef Documentation

#### 7.1394.2.1 blt\_addr

```
typedef unsigned long blt_addr
```

memory address type

#### 7.1394.2.2 blt\_bool

```
typedef unsigned char blt_bool
```

boolean type

#### 7.1394.2.3 blt\_char

```
typedef char blt_char
```

character type

#### 7.1394.2.4 blt\_int16s

```
typedef signed short blt_int16s
```

16-bit signed integer

#### 7.1394.2.5 blt\_int16u

```
typedef unsigned short blt_int16u
```

16-bit unsigned integer

#### 7.1394.2.6 blt\_int32s

```
typedef signed int blt_int32s
```

32-bit signed integer

#### 7.1394.2.7 blt\_int32u

```
typedef unsigned int blt_int32u
```

32-bit unsigned integer

#### 7.1394.2.8 blt\_int8s

```
typedef signed char blt_int8s
```

8-bit signed integer

#### 7.1394.2.9 blt\_int8u

```
typedef unsigned char blt_int8u
```

8-bit unsigned integer

## 7.1395 types.h File Reference

Bootloader types header file.

## Macros

- `#define BLT_TRUE (1)`  
*Boolean true value.*
- `#define BLT_FALSE (0)`  
*Boolean false value.*
- `#define BLT_NULL ((void *)0)`  
*NULL pointer value.*

## Typedefs

- `typedef unsigned char blt_bool`
- `typedef char blt_char`
- `typedef unsigned long blt_addr`
- `typedef unsigned char blt_int8u`
- `typedef signed char blt_int8s`
- `typedef unsigned short blt_int16u`
- `typedef signed short blt_int16s`
- `typedef unsigned int blt_int32u`
- `typedef signed int blt_int32s`

### 7.1395.1 Detailed Description

Bootloader types header file.

### 7.1395.2 Typedef Documentation

#### 7.1395.2.1 blt\_addr

```
typedef unsigned long blt_addr
```

memory address type

#### 7.1395.2.2 blt\_bool

```
typedef unsigned char blt_bool
```

boolean type

#### 7.1395.2.3 blt\_char

```
typedef char blt_char
```

character type

#### 7.1395.2.4 blt\_int16s

```
typedef signed short blt_int16s
```

16-bit signed integer

#### 7.1395.2.5 blt\_int16u

```
typedef unsigned short blt_int16u
```

16-bit unsigned integer

#### 7.1395.2.6 blt\_int32s

```
typedef signed int blt_int32s
```

32-bit signed integer

#### 7.1395.2.7 blt\_int32u

```
typedef unsigned int blt_int32u
```

32-bit unsigned integer

#### 7.1395.2.8 blt\_int8s

```
typedef signed char blt_int8s
```

8-bit signed integer

#### 7.1395.2.9 blt\_int8u

```
typedef unsigned char blt_int8u
```

8-bit unsigned integer

## 7.1396 types.h File Reference

Bootloader types header file.

## Macros

- `#define BLT_TRUE (1)`  
*Boolean true value.*
- `#define BLT_FALSE (0)`  
*Boolean false value.*
- `#define BLT_NULL ((void *)0)`  
*NULL pointer value.*

## Typedefs

- `typedef unsigned char blt_bool`
- `typedef char blt_char`
- `typedef unsigned long blt_addr`
- `typedef unsigned char blt_int8u`
- `typedef signed char blt_int8s`
- `typedef unsigned short blt_int16u`
- `typedef signed short blt_int16s`
- `typedef unsigned int blt_int32u`
- `typedef signed int blt_int32s`

### 7.1396.1 Detailed Description

Bootloader types header file.

### 7.1396.2 Typedef Documentation

#### 7.1396.2.1 blt\_addr

```
typedef unsigned long blt_addr
```

memory address type

#### 7.1396.2.2 blt\_bool

```
typedef unsigned char blt_bool
```

boolean type

#### 7.1396.2.3 blt\_char

```
typedef char blt_char
```

character type

#### 7.1396.2.4 blt\_int16s

```
typedef signed short blt_int16s
```

16-bit signed integer

#### 7.1396.2.5 blt\_int16u

```
typedef unsigned short blt_int16u
```

16-bit unsigned integer

#### 7.1396.2.6 blt\_int32s

```
typedef signed int blt_int32s
```

32-bit signed integer

#### 7.1396.2.7 blt\_int32u

```
typedef unsigned int blt_int32u
```

32-bit unsigned integer

#### 7.1396.2.8 blt\_int8s

```
typedef signed char blt_int8s
```

8-bit signed integer

#### 7.1396.2.9 blt\_int8u

```
typedef unsigned char blt_int8u
```

8-bit unsigned integer

## 7.1397 types.h File Reference

Bootloader types header file.

## Macros

- `#define BLT_TRUE (1)`  
*Boolean true value.*
- `#define BLT_FALSE (0)`  
*Boolean false value.*
- `#define BLT_NULL ((void *)0)`  
*NULL pointer value.*

## Typedefs

- `typedef unsigned char blt_bool`
- `typedef char blt_char`
- `typedef unsigned long blt_addr`
- `typedef unsigned char blt_int8u`
- `typedef signed char blt_int8s`
- `typedef unsigned short blt_int16u`
- `typedef signed short blt_int16s`
- `typedef unsigned int blt_int32u`
- `typedef signed int blt_int32s`

### 7.1397.1 Detailed Description

Bootloader types header file.

### 7.1397.2 Typedef Documentation

#### 7.1397.2.1 blt\_addr

```
typedef unsigned long blt_addr
```

memory address type

#### 7.1397.2.2 blt\_bool

```
typedef unsigned char blt_bool
```

boolean type

#### 7.1397.2.3 blt\_char

```
typedef char blt_char
```

character type



#### 7.1397.2.4 blt\_int16s

```
typedef signed short blt_int16s
```

16-bit signed integer

#### 7.1397.2.5 blt\_int16u

```
typedef unsigned short blt_int16u
```

16-bit unsigned integer

#### 7.1397.2.6 blt\_int32s

```
typedef signed int blt_int32s
```

32-bit signed integer

#### 7.1397.2.7 blt\_int32u

```
typedef unsigned int blt_int32u
```

32-bit unsigned integer

#### 7.1397.2.8 blt\_int8s

```
typedef signed char blt_int8s
```

8-bit signed integer

#### 7.1397.2.9 blt\_int8u

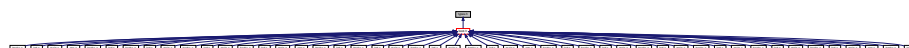
```
typedef unsigned char blt_int8u
```

8-bit unsigned integer

## 7.1398 types.h File Reference

Bootloader types header file.

This graph shows which files directly or indirectly include this file:



## Macros

- `#define BLT_TRUE (1)`  
*Boolean true value.*
- `#define BLT_FALSE (0)`  
*Boolean false value.*
- `#define BLT_NULL ((void *)0)`  
*NULL pointer value.*

## Typedefs

- `typedef unsigned char blt_bool`
- `typedef char blt_char`
- `typedef unsigned long blt_addr`
- `typedef unsigned char blt_int8u`
- `typedef signed char blt_int8s`
- `typedef unsigned short blt_int16u`
- `typedef signed short blt_int16s`
- `typedef unsigned long blt_int32u`
- `typedef signed long blt_int32s`

### 7.1398.1 Detailed Description

Bootloader types header file.

### 7.1398.2 Typedef Documentation

#### 7.1398.2.1 blt\_addr

```
typedef unsigned long blt_addr
```

memory address type

#### 7.1398.2.2 blt\_bool

```
typedef unsigned char blt_bool
```

boolean type

#### 7.1398.2.3 blt\_char

```
typedef char blt_char
```

character type

#### 7.1398.2.4 blt\_int16s

```
typedef signed short blt_int16s
```

16-bit signed integer

#### 7.1398.2.5 blt\_int16u

```
typedef unsigned short blt_int16u
```

16-bit unsigned integer

#### 7.1398.2.6 blt\_int32s

```
typedef signed long blt_int32s
```

32-bit signed integer

#### 7.1398.2.7 blt\_int32u

```
typedef unsigned long blt_int32u
```

32-bit unsigned integer

#### 7.1398.2.8 blt\_int8s

```
typedef signed char blt_int8s
```

8-bit signed integer

#### 7.1398.2.9 blt\_int8u

```
typedef unsigned char blt_int8u
```

8-bit unsigned integer

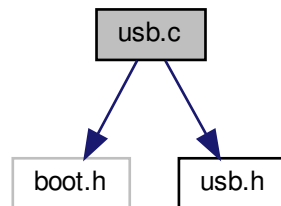
## 7.1399 usb.c File Reference

Bootloader USB communication interface source file.

```
#include "boot.h"
```

```
#include "usb.h"
```

Include dependency graph for \_template/usb.c:



### Data Structures

- struct [tFifoCtrl](#)  
*Structure type for fifo control.*
- struct [tFifoPipe](#)  
*Structure type for a fifo pipe.*

### Macros

- #define [FIFO\\_MAX\\_BUFFERS](#) (2)  
*Total number of fifo buffers.*
- #define [FIFO\\_ERR\\_INVALID\\_HANDLE](#) (255)  
*Invalid value for a fifo buffer handle.*
- #define [FIFO\\_PIPE\\_SIZE](#) (64)  
*Number of bytes that fit in the fifo pipe.*
- #define [BULK\\_DATA\\_MAX\\_PACKET\\_SIZE](#) (64)  
*Endpoint IN & OUT Packet size.*

### Functions

- static [blt\\_bool](#) [UsbReceiveByte](#) ([blt\\_int8u](#) \*data)  
*Receives a communication interface byte if one is present.*
- static [blt\\_bool](#) [UsbTransmitByte](#) ([blt\\_int8u](#) data)  
*Transmits a communication interface byte.*
- static void [UsbFifoMgrInit](#) (void)  
*Initializes the fifo manager. Each controlled fifo is assigned a unique handle, which is the same as its index into `fifoCtrl[]`. Each controlled fifo holds a pointer to the next free fifo control. For the last fifo in `fifoCtrl[]` this one is set to a null-pointer as an out of fifo control indicator. Function should be called once before any of the other fifo management functions are called.*

- static `blt_int8u` `UsbFifoMgrCreate` (`blt_int8u` \*buffer, `blt_int8u` length)  
*Places a data storage array under fifo management control. A handle for identifying the fifo in subsequent fifo management function calls is returned, if successful.*
- static `blt_bool` `UsbFifoMgrWrite` (`blt_int8u` handle, `blt_int8u` data)  
*Stores data in the fifo.*
- static `blt_bool` `UsbFifoMgrRead` (`blt_int8u` handle, `blt_int8u` \*data)  
*Retrieves data from the fifo.*
- static `blt_int8u` `UsbFifoMgrScan` (`blt_int8u` handle)  
*Returns the number of data entries currently present in the fifo.*
- void `UsbInit` (void)  
*Initializes the USB communication interface.*
- void `UsbFree` (void)  
*Releases the USB communication interface.*
- void `UsbTransmitPacket` (`blt_int8u` \*data, `blt_int8u` len)  
*Transmits a packet formatted for the communication interface.*
- `blt_bool` `UsbReceivePacket` (`blt_int8u` \*data, `blt_int8u` \*len)  
*Receives a communication interface packet if one is present.*
- void `UsbTransmitPipeBulkIN` (void)  
*Checks if there is still data left to transmit and if so submits it for transmission with the USB endpoint.*
- void `UsbReceivePipeBulkOUT` (void)  
*Stores data that was received on the Bulk OUT pipe in the fifo.*

## Variables

- static `tFifoCtrl` `fifoCtrl` [`FIFO_MAX_BUFFERS`]  
*Local variable that holds the fifo control structures.*
- static `tFifoCtrl` \* `fifoCtrlFree`  
*Local pointer that points to the next free fifo control structure.*
- static `tFifoPipe` `fifoPipeBulkIN`  
*Fifo pipe used for the bulk in endpoint.*
- static `tFifoPipe` `fifoPipeBulkOUT`  
*Fifo pipe used for the bulk out endpoint.*

## 7.1399.1 Detailed Description

Bootloader USB communication interface source file.

## 7.1399.2 Function Documentation

### 7.1399.2.1 UsbFifoMgrCreate()

```
static blt_int8u UsbFifoMgrCreate (
 blt_int8u * buffer,
 blt_int8u length) [static]
```

Places a data storage array under fifo management control. A handle for identifying the fifo in subsequent fifo management function calls is returned, if successful.

**Parameters**

|               |                                                                 |
|---------------|-----------------------------------------------------------------|
| <i>buffer</i> | Pointer to the first element in the data storage fifo.          |
| <i>length</i> | Maximum number of data elements that can be stored in the fifo. |

**Returns**

Fifo handle if successfull, or FIFO\_ERR\_INVALID\_HANDLE.

Referenced by UsbInit().

**7.1399.2.2 UsbFifoMgrInit()**

```
static void UsbFifoMgrInit (
 void) [static]
```

Initializes the fifo manager. Each controlled fifo is assigned a unique handle, which is the same as its index into fifoCtrl[]. Each controlled fifo holds a pointer to the next free fifo control. For the last fifo in fifoCtrl[] this one is set to a null-pointer as an out of fifo control indicator. Function should be called once before any of the other fifo management functions are called.

**Returns**

none.

Referenced by UsbInit().

**7.1399.2.3 UsbFifoMgrRead()**

```
static blt_bool UsbFifoMgrRead (
 blt_int8u handle,
 blt_int8u * data) [static]
```

Retrieves data from the fifo.

**Parameters**

|               |                                                 |
|---------------|-------------------------------------------------|
| <i>handle</i> | Identifies the fifo to read data from.          |
| <i>data</i>   | Pointer to where the read data is to be stored. |

**Returns**

BLT\_TRUE if the data was successfully read from the fifo, BLT\_FALSE otherwise.

Referenced by UsbReceiveByte(), and UsbTransmitPipeBulkIN().

#### 7.1399.2.4 UsbFifoMgrScan()

```
static blt_int8u UsbFifoMgrScan (
 blt_int8u handle) [static]
```

Returns the number of data entries currently present in the fifo.

##### Parameters

|               |                                            |
|---------------|--------------------------------------------|
| <i>handle</i> | Identifies the fifo that is to be scanned. |
|---------------|--------------------------------------------|

##### Returns

Number of data entries in the fifo if successful, otherwise 0.

Referenced by UsbTransmitPipeBulkIN().

#### 7.1399.2.5 UsbFifoMgrWrite()

```
static blt_bool UsbFifoMgrWrite (
 blt_int8u handle,
 blt_int8u data) [static]
```

Stores data in the fifo.

##### Parameters

|               |                                                        |
|---------------|--------------------------------------------------------|
| <i>handle</i> | Identifies the fifo to write data to.                  |
| <i>data</i>   | Pointer to the data that is to be written to the fifo. |

##### Returns

BLT\_TRUE if the data was successfully stored in the fifo, BLT\_FALSE otherwise.

Referenced by UsbReceivePipeBulkOUT(), and UsbTransmitByte().

#### 7.1399.2.6 UsbFree()

```
void UsbFree (
 void)
```

Releases the USB communication interface.

##### Returns

none.

Referenced by ComFree().

### 7.1399.2.7 UsbInit()

```
void UsbInit (
 void)
```

Initializes the USB communication interface.

#### Returns

none.

Referenced by ComInit().

### 7.1399.2.8 UsbReceiveByte()

```
static blt_bool UsbReceiveByte (
 blt_int8u * data) [static]
```

Receives a communication interface byte if one is present.

#### Parameters

|             |                                                 |
|-------------|-------------------------------------------------|
| <i>data</i> | Pointer to byte where the data is to be stored. |
|-------------|-------------------------------------------------|

#### Returns

BLT\_TRUE if a byte was received, BLT\_FALSE otherwise.

Referenced by UsbReceivePacket().

### 7.1399.2.9 UsbReceivePacket()

```
blt_bool UsbReceivePacket (
 blt_int8u * data,
 blt_int8u * len)
```

Receives a communication interface packet if one is present.

#### Parameters

|             |                                                         |
|-------------|---------------------------------------------------------|
| <i>data</i> | Pointer to byte array where the data is to be stored.   |
| <i>len</i>  | Pointer where the length of the packet is to be stored. |



**Returns**

BLT\_TRUE if a packet was received, BLT\_FALSE otherwise.

Referenced by ComTask().

**7.1399.2.10 UsbReceivePipeBulkOUT()**

```
void UsbReceivePipeBulkOUT (
 void)
```

Stores data that was received on the Bulk OUT pipe in the fifo.

**Returns**

none.

**7.1399.2.11 UsbTransmitByte()**

```
static blt_bool UsbTransmitByte (
 blt_int8u data) [static]
```

Transmits a communication interface byte.

**Parameters**

|             |                                          |
|-------------|------------------------------------------|
| <i>data</i> | Value of byte that is to be transmitted. |
|-------------|------------------------------------------|

**Returns**

BLT\_TRUE if the byte was transmitted, BLT\_FALSE otherwise.

Referenced by UsbTransmitPacket().

**7.1399.2.12 UsbTransmitPacket()**

```
void UsbTransmitPacket (
 blt_int8u * data,
 blt_int8u len)
```

Transmits a packet formatted for the communication interface.

**Parameters**

|             |                                                            |
|-------------|------------------------------------------------------------|
| <i>data</i> | Pointer to byte array with data that it to be transmitted. |
| <i>len</i>  | Number of bytes that are to be transmitted.                |

**Returns**

none.

Referenced by ComTransmitPacket().

**7.1399.2.13 UsbTransmitPipeBulkIN()**

```
static void UsbTransmitPipeBulkIN (
 void)
```

Checks if there is still data left to transmit and if so submits it for transmission with the USB endpoint.

**Returns**

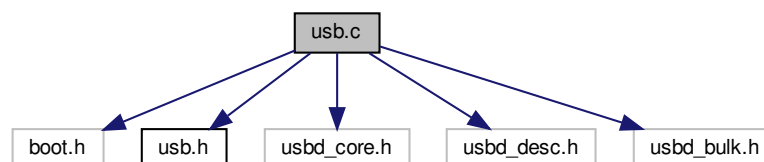
none.

**7.1400 usb.c File Reference**

Bootloader USB communication interface source file.

```
#include "boot.h"
#include "usb.h"
#include "usbd_core.h"
#include "usbd_desc.h"
#include "usbd_bulk.h"
```

Include dependency graph for ARMCM33\_STM32L5/usb.c:

**Data Structures**

- struct [tFifoCtrl](#)  
Structure type for fifo control.
- struct [tFifoPipe](#)  
Structure type for a fifo pipe.

## Macros

- #define `FIFO_MAX_BUFFERS` (2)  
*Total number of fifo buffers.*
- #define `FIFO_ERR_INVALID_HANDLE` (255)  
*Invalid value for a fifo buffer handle.*
- #define `FIFO_PIPE_SIZE` (64)  
*Number of bytes that fit in the fifo pipe.*

## Functions

- static `blt_bool` `UsbReceiveByte` (`blt_int8u` \*data)  
*Receives a communication interface byte if one is present.*
- static `blt_bool` `UsbTransmitByte` (`blt_int8u` data)  
*Transmits a communication interface byte.*
- static void `UsbFifoMgrInit` (void)  
*Initializes the fifo manager. Each controlled fifo is assigned a unique handle, which is the same as its index into `fifoCtrl[]`. Each controlled fifo holds a pointer to the next free fifo control. For the last fifo in `fifoCtrl[]` this one is set to a null-pointer as an out of fifo control indicator. Function should be called once before any of the other fifo management functions are called.*
- static `blt_int8u` `UsbFifoMgrCreate` (`blt_int8u` \*buffer, `blt_int8u` length)  
*Places a data storage array under fifo management control. A handle for identifying the fifo in subsequent fifo management function calls is returned, if successful.*
- static `blt_bool` `UsbFifoMgrWrite` (`blt_int8u` handle, `blt_int8u` data)  
*Stores data in the fifo.*
- static `blt_bool` `UsbFifoMgrRead` (`blt_int8u` handle, `blt_int8u` \*data)  
*Retrieves data from the fifo.*
- static `blt_int8u` `UsbFifoMgrScan` (`blt_int8u` handle)  
*Returns the number of data entries currently present in the fifo.*
- void `UsbInit` (void)  
*Initializes the USB communication interface.*
- void `UsbFree` (void)  
*Releases the USB communication interface.*
- void `UsbTransmitPacket` (`blt_int8u` \*data, `blt_int8u` len)  
*Transmits a packet formatted for the communication interface.*
- `blt_bool` `UsbReceivePacket` (`blt_int8u` \*data, `blt_int8u` \*len)  
*Receives a communication interface packet if one is present.*
- void `UsbTransmitPipeBulkIN` (void)  
*Checks if there is still data left to transmit and if so submits it for transmission with the USB endpoint.*
- void `UsbReceivePipeBulkOUT` (`blt_int8u` epnum)  
*Stores data that was received on the Bulk OUT pipe in the fifo.*

## Variables

- static `tFifoCtrl` `fifoCtrl` [`FIFO_MAX_BUFFERS`]  
*Local variable that holds the fifo control structures.*
- static `tFifoCtrl` \* `fifoCtrlFree`  
*Local pointer that points to the next free fifo control structure.*
- static `tFifoPipe` `fifoPipeBulkIN`  
*Fifo pipe used for the bulk in endpoint.*
- static `tFifoPipe` `fifoPipeBulkOUT`  
*Fifo pipe used for the bulk out endpoint.*
- static `USBD_HandleTypeDef` `hUsbDeviceFS`  
*USB device handle.*

### 7.1400.1 Detailed Description

Bootloader USB communication interface source file.

### 7.1400.2 Function Documentation

#### 7.1400.2.1 UsbFifoMgrCreate()

```
static blt_int8u UsbFifoMgrCreate (
 blt_int8u * buffer,
 blt_int8u length) [static]
```

Places a data storage array under fifo management control. A handle for identifying the fifo in subsequent fifo management function calls is returned, if successful.

##### Parameters

|               |                                                                 |
|---------------|-----------------------------------------------------------------|
| <i>buffer</i> | Pointer to the first element in the data storage fifo.          |
| <i>length</i> | Maximum number of data elements that can be stored in the fifo. |

##### Returns

Fifo handle if successfull, or FIFO\_ERR\_INVALID\_HANDLE.

Referenced by UsbInit().

#### 7.1400.2.2 UsbFifoMgrInit()

```
static void UsbFifoMgrInit (
 void) [static]
```

Initializes the fifo manager. Each controlled fifo is assigned a unique handle, which is the same as its index into fifoCtrl[]. Each controlled fifo holds a pointer to the next free fifo control. For the last fifo in fifoCtrl[] this one is set to a null-pointer as an out of fifo control indicator. Function should be called once before any of the other fifo management functions are called.

##### Returns

none.

Referenced by UsbInit().

#### 7.1400.2.3 UsbFifoMgrRead()

```
static blt_bool UsbFifoMgrRead (
 blt_int8u handle,
 blt_int8u * data) [static]
```

Retrieves data from the fifo.

**Parameters**

|               |                                                 |
|---------------|-------------------------------------------------|
| <i>handle</i> | Identifies the fifo to read data from.          |
| <i>data</i>   | Pointer to where the read data is to be stored. |

**Returns**

BLT\_TRUE if the data was successfully read from the fifo, BLT\_FALSE otherwise.

Referenced by UsbReceiveByte(), and UsbTransmitPipeBulkIN().

**7.1400.2.4 UsbFifoMgrScan()**

```
static blt_int8u UsbFifoMgrScan (
 blt_int8u handle) [static]
```

Returns the number of data entries currently present in the fifo.

**Parameters**

|               |                                            |
|---------------|--------------------------------------------|
| <i>handle</i> | Identifies the fifo that is to be scanned. |
|---------------|--------------------------------------------|

**Returns**

Number of data entries in the fifo if successful, otherwise 0.

Referenced by UsbTransmitPipeBulkIN().

**7.1400.2.5 UsbFifoMgrWrite()**

```
static blt_bool UsbFifoMgrWrite (
 blt_int8u handle,
 blt_int8u data) [static]
```

Stores data in the fifo.

**Parameters**

|               |                                                        |
|---------------|--------------------------------------------------------|
| <i>handle</i> | Identifies the fifo to write data to.                  |
| <i>data</i>   | Pointer to the data that is to be written to the fifo. |

**Returns**

BLT\_TRUE if the data was successfully stored in the fifo, BLT\_FALSE otherwise.

Referenced by `UsbReceivePipeBulkOUT()`, and `UsbTransmitByte()`.

#### 7.1400.2.6 `UsbFree()`

```
void UsbFree (
 void)
```

Releases the USB communication interface.

##### Returns

none.

#### 7.1400.2.7 `UsbInit()`

```
void UsbInit (
 void)
```

Initializes the USB communication interface.

##### Returns

none.

#### 7.1400.2.8 `UsbReceiveByte()`

```
static blt_bool UsbReceiveByte (
 blt_int8u * data) [static]
```

Receives a communication interface byte if one is present.

##### Parameters

|             |                                                 |
|-------------|-------------------------------------------------|
| <i>data</i> | Pointer to byte where the data is to be stored. |
|-------------|-------------------------------------------------|

##### Returns

BLT\_TRUE if a byte was received, BLT\_FALSE otherwise.

Referenced by `UsbReceivePacket()`.

### 7.1400.2.9 UsbReceivePacket()

```
blt_bool UsbReceivePacket (
 blt_int8u * data,
 blt_int8u * len)
```

Receives a communication interface packet if one is present.

#### Parameters

|             |                                                         |
|-------------|---------------------------------------------------------|
| <i>data</i> | Pointer to byte array where the data is to be stored.   |
| <i>len</i>  | Pointer where the length of the packet is to be stored. |

#### Returns

BLT\_TRUE if a packet was received, BLT\_FALSE otherwise.

### 7.1400.2.10 UsbReceivePipeBulkOUT()

```
void UsbReceivePipeBulkOUT (
 blt_int8u epnum)
```

Stores data that was received on the Bulk OUT pipe in the fifo.

#### Returns

none.

### 7.1400.2.11 UsbTransmitByte()

```
static blt_bool UsbTransmitByte (
 blt_int8u data) [static]
```

Transmits a communication interface byte.

#### Parameters

|             |                                          |
|-------------|------------------------------------------|
| <i>data</i> | Value of byte that is to be transmitted. |
|-------------|------------------------------------------|

#### Returns

BLT\_TRUE if the byte was transmitted, BLT\_FALSE otherwise.

Referenced by UsbTransmitPacket().

### 7.1400.2.12 UsbTransmitPacket()

```
void UsbTransmitPacket (
 blt_int8u * data,
 blt_int8u len)
```

Transmits a packet formatted for the communication interface.

#### Parameters

|             |                                                            |
|-------------|------------------------------------------------------------|
| <i>data</i> | Pointer to byte array with data that it to be transmitted. |
| <i>len</i>  | Number of bytes that are to be transmitted.                |

#### Returns

none.

### 7.1400.2.13 UsbTransmitPipeBulkIN()

```
void UsbTransmitPipeBulkIN (
 void)
```

Checks if there is still data left to transmit and if so submits it for transmission with the USB endpoint.

#### Returns

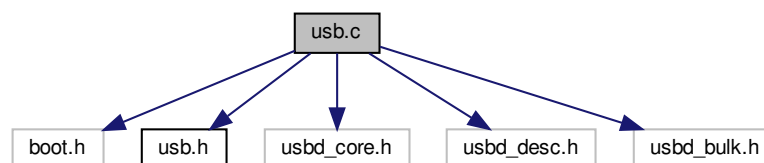
none.

## 7.1401 usb.c File Reference

Bootloader USB communication interface source file.

```
#include "boot.h"
#include "usb.h"
#include "usbd_core.h"
#include "usbd_desc.h"
#include "usbd_bulk.h"
```

Include dependency graph for ARMCM3\_STM32F1/usb.c:





## Data Structures

- struct [tFifoCtrl](#)  
*Structure type for fifo control.*
- struct [tFifoPipe](#)  
*Structure type for a fifo pipe.*

## Macros

- #define [FIFO\\_MAX\\_BUFFERS](#) (2)  
*Total number of fifo buffers.*
- #define [FIFO\\_ERR\\_INVALID\\_HANDLE](#) (255)  
*Invalid value for a fifo buffer handle.*
- #define [FIFO\\_PIPE\\_SIZE](#) (64)  
*Number of bytes that fit in the fifo pipe.*

## Functions

- static [blt\\_bool](#) [UsbReceiveByte](#) ([blt\\_int8u](#) \*data)  
*Receives a communication interface byte if one is present.*
- static [blt\\_bool](#) [UsbTransmitByte](#) ([blt\\_int8u](#) data)  
*Transmits a communication interface byte.*
- static void [UsbFifoMgrInit](#) (void)  
*Initializes the fifo manager. Each controlled fifo is assigned a unique handle, which is the same as its index into `fifoCtrl[]`. Each controlled fifo holds a pointer to the next free fifo control. For the last fifo in `fifoCtrl[]` this one is set to a null-pointer as an out of fifo control indicator. Function should be called once before any of the other fifo management functions are called.*
- static [blt\\_int8u](#) [UsbFifoMgrCreate](#) ([blt\\_int8u](#) \*buffer, [blt\\_int8u](#) length)  
*Places a data storage array under fifo management control. A handle for identifying the fifo in subsequent fifo management function calls is returned, if successful.*
- static [blt\\_bool](#) [UsbFifoMgrWrite](#) ([blt\\_int8u](#) handle, [blt\\_int8u](#) data)  
*Stores data in the fifo.*
- static [blt\\_bool](#) [UsbFifoMgrRead](#) ([blt\\_int8u](#) handle, [blt\\_int8u](#) \*data)  
*Retrieves data from the fifo.*
- static [blt\\_int8u](#) [UsbFifoMgrScan](#) ([blt\\_int8u](#) handle)  
*Returns the number of data entries currently present in the fifo.*
- void [UsbInit](#) (void)  
*Initializes the USB communication interface.*
- void [UsbFree](#) (void)  
*Releases the USB communication interface.*
- void [UsbTransmitPacket](#) ([blt\\_int8u](#) \*data, [blt\\_int8u](#) len)  
*Transmits a packet formatted for the communication interface.*
- [blt\\_bool](#) [UsbReceivePacket](#) ([blt\\_int8u](#) \*data, [blt\\_int8u](#) \*len)  
*Receives a communication interface packet if one is present.*
- void [UsbTransmitPipeBulkIN](#) (void)  
*Checks if there is still data left to transmit and if so submits it for transmission with the USB endpoint.*
- void [UsbReceivePipeBulkOUT](#) ([blt\\_int8u](#) epnum)  
*Stores data that was received on the Bulk OUT pipe in the fifo.*

## Variables

- static `tFifoCtrl fifoCtrl` [FIFO\_MAX\_BUFFERS]  
*Local variable that holds the fifo control structures.*
- static `tFifoCtrl * fifoCtrlFree`  
*Local pointer that points to the next free fifo control structure.*
- static `tFifoPipe fifoPipeBulkIN`  
*Fifo pipe used for the bulk in endpoint.*
- static `tFifoPipe fifoPipeBulkOUT`  
*Fifo pipe used for the bulk out endpoint.*
- static `USBD_HandleTypeDef hUsbDeviceFS`  
*USB device handle.*

### 7.1401.1 Detailed Description

Bootloader USB communication interface source file.

### 7.1401.2 Function Documentation

#### 7.1401.2.1 UsbFifoMgrCreate()

```
static blt_int8u UsbFifoMgrCreate (
 blt_int8u * buffer,
 blt_int8u length) [static]
```

Places a data storage array under fifo management control. A handle for identifying the fifo in subsequent fifo management function calls is returned, if successful.

#### Parameters

|               |                                                                 |
|---------------|-----------------------------------------------------------------|
| <i>buffer</i> | Pointer to the first element in the data storage fifo.          |
| <i>length</i> | Maximum number of data elements that can be stored in the fifo. |

#### Returns

Fifo handle if successfull, or FIFO\_ERR\_INVALID\_HANDLE.

Referenced by `UsbInit()`.

#### 7.1401.2.2 UsbFifoMgrInit()

```
static void UsbFifoMgrInit (
 void) [static]
```

Initializes the fifo manager. Each controlled fifo is assigned a unique handle, which is the same as its index into `fifoCtrl[]`. Each controlled fifo holds a pointer to the next free fifo control. For the last fifo in `fifoCtrl[]` this one is set to a null-pointer as an out of fifo control indicator. Function should be called once before any of the other fifo management functions are called.

#### Returns

none.

Referenced by `UsbInit()`.

#### 7.1401.2.3 UsbFifoMgrRead()

```
static blt_bool UsbFifoMgrRead (
 blt_int8u handle,
 blt_int8u * data) [static]
```

Retrieves data from the fifo.

#### Parameters

|               |                                                 |
|---------------|-------------------------------------------------|
| <i>handle</i> | Identifies the fifo to read data from.          |
| <i>data</i>   | Pointer to where the read data is to be stored. |

#### Returns

BLT\_TRUE if the data was successfully read from the fifo, BLT\_FALSE otherwise.

Referenced by `UsbReceiveByte()`, and `UsbTransmitPipeBulkIN()`.

#### 7.1401.2.4 UsbFifoMgrScan()

```
static blt_int8u UsbFifoMgrScan (
 blt_int8u handle) [static]
```

Returns the number of data entries currently present in the fifo.

#### Parameters

|               |                                            |
|---------------|--------------------------------------------|
| <i>handle</i> | Identifies the fifo that is to be scanned. |
|---------------|--------------------------------------------|

#### Returns

Number of data entries in the fifo if successful, otherwise 0.

Referenced by `UsbTransmitPipeBulkIN()`.

#### 7.1401.2.5 UsbFifoMgrWrite()

```
static blt_bool UsbFifoMgrWrite (
 blt_int8u handle,
 blt_int8u data) [static]
```

Stores data in the fifo.

##### Parameters

|               |                                                        |
|---------------|--------------------------------------------------------|
| <i>handle</i> | Identifies the fifo to write data to.                  |
| <i>data</i>   | Pointer to the data that is to be written to the fifo. |

##### Returns

BLT\_TRUE if the data was successfully stored in the fifo, BLT\_FALSE otherwise.

Referenced by UsbReceivePipeBulkOUT(), and UsbTransmitByte().

#### 7.1401.2.6 UsbFree()

```
void UsbFree (
 void)
```

Releases the USB communication interface.

##### Returns

none.

#### 7.1401.2.7 UsbInit()

```
void UsbInit (
 void)
```

Initializes the USB communication interface.

##### Returns

none.

#### 7.1401.2.8 UsbReceiveByte()

```
static blt_bool UsbReceiveByte (
 blt_int8u * data) [static]
```

Receives a communication interface byte if one is present.

## Parameters

|             |                                                 |
|-------------|-------------------------------------------------|
| <i>data</i> | Pointer to byte where the data is to be stored. |
|-------------|-------------------------------------------------|

## Returns

BLT\_TRUE if a byte was received, BLT\_FALSE otherwise.

Referenced by UsbReceivePacket().

## 7.1401.2.9 UsbReceivePacket()

```
blt_bool UsbReceivePacket (
 blt_int8u * data,
 blt_int8u * len)
```

Receives a communication interface packet if one is present.

## Parameters

|             |                                                         |
|-------------|---------------------------------------------------------|
| <i>data</i> | Pointer to byte array where the data is to be stored.   |
| <i>len</i>  | Pointer where the length of the packet is to be stored. |

## Returns

BLT\_TRUE if a packet was received, BLT\_FALSE otherwise.

## 7.1401.2.10 UsbReceivePipeBulkOUT()

```
void UsbReceivePipeBulkOUT (
 blt_int8u epnum)
```

Stores data that was received on the Bulk OUT pipe in the fifo.

## Returns

none.

## 7.1401.2.11 UsbTransmitByte()

```
static blt_bool UsbTransmitByte (
 blt_int8u data) [static]
```

Transmits a communication interface byte.

**Parameters**

|             |                                          |
|-------------|------------------------------------------|
| <i>data</i> | Value of byte that is to be transmitted. |
|-------------|------------------------------------------|

**Returns**

BLT\_TRUE if the byte was transmitted, BLT\_FALSE otherwise.

Referenced by UsbTransmitPacket().

**7.1401.2.12 UsbTransmitPacket()**

```
void UsbTransmitPacket (
 blt_int8u * data,
 blt_int8u len)
```

Transmits a packet formatted for the communication interface.

**Parameters**

|             |                                                            |
|-------------|------------------------------------------------------------|
| <i>data</i> | Pointer to byte array with data that it to be transmitted. |
| <i>len</i>  | Number of bytes that are to be transmitted.                |

**Returns**

none.

**7.1401.2.13 UsbTransmitPipeBulkIN()**

```
void UsbTransmitPipeBulkIN (
 void)
```

Checks if there is still data left to transmit and if so submits it for transmission with the USB endpoint.

**Returns**

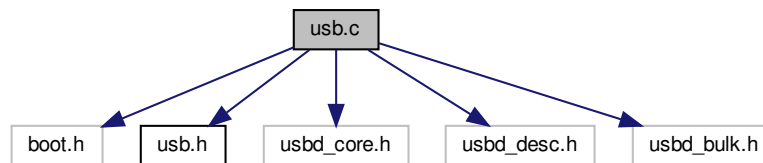
none.

## 7.1402 usb.c File Reference

Bootloader USB communication interface source file.

```
#include "boot.h"
#include "usb.h"
#include "usbd_core.h"
#include "usbd_desc.h"
#include "usbd_bulk.h"
```

Include dependency graph for ARMCM4\_STM32F3/usb.c:



### Data Structures

- struct [tFifoCtrl](#)  
*Structure type for fifo control.*
- struct [tFifoPipe](#)  
*Structure type for a fifo pipe.*

### Macros

- #define [FIFO\\_MAX\\_BUFFERS](#) (2)  
*Total number of fifo buffers.*
- #define [FIFO\\_ERR\\_INVALID\\_HANDLE](#) (255)  
*Invalid value for a fifo buffer handle.*
- #define [FIFO\\_PIPE\\_SIZE](#) (64)  
*Number of bytes that fit in the fifo pipe.*

### Functions

- static [blt\\_bool](#) [UsbReceiveByte](#) ([blt\\_int8u](#) \*data)  
*Receives a communication interface byte if one is present.*
- static [blt\\_bool](#) [UsbTransmitByte](#) ([blt\\_int8u](#) data)  
*Transmits a communication interface byte.*
- static void [UsbFifoMgrInit](#) (void)  
*Initializes the fifo manager. Each controlled fifo is assigned a unique handle, which is the same as its index into `fifoCtrl[]`. Each controlled fifo holds a pointer to the next free fifo control. For the last fifo in `fifoCtrl[]` this one is set to a null-pointer as an out of fifo control indicator. Function should be called once before any of the other fifo management functions are called.*
- static [blt\\_int8u](#) [UsbFifoMgrCreate](#) ([blt\\_int8u](#) \*buffer, [blt\\_int8u](#) length)

*Places a data storage array under fifo management control. A handle for identifying the fifo in subsequent fifo management function calls is returned, if successful.*

- static `blt_bool` `UsbFifoMgrWrite` (`blt_int8u` handle, `blt_int8u` data)  
*Stores data in the fifo.*
- static `blt_bool` `UsbFifoMgrRead` (`blt_int8u` handle, `blt_int8u` \*data)  
*Retrieves data from the fifo.*
- static `blt_int8u` `UsbFifoMgrScan` (`blt_int8u` handle)  
*Returns the number of data entries currently present in the fifo.*
- void `UsbInit` (void)  
*Initializes the USB communication interface.*
- void `UsbFree` (void)  
*Releases the USB communication interface.*
- void `UsbTransmitPacket` (`blt_int8u` \*data, `blt_int8u` len)  
*Transmits a packet formatted for the communication interface.*
- `blt_bool` `UsbReceivePacket` (`blt_int8u` \*data, `blt_int8u` \*len)  
*Receives a communication interface packet if one is present.*
- void `UsbTransmitPipeBulkIN` (void)  
*Checks if there is still data left to transmit and if so submits it for transmission with the USB endpoint.*
- void `UsbReceivePipeBulkOUT` (`blt_int8u` epnum)  
*Stores data that was received on the Bulk OUT pipe in the fifo.*

## Variables

- static `tFifoCtrl` `fifoCtrl` [`FIFO_MAX_BUFFERS`]  
*Local variable that holds the fifo control structures.*
- static `tFifoCtrl` \* `fifoCtrlFree`  
*Local pointer that points to the next free fifo control structure.*
- static `tFifoPipe` `fifoPipeBulkIN`  
*Fifo pipe used for the bulk in endpoint.*
- static `tFifoPipe` `fifoPipeBulkOUT`  
*Fifo pipe used for the bulk out endpoint.*
- static `USBD_HandleTypeDef` `hUsbDeviceFS`  
*USB device handle.*

## 7.1402.1 Detailed Description

Bootloader USB communication interface source file.

## 7.1402.2 Function Documentation

### 7.1402.2.1 UsbFifoMgrCreate()

```
static blt_int8u UsbFifoMgrCreate (
 blt_int8u * buffer,
 blt_int8u length) [static]
```

Places a data storage array under fifo management control. A handle for identifying the fifo in subsequent fifo management function calls is returned, if successful.



**Parameters**

|               |                                                                 |
|---------------|-----------------------------------------------------------------|
| <i>buffer</i> | Pointer to the first element in the data storage fifo.          |
| <i>length</i> | Maximum number of data elements that can be stored in the fifo. |

**Returns**

Fifo handle if successfull, or FIFO\_ERR\_INVALID\_HANDLE.

Referenced by UsbInit().

**7.1402.2.2 UsbFifoMgrInit()**

```
static void UsbFifoMgrInit (
 void) [static]
```

Initializes the fifo manager. Each controlled fifo is assigned a unique handle, which is the same as its index into fifoCtrl[]. Each controlled fifo holds a pointer to the next free fifo control. For the last fifo in fifoCtrl[] this one is set to a null-pointer as an out of fifo control indicator. Function should be called once before any of the other fifo management functions are called.

**Returns**

none.

Referenced by UsbInit().

**7.1402.2.3 UsbFifoMgrRead()**

```
static blt_bool UsbFifoMgrRead (
 blt_int8u handle,
 blt_int8u * data) [static]
```

Retrieves data from the fifo.

**Parameters**

|               |                                                 |
|---------------|-------------------------------------------------|
| <i>handle</i> | Identifies the fifo to read data from.          |
| <i>data</i>   | Pointer to where the read data is to be stored. |

**Returns**

BLT\_TRUE if the data was successfully read from the fifo, BLT\_FALSE otherwise.

Referenced by UsbReceiveByte(), and UsbTransmitPipeBulkIN().

#### 7.1402.2.4 UsbFifoMgrScan()

```
static blt_int8u UsbFifoMgrScan (
 blt_int8u handle) [static]
```

Returns the number of data entries currently present in the fifo.

##### Parameters

|               |                                            |
|---------------|--------------------------------------------|
| <i>handle</i> | Identifies the fifo that is to be scanned. |
|---------------|--------------------------------------------|

##### Returns

Number of data entries in the fifo if successful, otherwise 0.

Referenced by UsbTransmitPipeBulkIN().

#### 7.1402.2.5 UsbFifoMgrWrite()

```
static blt_bool UsbFifoMgrWrite (
 blt_int8u handle,
 blt_int8u data) [static]
```

Stores data in the fifo.

##### Parameters

|               |                                                        |
|---------------|--------------------------------------------------------|
| <i>handle</i> | Identifies the fifo to write data to.                  |
| <i>data</i>   | Pointer to the data that is to be written to the fifo. |

##### Returns

BLT\_TRUE if the data was successfully stored in the fifo, BLT\_FALSE otherwise.

Referenced by UsbReceivePipeBulkOUT(), and UsbTransmitByte().

#### 7.1402.2.6 UsbFree()

```
void UsbFree (
 void)
```

Releases the USB communication interface.

##### Returns

none.

### 7.1402.2.7 UsbInit()

```
void UsbInit (
 void)
```

Initializes the USB communication interface.

#### Returns

none.

### 7.1402.2.8 UsbReceiveByte()

```
static blt_bool UsbReceiveByte (
 blt_int8u * data) [static]
```

Receives a communication interface byte if one is present.

#### Parameters

|             |                                                 |
|-------------|-------------------------------------------------|
| <i>data</i> | Pointer to byte where the data is to be stored. |
|-------------|-------------------------------------------------|

#### Returns

BLT\_TRUE if a byte was received, BLT\_FALSE otherwise.

Referenced by UsbReceivePacket().

### 7.1402.2.9 UsbReceivePacket()

```
blt_bool UsbReceivePacket (
 blt_int8u * data,
 blt_int8u * len)
```

Receives a communication interface packet if one is present.

#### Parameters

|             |                                                         |
|-------------|---------------------------------------------------------|
| <i>data</i> | Pointer to byte array where the data is to be stored.   |
| <i>len</i>  | Pointer where the length of the packet is to be stored. |

#### Returns

BLT\_TRUE if a packet was received, BLT\_FALSE otherwise.

#### 7.1402.2.10 UsbReceivePipeBulkOUT()

```
void UsbReceivePipeBulkOUT (
 blt_int8u epnum)
```

Stores data that was received on the Bulk OUT pipe in the fifo.

##### Returns

none.

#### 7.1402.2.11 UsbTransmitByte()

```
static blt_bool UsbTransmitByte (
 blt_int8u data) [static]
```

Transmits a communication interface byte.

##### Parameters

|             |                                          |
|-------------|------------------------------------------|
| <i>data</i> | Value of byte that is to be transmitted. |
|-------------|------------------------------------------|

##### Returns

BLT\_TRUE if the byte was transmitted, BLT\_FALSE otherwise.

Referenced by UsbTransmitPacket().

#### 7.1402.2.12 UsbTransmitPacket()

```
void UsbTransmitPacket (
 blt_int8u * data,
 blt_int8u len)
```

Transmits a packet formatted for the communication interface.

##### Parameters

|             |                                                            |
|-------------|------------------------------------------------------------|
| <i>data</i> | Pointer to byte array with data that it to be transmitted. |
| <i>len</i>  | Number of bytes that are to be transmitted.                |

##### Returns

none.

### 7.1402.2.13 UsbTransmitPipeBulkIN()

```
void UsbTransmitPipeBulkIN (
 void)
```

Checks if there is still data left to transmit and if so submits it for transmission with the USB endpoint.

#### Returns

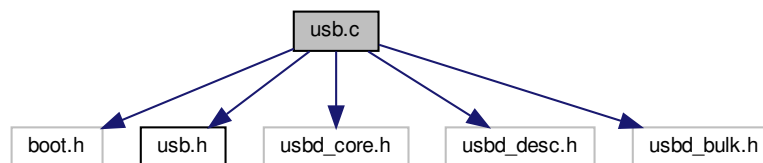
none.

## 7.1403 usb.c File Reference

Bootloader USB communication interface source file.

```
#include "boot.h"
#include "usb.h"
#include "usbd_core.h"
#include "usbd_desc.h"
#include "usbd_bulk.h"
```

Include dependency graph for ARMCM4\_STM32F4/usb.c:



### Data Structures

- struct [tFifoCtrl](#)  
*Structure type for fifo control.*
- struct [tFifoPipe](#)  
*Structure type for a fifo pipe.*

### Macros

- #define [FIFO\\_MAX\\_BUFFERS](#) (2)  
*Total number of fifo buffers.*
- #define [FIFO\\_ERR\\_INVALID\\_HANDLE](#) (255)  
*Invalid value for a fifo buffer handle.*
- #define [FIFO\\_PIPE\\_SIZE](#) (64)  
*Number of bytes that fit in the fifo pipe.*

## Functions

- static `blt_bool` `UsbReceiveByte` (`blt_int8u` \*data)  
*Receives a communication interface byte if one is present.*
- static `blt_bool` `UsbTransmitByte` (`blt_int8u` data)  
*Transmits a communication interface byte.*
- static void `UsbFifoMgrInit` (void)  
*Initializes the fifo manager. Each controlled fifo is assigned a unique handle, which is the same as its index into `fifoCtrl[]`. Each controlled fifo holds a pointer to the next free fifo control. For the last fifo in `fifoCtrl[]` this one is set to a null-pointer as an out of fifo control indicator. Function should be called once before any of the other fifo management functions are called.*
- static `blt_int8u` `UsbFifoMgrCreate` (`blt_int8u` \*buffer, `blt_int8u` length)  
*Places a data storage array under fifo management control. A handle for identifying the fifo in subsequent fifo management function calls is returned, if successful.*
- static `blt_bool` `UsbFifoMgrWrite` (`blt_int8u` handle, `blt_int8u` data)  
*Stores data in the fifo.*
- static `blt_bool` `UsbFifoMgrRead` (`blt_int8u` handle, `blt_int8u` \*data)  
*Retrieves data from the fifo.*
- static `blt_int8u` `UsbFifoMgrScan` (`blt_int8u` handle)  
*Returns the number of data entries currently present in the fifo.*
- void `UsbInit` (void)  
*Initializes the USB communication interface.*
- void `UsbFree` (void)  
*Releases the USB communication interface.*
- void `UsbTransmitPacket` (`blt_int8u` \*data, `blt_int8u` len)  
*Transmits a packet formatted for the communication interface.*
- `blt_bool` `UsbReceivePacket` (`blt_int8u` \*data, `blt_int8u` \*len)  
*Receives a communication interface packet if one is present.*
- void `UsbTransmitPipeBulkIN` (void)  
*Checks if there is still data left to transmit and if so submits it for transmission with the USB endpoint.*
- void `UsbReceivePipeBulkOUT` (`blt_int8u` epnum)  
*Stores data that was received on the Bulk OUT pipe in the fifo.*

## Variables

- static `tFifoCtrl` `fifoCtrl` [`FIFO_MAX_BUFFERS`]  
*Local variable that holds the fifo control structures.*
- static `tFifoCtrl` \* `fifoCtrlFree`  
*Local pointer that points to the next free fifo control structure.*
- static `tFifoPipe` `fifoPipeBulkIN`  
*Fifo pipe used for the bulk in endpoint.*
- static `tFifoPipe` `fifoPipeBulkOUT`  
*Fifo pipe used for the bulk out endpoint.*
- static `USBD_HandleTypeDef` `hUsbDeviceFS`  
*USB device handle.*

### 7.1403.1 Detailed Description

Bootloader USB communication interface source file.

## 7.1403.2 Function Documentation

### 7.1403.2.1 UsbFifoMgrCreate()

```
static blt_int8u UsbFifoMgrCreate (
 blt_int8u * buffer,
 blt_int8u length) [static]
```

Places a data storage array under fifo management control. A handle for identifying the fifo in subsequent fifo management function calls is returned, if successful.

#### Parameters

|               |                                                                 |
|---------------|-----------------------------------------------------------------|
| <i>buffer</i> | Pointer to the first element in the data storage fifo.          |
| <i>length</i> | Maximum number of data elements that can be stored in the fifo. |

#### Returns

Fifo handle if successful, or FIFO\_ERR\_INVALID\_HANDLE.

Referenced by UsbInit().

### 7.1403.2.2 UsbFifoMgrInit()

```
static void UsbFifoMgrInit (
 void) [static]
```

Initializes the fifo manager. Each controlled fifo is assigned a unique handle, which is the same as its index into fifoCtrl[]. Each controlled fifo holds a pointer to the next free fifo control. For the last fifo in fifoCtrl[] this one is set to a null-pointer as an out of fifo control indicator. Function should be called once before any of the other fifo management functions are called.

#### Returns

none.

Referenced by UsbInit().

### 7.1403.2.3 UsbFifoMgrRead()

```
static blt_bool UsbFifoMgrRead (
 blt_int8u handle,
 blt_int8u * data) [static]
```

Retrieves data from the fifo.

**Parameters**

|               |                                                 |
|---------------|-------------------------------------------------|
| <i>handle</i> | Identifies the fifo to read data from.          |
| <i>data</i>   | Pointer to where the read data is to be stored. |

**Returns**

BLT\_TRUE if the data was successfully read from the fifo, BLT\_FALSE otherwise.

Referenced by UsbReceiveByte(), and UsbTransmitPipeBulkIN().

**7.1403.2.4 UsbFifoMgrScan()**

```
static blt_int8u UsbFifoMgrScan (
 blt_int8u handle) [static]
```

Returns the number of data entries currently present in the fifo.

**Parameters**

|               |                                            |
|---------------|--------------------------------------------|
| <i>handle</i> | Identifies the fifo that is to be scanned. |
|---------------|--------------------------------------------|

**Returns**

Number of data entries in the fifo if successful, otherwise 0.

Referenced by UsbTransmitPipeBulkIN().

**7.1403.2.5 UsbFifoMgrWrite()**

```
static blt_bool UsbFifoMgrWrite (
 blt_int8u handle,
 blt_int8u data) [static]
```

Stores data in the fifo.

**Parameters**

|               |                                                        |
|---------------|--------------------------------------------------------|
| <i>handle</i> | Identifies the fifo to write data to.                  |
| <i>data</i>   | Pointer to the data that is to be written to the fifo. |

**Returns**

BLT\_TRUE if the data was successfully stored in the fifo, BLT\_FALSE otherwise.



Referenced by `UsbReceivePipeBulkOUT()`, and `UsbTransmitByte()`.

#### 7.1403.2.6 `UsbFree()`

```
void UsbFree (
 void)
```

Releases the USB communication interface.

##### Returns

none.

#### 7.1403.2.7 `UsbInit()`

```
void UsbInit (
 void)
```

Initializes the USB communication interface.

##### Returns

none.

#### 7.1403.2.8 `UsbReceiveByte()`

```
static blt_bool UsbReceiveByte (
 blt_int8u * data) [static]
```

Receives a communication interface byte if one is present.

##### Parameters

|             |                                                 |
|-------------|-------------------------------------------------|
| <i>data</i> | Pointer to byte where the data is to be stored. |
|-------------|-------------------------------------------------|

##### Returns

BLT\_TRUE if a byte was received, BLT\_FALSE otherwise.

Referenced by `UsbReceivePacket()`.

#### 7.1403.2.9 UsbReceivePacket()

```
blt_bool UsbReceivePacket (
 blt_int8u * data,
 blt_int8u * len)
```

Receives a communication interface packet if one is present.

##### Parameters

|             |                                                         |
|-------------|---------------------------------------------------------|
| <i>data</i> | Pointer to byte array where the data is to be stored.   |
| <i>len</i>  | Pointer where the length of the packet is to be stored. |

##### Returns

BLT\_TRUE if a packet was received, BLT\_FALSE otherwise.

#### 7.1403.2.10 UsbReceivePipeBulkOUT()

```
void UsbReceivePipeBulkOUT (
 blt_int8u epnum)
```

Stores data that was received on the Bulk OUT pipe in the fifo.

##### Returns

none.

#### 7.1403.2.11 UsbTransmitByte()

```
static blt_bool UsbTransmitByte (
 blt_int8u data) [static]
```

Transmits a communication interface byte.

##### Parameters

|             |                                          |
|-------------|------------------------------------------|
| <i>data</i> | Value of byte that is to be transmitted. |
|-------------|------------------------------------------|

##### Returns

BLT\_TRUE if the byte was transmitted, BLT\_FALSE otherwise.

Referenced by UsbTransmitPacket().

## 7.1403.2.12 UsbTransmitPacket()

```
void UsbTransmitPacket (
 blt_int8u * data,
 blt_int8u len)
```

Transmits a packet formatted for the communication interface.

## Parameters

|             |                                                            |
|-------------|------------------------------------------------------------|
| <i>data</i> | Pointer to byte array with data that it to be transmitted. |
| <i>len</i>  | Number of bytes that are to be transmitted.                |

## Returns

none.

## 7.1403.2.13 UsbTransmitPipeBulkIN()

```
void UsbTransmitPipeBulkIN (
 void)
```

Checks if there is still data left to transmit and if so submits it for transmission with the USB endpoint.

## Returns

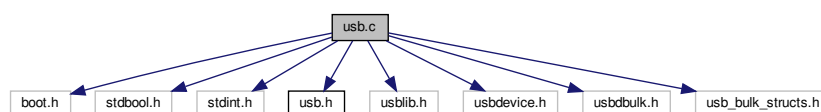
none.

## 7.1404 usb.c File Reference

Bootloader USB communication interface source file.

```
#include "boot.h"
#include <stdbool.h>
#include <stdint.h>
#include "usb.h"
#include "usblib.h"
#include "usbdevice.h"
#include "usdbulk.h"
#include "usb_bulk_structs.h"
```

Include dependency graph for ARMCM4\_TM4C/usb.c:



## Data Structures

- struct [tFifoCtrl](#)  
*Structure type for fifo control.*
- struct [tFifoPipe](#)  
*Structure type for a fifo pipe.*

## Macros

- #define [FIFO\\_MAX\\_BUFFERS](#) (2)  
*Total number of fifo buffers.*
- #define [FIFO\\_ERR\\_INVALID\\_HANDLE](#) (255)  
*Invalid value for a fifo buffer handle.*
- #define [FIFO\\_PIPE\\_SIZE](#) (64)  
*Number of bytes that fit in the fifo pipe.*

## Functions

- static [blt\\_bool](#) [UsbReceiveByte](#) ([blt\\_int8u](#) \*data)  
*Receives a communication interface byte if one is present.*
- static [blt\\_bool](#) [UsbTransmitByte](#) ([blt\\_int8u](#) data)  
*Transmits a communication interface byte.*
- static void [UsbTransmitPipeBulkIN](#) (void)  
*Checks if there is still data left to transmit and if so submits it for transmission with the USB endpoint.*
- static void [UsbReceivePipeBulkOUT](#) ([blt\\_int8u](#) \*data, [blt\\_int32u](#) len)  
*Stores data that was received on the Bulk OUT pipe in the fifo.*
- static void [UsbFifoMgrInit](#) (void)  
*Initializes the fifo manager. Each controlled fifo is assigned a unique handle, which is the same as its index into [fifoCtrl\[\]](#). Each controlled fifo holds a pointer to the next free fifo control. For the last fifo in [fifoCtrl\[\]](#) this one is set to a null-pointer as an out of fifo control indicator. Function should be called once before any of the other fifo management functions are called.*
- static [blt\\_int8u](#) [UsbFifoMgrCreate](#) ([blt\\_int8u](#) \*buffer, [blt\\_int8u](#) length)  
*Places a data storage array under fifo management control. A handle for identifying the fifo in subsequent fifo management function calls is returned, if successful.*
- static [blt\\_bool](#) [UsbFifoMgrWrite](#) ([blt\\_int8u](#) handle, [blt\\_int8u](#) data)  
*Stores data in the fifo.*
- static [blt\\_bool](#) [UsbFifoMgrRead](#) ([blt\\_int8u](#) handle, [blt\\_int8u](#) \*data)  
*Retrieves data from the fifo.*
- static [blt\\_int8u](#) [UsbFifoMgrScan](#) ([blt\\_int8u](#) handle)  
*Returns the number of data entries currently present in the fifo.*
- void [UsbInit](#) (void)  
*Initializes the USB communication interface.*
- void [UsbFree](#) (void)  
*Releases the USB communication interface.*
- void [UsbTransmitPacket](#) ([blt\\_int8u](#) \*data, [blt\\_int8u](#) len)  
*Transmits a packet formatted for the communication interface.*
- [blt\\_bool](#) [UsbReceivePacket](#) ([blt\\_int8u](#) \*data, [blt\\_int8u](#) \*len)  
*Receives a communication interface packet if one is present.*
- [uint32\\_t](#) [UsbBulkTxHandler](#) (void \*pvCBData, [uint32\\_t](#) ui32Event, [uint32\\_t](#) ui32MsgValue, void \*pvMsgData)  
*Handles bulk driver notifications related to the transmit channel (data to the USB host).*
- [uint32\\_t](#) [UsbBulkRxHandler](#) (void \*pvCBData, [uint32\\_t](#) ui32Event, [uint32\\_t](#) ui32MsgValue, void \*pvMsgData)  
*Handles bulk driver notifications related to the receive channel (data from the USB host).*

## Variables

- static `tFifoCtrl fifoCtrl` [FIFO\_MAX\_BUFFERS]  
*Local variable that holds the fifo control structures.*
- static `tFifoCtrl * fifoCtrlFree`  
*Local pointer that points to the next free fifo control structure.*
- static `tFifoPipe fifoPipeBulkIN`  
*Fifo pipe used for the bulk in endpoint.*
- static `tFifoPipe fifoPipeBulkOUT`  
*Fifo pipe used for the bulk out endpoint.*
- static `tBulkInstance * instanceHandle`  
*Holds a handle to the bulk device instance.*

### 7.1404.1 Detailed Description

Bootloader USB communication interface source file.

### 7.1404.2 Function Documentation

#### 7.1404.2.1 UsbBulkRxHandler()

```
uint32_t UsbBulkRxHandler (
 void * pvCBData,
 uint32_t ui32Event,
 uint32_t ui32MsgValue,
 void * pvMsgData)
```

Handles bulk driver notifications related to the receive channel (data from the USB host).

#### Parameters

|                     |                                                           |
|---------------------|-----------------------------------------------------------|
| <i>pvCBData</i>     | is the client-supplied callback pointer for this channel. |
| <i>ui32Event</i>    | identifies the event we are being notified about.         |
| <i>ui32MsgValue</i> | is an event-specific value.                               |
| <i>pvMsgData</i>    | is an event-specific pointer.                             |

#### Returns

The return value is event-specific.

#### 7.1404.2.2 UsbBulkTxHandler()

```
uint32_t UsbBulkTxHandler (
 void * pvCBData,
```

```
uint32_t ui32Event,
uint32_t ui32MsgValue,
void * pvMsgData)
```

Handles bulk driver notifications related to the transmit channel (data to the USB host).

#### Parameters

|                     |                                                           |
|---------------------|-----------------------------------------------------------|
| <i>pvCBData</i>     | is the client-supplied callback pointer for this channel. |
| <i>ui32Event</i>    | identifies the event we are being notified about.         |
| <i>ui32MsgValue</i> | is an event-specific value.                               |
| <i>pvMsgData</i>    | is an event-specific pointer.                             |

#### Returns

The return value is event-specific.

#### 7.1404.2.3 UsbFifoMgrCreate()

```
static blt_int8u UsbFifoMgrCreate (
 blt_int8u * buffer,
 blt_int8u length) [static]
```

Places a data storage array under fifo management control. A handle for identifying the fifo in subsequent fifo management function calls is returned, if successful.

#### Parameters

|               |                                                                 |
|---------------|-----------------------------------------------------------------|
| <i>buffer</i> | Pointer to the first element in the data storage fifo.          |
| <i>length</i> | Maximum number of data elements that can be stored in the fifo. |

#### Returns

Fifo handle if successfull, or FIFO\_ERR\_INVALID\_HANDLE.

Referenced by UsbInit().

#### 7.1404.2.4 UsbFifoMgrInit()

```
static void UsbFifoMgrInit (
 void) [static]
```

Initializes the fifo manager. Each controlled fifo is assigned a unique handle, which is the same as its index into `fifoCtrl[]`. Each controlled fifo holds a pointer to the next free fifo control. For the last fifo in `fifoCtrl[]` this one is set to a null-pointer as an out of fifo control indicator. Function should be called once before any of the other fifo management functions are called.

**Returns**

none.

Referenced by UsbInit().

**7.1404.2.5 UsbFifoMgrRead()**

```
static blt_bool UsbFifoMgrRead (
 blt_int8u handle,
 blt_int8u * data) [static]
```

Retrieves data from the fifo.

**Parameters**

|               |                                                 |
|---------------|-------------------------------------------------|
| <i>handle</i> | Identifies the fifo to read data from.          |
| <i>data</i>   | Pointer to where the read data is to be stored. |

**Returns**

BLT\_TRUE if the data was successfully read from the fifo, BLT\_FALSE otherwise.

Referenced by UsbReceiveByte(), and UsbTransmitPipeBulkIN().

**7.1404.2.6 UsbFifoMgrScan()**

```
static blt_int8u UsbFifoMgrScan (
 blt_int8u handle) [static]
```

Returns the number of data entries currently present in the fifo.

**Parameters**

|               |                                            |
|---------------|--------------------------------------------|
| <i>handle</i> | Identifies the fifo that is to be scanned. |
|---------------|--------------------------------------------|

**Returns**

Number of data entries in the fifo if successful, otherwise 0.

Referenced by UsbTransmitPipeBulkIN().

#### 7.1404.2.7 UsbFifoMgrWrite()

```
static blt_bool UsbFifoMgrWrite (
 blt_int8u handle,
 blt_int8u data) [static]
```

Stores data in the fifo.

##### Parameters

|               |                                                        |
|---------------|--------------------------------------------------------|
| <i>handle</i> | Identifies the fifo to write data to.                  |
| <i>data</i>   | Pointer to the data that is to be written to the fifo. |

##### Returns

BLT\_TRUE if the data was successfully stored in the fifo, BLT\_FALSE otherwise.

Referenced by UsbReceivePipeBulkOUT(), and UsbTransmitByte().

#### 7.1404.2.8 UsbFree()

```
void UsbFree (
 void)
```

Releases the USB communication interface.

##### Returns

none.

#### 7.1404.2.9 UsbInit()

```
void UsbInit (
 void)
```

Initializes the USB communication interface.

##### Returns

none.

#### 7.1404.2.10 UsbReceiveByte()

```
static blt_bool UsbReceiveByte (
 blt_int8u * data) [static]
```

Receives a communication interface byte if one is present.



**Parameters**

|             |                                                 |
|-------------|-------------------------------------------------|
| <i>data</i> | Pointer to byte where the data is to be stored. |
|-------------|-------------------------------------------------|

**Returns**

BLT\_TRUE if a byte was received, BLT\_FALSE otherwise.

Referenced by UsbReceivePacket().

**7.1404.2.11 UsbReceivePacket()**

```
blt_bool UsbReceivePacket (
 blt_int8u * data,
 blt_int8u * len)
```

Receives a communication interface packet if one is present.

**Parameters**

|             |                                                         |
|-------------|---------------------------------------------------------|
| <i>data</i> | Pointer to byte array where the data is to be stored.   |
| <i>len</i>  | Pointer where the length of the packet is to be stored. |

**Returns**

BLT\_TRUE if a packet was received, BLT\_FALSE otherwise.

**7.1404.2.12 UsbReceivePipeBulkOUT()**

```
static void UsbReceivePipeBulkOUT (
 blt_int8u * data,
 blt_int32u len) [static]
```

Stores data that was received on the Bulk OUT pipe in the fifo.

**Parameters**

|             |                                                |
|-------------|------------------------------------------------|
| <i>data</i> | Pointer to buffer with the newly received data |
| <i>len</i>  | Number of received data bytes.                 |

**Returns**

none.

Referenced by `UsbBulkRxHandler()`.

#### 7.1404.2.13 `UsbTransmitByte()`

```
static blt_bool UsbTransmitByte (
 blt_int8u data) [static]
```

Transmits a communication interface byte.

##### Parameters

|             |                                          |
|-------------|------------------------------------------|
| <i>data</i> | Value of byte that is to be transmitted. |
|-------------|------------------------------------------|

##### Returns

BLT\_TRUE if the byte was transmitted, BLT\_FALSE otherwise.

Referenced by `UsbTransmitPacket()`.

#### 7.1404.2.14 `UsbTransmitPacket()`

```
void UsbTransmitPacket (
 blt_int8u * data,
 blt_int8u len)
```

Transmits a packet formatted for the communication interface.

##### Parameters

|             |                                                            |
|-------------|------------------------------------------------------------|
| <i>data</i> | Pointer to byte array with data that it to be transmitted. |
| <i>len</i>  | Number of bytes that are to be transmitted.                |

##### Returns

none.

#### 7.1404.2.15 `UsbTransmitPipeBulkIN()`

```
static void UsbTransmitPipeBulkIN (
 void) [static]
```

Checks if there is still data left to transmit and if so submits it for transmission with the USB endpoint.

### Returns

none.

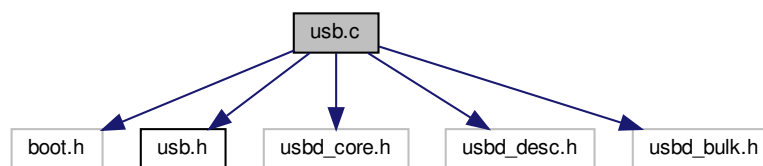
Referenced by `UsbBulkTxHandler()`, and `UsbTransmitPacket()`.

## 7.1405 usb.c File Reference

Bootloader USB communication interface source file.

```
#include "boot.h"
#include "usb.h"
#include "usbd_core.h"
#include "usbd_desc.h"
#include "usbd_bulk.h"
```

Include dependency graph for `ARMCM7_STM32F7/usb.c`:



### Data Structures

- struct [tFifoCtrl](#)  
*Structure type for fifo control.*
- struct [tFifoPipe](#)  
*Structure type for a fifo pipe.*

### Macros

- #define [FIFO\\_MAX\\_BUFFERS](#) (2)  
*Total number of fifo buffers.*
- #define [FIFO\\_ERR\\_INVALID\\_HANDLE](#) (255)  
*Invalid value for a fifo buffer handle.*
- #define [FIFO\\_PIPE\\_SIZE](#) (64)  
*Number of bytes that fit in the fifo pipe.*

## Functions

- static `blt_bool` `UsbReceiveByte` (`blt_int8u` \*data)  
*Receives a communication interface byte if one is present.*
- static `blt_bool` `UsbTransmitByte` (`blt_int8u` data)  
*Transmits a communication interface byte.*
- static void `UsbFifoMgrInit` (void)  
*Initializes the fifo manager. Each controlled fifo is assigned a unique handle, which is the same as its index into `fifoCtrl[]`. Each controlled fifo holds a pointer to the next free fifo control. For the last fifo in `fifoCtrl[]` this one is set to a null-pointer as an out of fifo control indicator. Function should be called once before any of the other fifo management functions are called.*
- static `blt_int8u` `UsbFifoMgrCreate` (`blt_int8u` \*buffer, `blt_int8u` length)  
*Places a data storage array under fifo management control. A handle for identifying the fifo in subsequent fifo management function calls is returned, if successful.*
- static `blt_bool` `UsbFifoMgrWrite` (`blt_int8u` handle, `blt_int8u` data)  
*Stores data in the fifo.*
- static `blt_bool` `UsbFifoMgrRead` (`blt_int8u` handle, `blt_int8u` \*data)  
*Retrieves data from the fifo.*
- static `blt_int8u` `UsbFifoMgrScan` (`blt_int8u` handle)  
*Returns the number of data entries currently present in the fifo.*
- void `UsbInit` (void)  
*Initializes the USB communication interface.*
- void `UsbFree` (void)  
*Releases the USB communication interface.*
- void `UsbTransmitPacket` (`blt_int8u` \*data, `blt_int8u` len)  
*Transmits a packet formatted for the communication interface.*
- `blt_bool` `UsbReceivePacket` (`blt_int8u` \*data, `blt_int8u` \*len)  
*Receives a communication interface packet if one is present.*
- void `UsbTransmitPipeBulkIN` (void)  
*Checks if there is still data left to transmit and if so submits it for transmission with the USB endpoint.*
- void `UsbReceivePipeBulkOUT` (`blt_int8u` epnum)  
*Stores data that was received on the Bulk OUT pipe in the fifo.*

## Variables

- static `tFifoCtrl` `fifoCtrl` [`FIFO_MAX_BUFFERS`]  
*Local variable that holds the fifo control structures.*
- static `tFifoCtrl` \* `fifoCtrlFree`  
*Local pointer that points to the next free fifo control structure.*
- static `tFifoPipe` `fifoPipeBulkIN`  
*Fifo pipe used for the bulk in endpoint.*
- static `tFifoPipe` `fifoPipeBulkOUT`  
*Fifo pipe used for the bulk out endpoint.*
- static `USBD_HandleTypeDef` `hUsbDeviceFS`  
*USB device handle.*

### 7.1405.1 Detailed Description

Bootloader USB communication interface source file.

## 7.1405.2 Function Documentation

### 7.1405.2.1 UsbFifoMgrCreate()

```
static blt_int8u UsbFifoMgrCreate (
 blt_int8u * buffer,
 blt_int8u length) [static]
```

Places a data storage array under fifo management control. A handle for identifying the fifo in subsequent fifo management function calls is returned, if successful.

#### Parameters

|               |                                                                 |
|---------------|-----------------------------------------------------------------|
| <i>buffer</i> | Pointer to the first element in the data storage fifo.          |
| <i>length</i> | Maximum number of data elements that can be stored in the fifo. |

#### Returns

Fifo handle if successfull, or FIFO\_ERR\_INVALID\_HANDLE.

Referenced by UsbInit().

### 7.1405.2.2 UsbFifoMgrInit()

```
static void UsbFifoMgrInit (
 void) [static]
```

Initializes the fifo manager. Each controlled fifo is assigned a unique handle, which is the same as its index into fifoCtrl[]. Each controlled fifo holds a pointer to the next free fifo control. For the last fifo in fifoCtrl[] this one is set to a null-pointer as an out of fifo control indicator. Function should be called once before any of the other fifo management functions are called.

#### Returns

none.

Referenced by UsbInit().

### 7.1405.2.3 UsbFifoMgrRead()

```
static blt_bool UsbFifoMgrRead (
 blt_int8u handle,
 blt_int8u * data) [static]
```

Retrieves data from the fifo.

**Parameters**

|               |                                                 |
|---------------|-------------------------------------------------|
| <i>handle</i> | Identifies the fifo to read data from.          |
| <i>data</i>   | Pointer to where the read data is to be stored. |

**Returns**

BLT\_TRUE if the data was successfully read from the fifo, BLT\_FALSE otherwise.

Referenced by UsbReceiveByte(), and UsbTransmitPipeBulkIN().

**7.1405.2.4 UsbFifoMgrScan()**

```
static blt_int8u UsbFifoMgrScan (
 blt_int8u handle) [static]
```

Returns the number of data entries currently present in the fifo.

**Parameters**

|               |                                            |
|---------------|--------------------------------------------|
| <i>handle</i> | Identifies the fifo that is to be scanned. |
|---------------|--------------------------------------------|

**Returns**

Number of data entries in the fifo if successful, otherwise 0.

Referenced by UsbTransmitPipeBulkIN().

**7.1405.2.5 UsbFifoMgrWrite()**

```
static blt_bool UsbFifoMgrWrite (
 blt_int8u handle,
 blt_int8u data) [static]
```

Stores data in the fifo.

**Parameters**

|               |                                                        |
|---------------|--------------------------------------------------------|
| <i>handle</i> | Identifies the fifo to write data to.                  |
| <i>data</i>   | Pointer to the data that is to be written to the fifo. |

**Returns**

BLT\_TRUE if the data was successfully stored in the fifo, BLT\_FALSE otherwise.

Referenced by `UsbReceivePipeBulkOUT()`, and `UsbTransmitByte()`.

#### 7.1405.2.6 `UsbFree()`

```
void UsbFree (
 void)
```

Releases the USB communication interface.

##### Returns

none.

#### 7.1405.2.7 `UsbInit()`

```
void UsbInit (
 void)
```

Initializes the USB communication interface.

##### Returns

none.

#### 7.1405.2.8 `UsbReceiveByte()`

```
static blt_bool UsbReceiveByte (
 blt_int8u * data) [static]
```

Receives a communication interface byte if one is present.

##### Parameters

|             |                                                 |
|-------------|-------------------------------------------------|
| <i>data</i> | Pointer to byte where the data is to be stored. |
|-------------|-------------------------------------------------|

##### Returns

BLT\_TRUE if a byte was received, BLT\_FALSE otherwise.

Referenced by `UsbReceivePacket()`.

### 7.1405.2.9 UsbReceivePacket()

```
blt_bool UsbReceivePacket (
 blt_int8u * data,
 blt_int8u * len)
```

Receives a communication interface packet if one is present.

#### Parameters

|             |                                                         |
|-------------|---------------------------------------------------------|
| <i>data</i> | Pointer to byte array where the data is to be stored.   |
| <i>len</i>  | Pointer where the length of the packet is to be stored. |

#### Returns

BLT\_TRUE if a packet was received, BLT\_FALSE otherwise.

### 7.1405.2.10 UsbReceivePipeBulkOUT()

```
void UsbReceivePipeBulkOUT (
 blt_int8u epnum)
```

Stores data that was received on the Bulk OUT pipe in the fifo.

#### Returns

none.

### 7.1405.2.11 UsbTransmitByte()

```
static blt_bool UsbTransmitByte (
 blt_int8u data) [static]
```

Transmits a communication interface byte.

#### Parameters

|             |                                          |
|-------------|------------------------------------------|
| <i>data</i> | Value of byte that is to be transmitted. |
|-------------|------------------------------------------|

#### Returns

BLT\_TRUE if the byte was transmitted, BLT\_FALSE otherwise.

Referenced by UsbTransmitPacket().



### 7.1405.2.12 UsbTransmitPacket()

```
void UsbTransmitPacket (
 blt_int8u * data,
 blt_int8u len)
```

Transmits a packet formatted for the communication interface.

#### Parameters

|             |                                                            |
|-------------|------------------------------------------------------------|
| <i>data</i> | Pointer to byte array with data that it to be transmitted. |
| <i>len</i>  | Number of bytes that are to be transmitted.                |

#### Returns

none.

### 7.1405.2.13 UsbTransmitPipeBulkIN()

```
void UsbTransmitPipeBulkIN (
 void)
```

Checks if there is still data left to transmit and if so submits it for transmission with the USB endpoint.

#### Returns

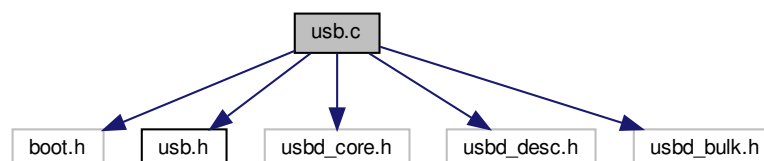
none.

## 7.1406 usb.c File Reference

Bootloader USB communication interface source file.

```
#include "boot.h"
#include "usb.h"
#include "usbd_core.h"
#include "usbd_desc.h"
#include "usbd_bulk.h"
```

Include dependency graph for ARMCM7\_STM32H7/usb.c:



## Data Structures

- struct [tFifoCtrl](#)  
*Structure type for fifo control.*
- struct [tFifoPipe](#)  
*Structure type for a fifo pipe.*

## Macros

- #define [FIFO\\_MAX\\_BUFFERS](#) (2)  
*Total number of fifo buffers.*
- #define [FIFO\\_ERR\\_INVALID\\_HANDLE](#) (255)  
*Invalid value for a fifo buffer handle.*
- #define [FIFO\\_PIPE\\_SIZE](#) (64)  
*Number of bytes that fit in the fifo pipe.*

## Functions

- static [blt\\_bool](#) [UsbReceiveByte](#) ([blt\\_int8u](#) \*data)  
*Receives a communication interface byte if one is present.*
- static [blt\\_bool](#) [UsbTransmitByte](#) ([blt\\_int8u](#) data)  
*Transmits a communication interface byte.*
- static void [UsbFifoMgrInit](#) (void)  
*Initializes the fifo manager. Each controlled fifo is assigned a unique handle, which is the same as its index into [fifoCtrl\[\]](#). Each controlled fifo holds a pointer to the next free fifo control. For the last fifo in [fifoCtrl\[\]](#) this one is set to a null-pointer as an out of fifo control indicator. Function should be called once before any of the other fifo management functions are called.*
- static [blt\\_int8u](#) [UsbFifoMgrCreate](#) ([blt\\_int8u](#) \*buffer, [blt\\_int8u](#) length)  
*Places a data storage array under fifo management control. A handle for identifying the fifo in subsequent fifo management function calls is returned, if successful.*
- static [blt\\_bool](#) [UsbFifoMgrWrite](#) ([blt\\_int8u](#) handle, [blt\\_int8u](#) data)  
*Stores data in the fifo.*
- static [blt\\_bool](#) [UsbFifoMgrRead](#) ([blt\\_int8u](#) handle, [blt\\_int8u](#) \*data)  
*Retrieves data from the fifo.*
- static [blt\\_int8u](#) [UsbFifoMgrScan](#) ([blt\\_int8u](#) handle)  
*Returns the number of data entries currently present in the fifo.*
- void [UsbInit](#) (void)  
*Initializes the USB communication interface.*
- void [UsbFree](#) (void)  
*Releases the USB communication interface.*
- void [UsbTransmitPacket](#) ([blt\\_int8u](#) \*data, [blt\\_int8u](#) len)  
*Transmits a packet formatted for the communication interface.*
- [blt\\_bool](#) [UsbReceivePacket](#) ([blt\\_int8u](#) \*data, [blt\\_int8u](#) \*len)  
*Receives a communication interface packet if one is present.*
- void [UsbTransmitPipeBulkIN](#) (void)  
*Checks if there is still data left to transmit and if so submits it for transmission with the USB endpoint.*
- void [UsbReceivePipeBulkOUT](#) ([blt\\_int8u](#) epnum)  
*Stores data that was received on the Bulk OUT pipe in the fifo.*

## Variables

- static `tFifoCtrl fifoCtrl` [FIFO\_MAX\_BUFFERS]  
*Local variable that holds the fifo control structures.*
- static `tFifoCtrl * fifoCtrlFree`  
*Local pointer that points to the next free fifo control structure.*
- static `tFifoPipe fifoPipeBulkIN`  
*Fifo pipe used for the bulk in endpoint.*
- static `tFifoPipe fifoPipeBulkOUT`  
*Fifo pipe used for the bulk out endpoint.*
- static `USBD_HandleTypeDef hUsbDeviceFS`  
*USB device handle.*

### 7.1406.1 Detailed Description

Bootloader USB communication interface source file.

### 7.1406.2 Function Documentation

#### 7.1406.2.1 UsbFifoMgrCreate()

```
static blt_int8u UsbFifoMgrCreate (
 blt_int8u * buffer,
 blt_int8u length) [static]
```

Places a data storage array under fifo management control. A handle for identifying the fifo in subsequent fifo management function calls is returned, if successful.

#### Parameters

|               |                                                                 |
|---------------|-----------------------------------------------------------------|
| <i>buffer</i> | Pointer to the first element in the data storage fifo.          |
| <i>length</i> | Maximum number of data elements that can be stored in the fifo. |

#### Returns

Fifo handle if successfull, or FIFO\_ERR\_INVALID\_HANDLE.

Referenced by `UsbInit()`.

#### 7.1406.2.2 UsbFifoMgrInit()

```
static void UsbFifoMgrInit (
 void) [static]
```

Initializes the fifo manager. Each controlled fifo is assigned a unique handle, which is the same as its index into `fifoCtrl[]`. Each controlled fifo holds a pointer to the next free fifo control. For the last fifo in `fifoCtrl[]` this one is set to a null-pointer as an out of fifo control indicator. Function should be called once before any of the other fifo management functions are called.

#### Returns

none.

Referenced by `UsbInit()`.

#### 7.1406.2.3 UsbFifoMgrRead()

```
static blt_bool UsbFifoMgrRead (
 blt_int8u handle,
 blt_int8u * data) [static]
```

Retrieves data from the fifo.

#### Parameters

|               |                                                 |
|---------------|-------------------------------------------------|
| <i>handle</i> | Identifies the fifo to read data from.          |
| <i>data</i>   | Pointer to where the read data is to be stored. |

#### Returns

BLT\_TRUE if the data was successfully read from the fifo, BLT\_FALSE otherwise.

Referenced by `UsbReceiveByte()`, and `UsbTransmitPipeBulkIN()`.

#### 7.1406.2.4 UsbFifoMgrScan()

```
static blt_int8u UsbFifoMgrScan (
 blt_int8u handle) [static]
```

Returns the number of data entries currently present in the fifo.

#### Parameters

|               |                                            |
|---------------|--------------------------------------------|
| <i>handle</i> | Identifies the fifo that is to be scanned. |
|---------------|--------------------------------------------|

#### Returns

Number of data entries in the fifo if successful, otherwise 0.

Referenced by `UsbTransmitPipeBulkIN()`.

### 7.1406.2.5 UsbFifoMgrWrite()

```
static blt_bool UsbFifoMgrWrite (
 blt_int8u handle,
 blt_int8u data) [static]
```

Stores data in the fifo.

#### Parameters

|               |                                                        |
|---------------|--------------------------------------------------------|
| <i>handle</i> | Identifies the fifo to write data to.                  |
| <i>data</i>   | Pointer to the data that is to be written to the fifo. |

#### Returns

BLT\_TRUE if the data was successfully stored in the fifo, BLT\_FALSE otherwise.

Referenced by UsbReceivePipeBulkOUT(), and UsbTransmitByte().

### 7.1406.2.6 UsbFree()

```
void UsbFree (
 void)
```

Releases the USB communication interface.

#### Returns

none.

### 7.1406.2.7 UsbInit()

```
void UsbInit (
 void)
```

Initializes the USB communication interface.

#### Returns

none.

### 7.1406.2.8 UsbReceiveByte()

```
static blt_bool UsbReceiveByte (
 blt_int8u * data) [static]
```

Receives a communication interface byte if one is present.

**Parameters**

|             |                                                 |
|-------------|-------------------------------------------------|
| <i>data</i> | Pointer to byte where the data is to be stored. |
|-------------|-------------------------------------------------|

**Returns**

BLT\_TRUE if a byte was received, BLT\_FALSE otherwise.

Referenced by UsbReceivePacket().

**7.1406.2.9 UsbReceivePacket()**

```
blt_bool UsbReceivePacket (
 blt_int8u * data,
 blt_int8u * len)
```

Receives a communication interface packet if one is present.

**Parameters**

|             |                                                         |
|-------------|---------------------------------------------------------|
| <i>data</i> | Pointer to byte array where the data is to be stored.   |
| <i>len</i>  | Pointer where the length of the packet is to be stored. |

**Returns**

BLT\_TRUE if a packet was received, BLT\_FALSE otherwise.

**7.1406.2.10 UsbReceivePipeBulkOUT()**

```
void UsbReceivePipeBulkOUT (
 blt_int8u epnum)
```

Stores data that was received on the Bulk OUT pipe in the fifo.

**Returns**

none.

**7.1406.2.11 UsbTransmitByte()**

```
static blt_bool UsbTransmitByte (
 blt_int8u data) [static]
```

Transmits a communication interface byte.

**Parameters**

|             |                                          |
|-------------|------------------------------------------|
| <i>data</i> | Value of byte that is to be transmitted. |
|-------------|------------------------------------------|

**Returns**

BLT\_TRUE if the byte was transmitted, BLT\_FALSE otherwise.

Referenced by UsbTransmitPacket().

**7.1406.2.12 UsbTransmitPacket()**

```
void UsbTransmitPacket (
 blt_int8u * data,
 blt_int8u len)
```

Transmits a packet formatted for the communication interface.

**Parameters**

|             |                                                            |
|-------------|------------------------------------------------------------|
| <i>data</i> | Pointer to byte array with data that it to be transmitted. |
| <i>len</i>  | Number of bytes that are to be transmitted.                |

**Returns**

none.

**7.1406.2.13 UsbTransmitPipeBulkIN()**

```
void UsbTransmitPipeBulkIN (
 void)
```

Checks if there is still data left to transmit and if so submits it for transmission with the USB endpoint.

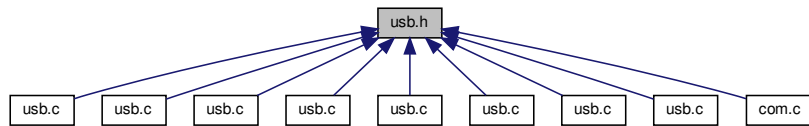
**Returns**

none.

## 7.1407 usb.h File Reference

Bootloader USB communication interface header file.

This graph shows which files directly or indirectly include this file:



### Functions

- void **UsbInit** (void)  
*Initializes the USB communication interface.*
- void **UsbFree** (void)  
*Releases the USB communication interface.*
- void **UsbTransmitPacket** (blt\_int8u \*data, blt\_int8u len)  
*Transmits a packet formatted for the communication interface.*
- blt\_bool **UsbReceivePacket** (blt\_int8u \*data, blt\_int8u \*len)  
*Receives a communication interface packet if one is present.*
- void **UsbEnterLowPowerModeHook** (void)  
*Callback that gets called whenever the USB host requests the device to enter a low power mode.*
- void **UsbLeaveLowPowerModeHook** (void)  
*Callback that gets called whenever the USB host requests the device to exit low power mode.*
- void **UsbConnectHook** (blt\_bool connect)  
*Callback that gets called whenever the USB device should be connected to the USB bus.*

### 7.1407.1 Detailed Description

Bootloader USB communication interface header file.

### 7.1407.2 Function Documentation

#### 7.1407.2.1 UsbConnectHook()

```
void UsbConnectHook (
 blt_bool connect)
```

Callback that gets called whenever the USB device should be connected to the USB bus.



**Parameters**

|                |                                                  |
|----------------|--------------------------------------------------|
| <i>connect</i> | BLT_TRUE to connect and BLT_FALSE to disconnect. |
|----------------|--------------------------------------------------|

**Returns**

none.

Referenced by `UsbFree()`, and `UsbInit()`.

**7.1407.2.2 UsbEnterLowPowerModeHook()**

```
void UsbEnterLowPowerModeHook (
 void)
```

Callback that gets called whenever the USB host requests the device to enter a low power mode.

**Returns**

none.

Referenced by `UsbBulkRxHandler()`.

**7.1407.2.3 UsbFree()**

```
void UsbFree (
 void)
```

Releases the USB communication interface.

**Returns**

none.

Referenced by `ComFree()`.

**7.1407.2.4 UsbInit()**

```
void UsbInit (
 void)
```

Initializes the USB communication interface.

**Returns**

none.

Referenced by `ComInit()`.

### 7.1407.2.5 UsbLeaveLowPowerModeHook()

```
void UsbLeaveLowPowerModeHook (
 void)
```

Callback that gets called whenever the USB host requests the device to exit low power mode.

#### Returns

none.

Referenced by UsbBulkRxHandler().

### 7.1407.2.6 UsbReceivePacket()

```
blt_bool UsbReceivePacket (
 blt_int8u * data,
 blt_int8u * len)
```

Receives a communication interface packet if one is present.

#### Parameters

|             |                                                         |
|-------------|---------------------------------------------------------|
| <i>data</i> | Pointer to byte array where the data is to be stored.   |
| <i>len</i>  | Pointer where the length of the packet is to be stored. |

#### Returns

BLT\_TRUE if a packet was received, BLT\_FALSE otherwise.

Referenced by ComTask().

### 7.1407.2.7 UsbTransmitPacket()

```
void UsbTransmitPacket (
 blt_int8u * data,
 blt_int8u len)
```

Transmits a packet formatted for the communication interface.

#### Parameters

|             |                                                            |
|-------------|------------------------------------------------------------|
| <i>data</i> | Pointer to byte array with data that it to be transmitted. |
| <i>len</i>  | Number of bytes that are to be transmitted.                |

### Returns

none.

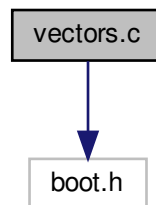
Referenced by ComTransmitPacket().

## 7.1408 vectors.c File Reference

Bootloader interrupt vector table source file.

```
#include "boot.h"
```

Include dependency graph for ARMCM3\_EFM32\_Olimex\_EM32G880F128STK\_GCC/Boot/vectors.c:



### Data Structures

- union [tlsrFunc](#)  
*Structure type for vector table entries.*

### Functions

- void [UnusedISR](#) (void)  
*Catch-all for unused interrupt service routines.*
- void [reset\\_handler](#) (void)  
*Reset interrupt service routine. Configures the stack, initializes RAM and jumps to function main.*
- [\\_\\_attribute\\_\\_](#) ((section(".isr\_vector"))) const  
*Interrupt vector table.*

#### 7.1408.1 Detailed Description

Bootloader interrupt vector table source file.

#### 7.1408.2 Function Documentation

### 7.1408.2.1 reset\_handler()

```
void reset_handler (
 void)
```

Reset interrupt service routine. Configures the stack, initializes RAM and jumps to function main.

#### Returns

none.

Reset interrupt service routine. Configures the stack, initializes RAM and jumps to function main.

#### Returns

none.

Referenced by UnusedISR().

### 7.1408.2.2 UnusedISR()

```
void UnusedISR (
 void)
```

Catch-all for unused interrupt service routines.

#### Returns

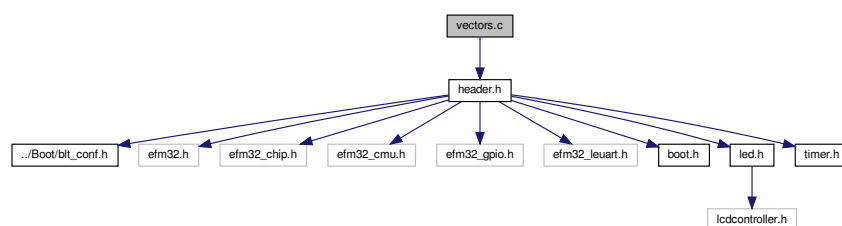
none.

## 7.1409 vectors.c File Reference

Demo program interrupt vectors source file.

```
#include "header.h"
```

Include dependency graph for ARMC3M3\_EFM32\_Olimex\_EM32G880F128STK\_GCC/Prog/vectors.c:



## Data Structures

- union [tlsrFunc](#)  
*Structure type for vector table entries.*

## Functions

- void [reset\\_handler](#) (void)  
*Reset interrupt service routine. Configures the stack, initializes RAM and jumps to function main.*
- void [UnusedISR](#) (void)  
*Catch-all for unused interrupt service routines.*
- [\\_\\_attribute\\_\\_](#) ((section(".isr\_vector"))) const  
*Interrupt vector table.*

## Variables

- unsigned long [\\_estack](#)  
*Stack end address (memory.x)*

### 7.1409.1 Detailed Description

Demo program interrupt vectors source file.

### 7.1409.2 Function Documentation

#### 7.1409.2.1 reset\_handler()

```
void reset_handler (
 void)
```

Reset interrupt service routine. Configures the stack, initializes RAM and jumps to function main.

#### Returns

none.

Reset interrupt service routine. Configures the stack, initializes RAM and jumps to function main.

#### Returns

none.

Referenced by [\\_\\_attribute\\_\\_](#) ().

### 7.1409.2.2 UnusedISR()

```
void UnusedISR (
 void)
```

Catch-all for unused interrupt service routines.

#### Returns

none.

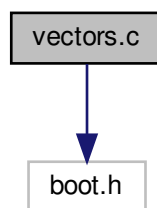
Referenced by `__attribute__()`.

## 7.1410 vectors.c File Reference

Bootloader interrupt vector table source file.

```
#include "boot.h"
```

Include dependency graph for ARMCM3\_EFM32\_Olimex\_EM32G880F128STK\_IAR/Boot/vectors.c:



## Data Structures

- union [tlsrFunc](#)  
*Structure type for vector table entries.*

## Functions

- void [reset\\_handler](#) (void)  
*Reset interrupt service routine. Configures the stack, initializes RAM and jumps to function main.*
- void [UnusedISR](#) (void)  
*Catch-all for unused interrupt service routines.*

## Variables

- `__root` const [tlsrFunc](#) `__vector_table []` [intvec](#)  
*Interrupt vector table.*

### 7.1410.1 Detailed Description

Bootloader interrupt vector table source file.

### 7.1410.2 Function Documentation

#### 7.1410.2.1 reset\_handler()

```
void reset_handler (
 void)
```

Reset interrupt service routine. Configures the stack, initializes RAM and jumps to function main.

##### Returns

none.

Reset interrupt service routine. Configures the stack, initializes RAM and jumps to function main.

##### Returns

none.

#### 7.1410.2.2 UnusedISR()

```
void UnusedISR (
 void)
```

Catch-all for unused interrupt service routines.

##### Returns

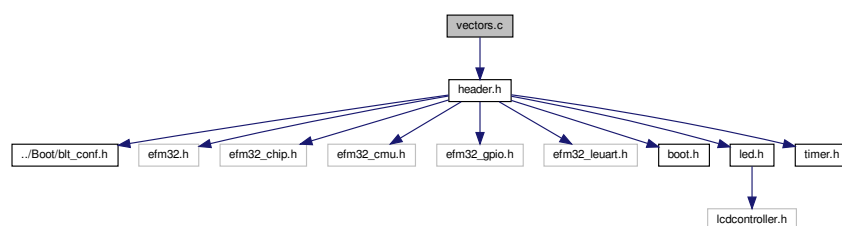
none.

## 7.1411 vectors.c File Reference

Demo program interrupt vectors source file.

```
#include "header.h"
```

Include dependency graph for ARMC32\_EFM32\_Olimex\_EM32G880F128STK\_IAR/Prog/vectors.c:



## Data Structures

- union [tlsrFunc](#)  
*Structure type for vector table entries.*

## Functions

- void [UnusedISR](#) (void)  
*Catch-all for unused interrupt service routines.*

## Variables

- `__root` const [tlsrFunc](#) `__vector_table []` [intvec](#)  
*Interrupt vector table.*

### 7.1411.1 Detailed Description

Demo program interrupt vectors source file.

### 7.1411.2 Function Documentation

#### 7.1411.2.1 UnusedISR()

```
void UnusedISR (
 void)
```

Catch-all for unused interrupt service routines.

#### Returns

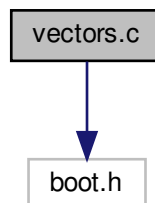
none.

## 7.1412 vectors.c File Reference

Bootloader application source file.

```
#include "boot.h"
```

Include dependency graph for ARMCM3\_LM3S\_EK\_LM3S6965\_GCC/Boot/vectors.c:





## Data Structures

- union [tlsrFunc](#)

*Structure type for vector table entries.*

## Functions

- void [UnusedISR](#) (void)  
*Catch-all for unused interrupt service routines.*
- void [reset\\_handler](#) (void)  
*Reset interrupt service routine. Configures the stack, initializes RAM and jumps to function main.*
- [\\_\\_attribute\\_\\_](#) ((section(".isr\_vector"))) const  
*Interrupt vector table.*

### 7.1412.1 Detailed Description

Bootloader application source file.

### 7.1412.2 Function Documentation

#### 7.1412.2.1 [reset\\_handler\(\)](#)

```
void reset_handler (
 void)
```

Reset interrupt service routine. Configures the stack, initializes RAM and jumps to function main.

#### Returns

none.

Reset interrupt service routine. Configures the stack, initializes RAM and jumps to function main.

#### Returns

none.

Referenced by [UnusedISR\(\)](#).



## 7.1413.2 Function Documentation

### 7.1413.2.1 reset\_handler()

```
void reset_handler (
 void)
```

Reset interrupt service routine. Configures the stack, initializes RAM and jumps to function main.

#### Returns

none.

Reset interrupt service routine. Configures the stack, initializes RAM and jumps to function main.

#### Returns

none.

Referenced by `__attribute__()`.

### 7.1413.2.2 UnusedISR()

```
void UnusedISR (
 void)
```

Catch-all for unused interrupt service routines.

#### Returns

none.

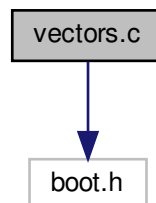
Referenced by `__attribute__()`.

## 7.1414 vectors.c File Reference

Bootloader interrupt vector table source file.

```
#include "boot.h"
```

Include dependency graph for ARMC3\_LM3S\_EK\_LM3S6965\_IAR/Boot/vectors.c:



## Data Structures

- union [tlsrFunc](#)

*Structure type for vector table entries.*

## Functions

- void [reset\\_handler](#) (void)

*Reset interrupt service routine. Configures the stack, initializes RAM and jumps to function main.*

- void [UnusedISR](#) (void)

*Catch-all for unused interrupt service routines.*

## Variables

- `__root const tlsrFunc __vector_table [ ] intvec`

*Interrupt vector table.*

### 7.1414.1 Detailed Description

Bootloader interrupt vector table source file.

### 7.1414.2 Function Documentation

#### 7.1414.2.1 [reset\\_handler\(\)](#)

```
void reset_handler (
 void)
```

Reset interrupt service routine. Configures the stack, initializes RAM and jumps to function main.

#### Returns

none.

Reset interrupt service routine. Configures the stack, initializes RAM and jumps to function main.

#### Returns

none.

#### 7.1414.2.2 UnusedISR()

```
void UnusedISR (
 void)
```

Catch-all for unused interrupt service routines.

## Returns

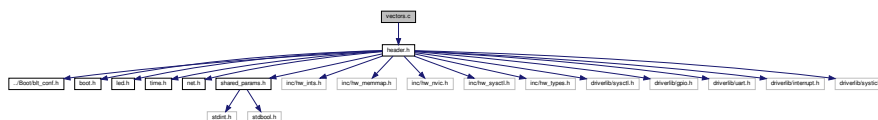
none.

## 7.1415 vectors.c File Reference

Demo program interrupt vectors source file.

```
#include "header.h"
```

```
Include dependency graph for ARMCM3_LM3S_EK_LM3S6965_IAR/Prog/vectors.c:
```



## Data Structures

- union `tlsrFunc`  
*Structure type for vector table entries.*

## Functions

- void **UnusedISR** (void)  
*Catch-all for unused interrupt service routines.*

## Variables

- `__root const tlsrFunc __vector_table [ ] intvec`  
*Interrupt vector table.*

### 7.1415.1 Detailed Description

Demo program interrupt vectors source file.

## 7.1415.2 Function Documentation

### 7.1415.2.1 UnusedISR()

```
void UnusedISR (
 void)
```

Catch-all for unused interrupt service routines.

#### Returns

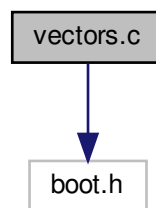
none.

## 7.1416 vectors.c File Reference

Bootloader interrupt vector table source file.

```
#include "boot.h"
```

Include dependency graph for ARMC3M3\_LM3S\_EK\_LM3S8962\_GCC/Boot/vectors.c:



## Data Structures

- union [tlsrFunc](#)  
*Structure type for vector table entries.*

## Functions

- void [UnusedISR](#) (void)  
*Catch-all for unused interrupt service routines.*
- void [reset\\_handler](#) (void)  
*Reset interrupt service routine. Configures the stack, initializes RAM and jumps to function main.*
- [\\_\\_attribute\\_\\_](#) ((section(".isr\_vector"))) const  
*Interrupt vector table.*

### 7.1416.1 Detailed Description

Bootloader interrupt vector table source file.

## 7.1416.2 Function Documentation

### 7.1416.2.1 reset\_handler()

```
void reset_handler (
 void)
```

Reset interrupt service routine. Configures the stack, initializes RAM and jumps to function main.

#### Returns

none.

Reset interrupt service routine. Configures the stack, initializes RAM and jumps to function main.

#### Returns

none.

Referenced by UnusedISR().

### 7.1416.2.2 UnusedISR()

```
void UnusedISR (
 void)
```

Catch-all for unused interrupt service routines.

#### Returns

none.

## 7.1417 vectors.c File Reference

Demo program interrupt vectors source file.

```
#include "header.h"
```

Include dependency graph for ARMCM3\_LM3S\_EK\_LM3S8962\_GCC/Prog/vectors.c:



## Data Structures

- union [tlsrFunc](#)  
*Structure type for vector table entries.*

## Functions

- void [reset\\_handler](#) (void)  
*Reset interrupt service routine. Configures the stack, initializes RAM and jumps to function main.*
- void [UnusedISR](#) (void)  
*Catch-all for unused interrupt service routines.*
- [\\_\\_attribute\\_\\_](#) ((section(".isr\_vector"))) const  
*Interrupt vector table.*

## Variables

- unsigned long [\\_estack](#)  
*Stack end address (memory.x)*

### 7.1417.1 Detailed Description

Demo program interrupt vectors source file.

### 7.1417.2 Function Documentation

#### 7.1417.2.1 reset\_handler()

```
void reset_handler (
 void)
```

Reset interrupt service routine. Configures the stack, initializes RAM and jumps to function main.

#### Returns

none.

Reset interrupt service routine. Configures the stack, initializes RAM and jumps to function main.

#### Returns

none.

Referenced by [\\_\\_attribute\\_\\_](#) ().



### 7.1417.2.2 UnusedISR()

```
void UnusedISR (
 void)
```

Catch-all for unused interrupt service routines.

#### Returns

none.

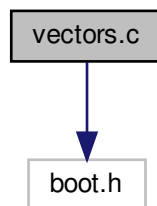
Referenced by `__attribute__()`.

## 7.1418 vectors.c File Reference

Bootloader interrupt vector table source file.

```
#include "boot.h"
```

Include dependency graph for ARMCM3\_LM3S\_EK\_LM3S8962\_IAR/Boot/vectors.c:



### Data Structures

- union [tlsrFunc](#)  
*Structure type for vector table entries.*

### Functions

- void [reset\\_handler](#) (void)  
*Reset interrupt service routine. Configures the stack, initializes RAM and jumps to function main.*
- void [UnusedISR](#) (void)  
*Catch-all for unused interrupt service routines.*

### Variables

- `__root` const [tlsrFunc](#) `__vector_table []` [intvec](#)  
*Interrupt vector table.*

### 7.1418.1 Detailed Description

Bootloader interrupt vector table source file.

### 7.1418.2 Function Documentation

#### 7.1418.2.1 reset\_handler()

```
void reset_handler (
 void)
```

Reset interrupt service routine. Configures the stack, initializes RAM and jumps to function main.

##### Returns

none.

Reset interrupt service routine. Configures the stack, initializes RAM and jumps to function main.

##### Returns

none.

#### 7.1418.2.2 UnusedISR()

```
void UnusedISR (
 void)
```

Catch-all for unused interrupt service routines.

##### Returns

none.

## 7.1419 vectors.c File Reference

Demo program interrupt vectors source file.

```
#include "header.h"
```

Include dependency graph for ARMCM3\_LM3S\_EK\_LM3S8962\_IAR/Prog/vectors.c:



## Data Structures

- union [tlsrFunc](#)  
*Structure type for vector table entries.*

## Functions

- void [UnusedISR](#) (void)  
*Catch-all for unused interrupt service routines.*

## Variables

- `__root` const [tlsrFunc](#) `__vector_table []` [intvec](#)  
*Interrupt vector table.*

### 7.1419.1 Detailed Description

Demo program interrupt vectors source file.

### 7.1419.2 Function Documentation

#### 7.1419.2.1 UnusedISR()

```
void UnusedISR (
 void)
```

Catch-all for unused interrupt service routines.

#### Returns

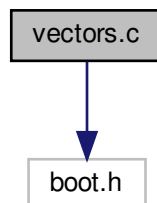
none.

## 7.1420 vectors.c File Reference

Bootloader interrupt vector table source file.

```
#include "boot.h"
```

Include dependency graph for ARMCM4\_TM4C\_DK\_TM4C123G\_IAR/Boot/vectors.c:



## Data Structures

- union [tlsrFunc](#)

*Structure type for vector table entries.*

## Functions

- void [reset\\_handler](#) (void)

*Reset interrupt service routine. Configures the stack, initializes RAM and jumps to function main.*

- void [UnusedISR](#) (void)

*Catch-all for unused interrupt service routines.*

## Variables

- `__root const tlsrFunc __vector_table [ ] intvec`

*Interrupt vector table.*

### 7.1420.1 Detailed Description

Bootloader interrupt vector table source file.

### 7.1420.2 Function Documentation

#### 7.1420.2.1 [reset\\_handler\(\)](#)

```
void reset_handler (
 void)
```

Reset interrupt service routine. Configures the stack, initializes RAM and jumps to function main.

#### Returns

none.

Reset interrupt service routine. Configures the stack, initializes RAM and jumps to function main.

#### Returns

none.

### 7.1420.2.2 UnusedISR()

```
void UnusedISR (
 void)
```

Catch-all for unused interrupt service routines.

## Returns

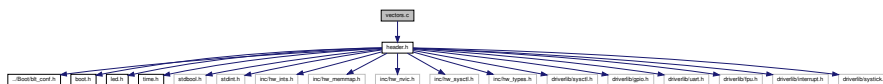
none.

## 7.1421 vectors.c File Reference

Demo program interrupt vectors source file.

```
#include "header.h"
```

Include dependency graph for ARMCM4\_TM4C\_DK\_TM4C123G\_IAR/Prog/vectors.c:



## Data Structures

- union `tlsrFunc`  
*Structure type for vector table entries.*

## Functions

- void **ResetISR** (void)  
*Interrupt service routines for the reset event.*
- void **UnusedISR** (void)  
*Catch-all for unused interrupt service routines.*

## Variables

- `__root const tlsrFunc __vector_table [] intvec`  
*Interrupt vector table.*

### 7.1421.1 Detailed Description

Demo program interrupt vectors source file.

## 7.1421.2 Function Documentation

### 7.1421.2.1 ResetISR()

```
void ResetISR (
 void)
```

Interrupt service routines for the reset event.

#### Returns

none.

### 7.1421.2.2 UnusedISR()

```
void UnusedISR (
 void)
```

Catch-all for unused interrupt service routines.

#### Returns

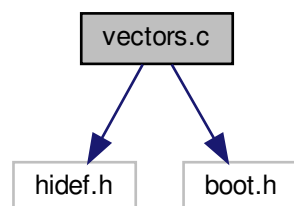
none.

## 7.1422 vectors.c File Reference

Demo program interrupt vectors source file.

```
#include <hidef.h>
#include "boot.h"
```

Include dependency graph for HCS12\_DevKit\_S12G128\_CodeWarrior/Boot/vectors.c:



## Macros

- #define [VCT\\_USER\\_PROGRAM\\_VECTOR\\_TABLE\\_STARTADDR](#) (0xE780)  
*Start address of the user program's vector table.*

## Typedefs

- typedef void(\* [near](#)) (void)  
*Type for vector table entries.*

## Functions

- void [Vector0\\_handler](#) (void)  
*ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.*
- void [Vector1\\_handler](#) (void)  
*ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.*
- void [Vector2\\_handler](#) (void)  
*ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.*
- void [Vector3\\_handler](#) (void)  
*ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.*
- void [Vector4\\_handler](#) (void)  
*ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.*
- void [Vector5\\_handler](#) (void)  
*ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.*
- void [Vector6\\_handler](#) (void)  
*ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.*
- void [Vector7\\_handler](#) (void)  
*ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.*
- void [Vector8\\_handler](#) (void)  
*ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.*
- void [Vector9\\_handler](#) (void)  
*ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.*
- void [Vector10\\_handler](#) (void)  
*ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.*
- void [Vector11\\_handler](#) (void)  
*ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.*
- void [Vector12\\_handler](#) (void)  
*ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.*
- void [Vector13\\_handler](#) (void)





- void **Vector32\_handler** (void)

- void **Vector33\_handler** (void)

- void **Vector34\_handler** (void)

- void **Vector35\_handler** (void)

- void **Vector36\_handler** (void)

- void **Vector37\_handler** (void)

- void **Vector38\_handler** (void)

- void **Vector39** handler (void)

- void **Vector40\_handler** (void)

- void **Vector41\_handler** (void)

- void **Vector42\_handler** (void)

- void **Vector43\_handler** (void)

- void **Vector44\_handler** (void)

- void **Vector45\_handler** (void)

- void **Vector46\_handler** (void)

- void **Vector47** handler (void)

- void **Vector48** handler (void)

- void **Vector49** handler (void)

*ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.*

- void [Vector50\\_handler](#) (void)

*ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.*

- void [Vector51\\_handler](#) (void)

*ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.*

- void [Vector52\\_handler](#) (void)

*ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.*

- void [Vector53\\_handler](#) (void)

*ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.*

- void [Vector54\\_handler](#) (void)

*ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.*

- void [Vector55\\_handler](#) (void)

*ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.*

- void [Vector56\\_handler](#) (void)

*ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.*

- void [Vector57\\_handler](#) (void)

*ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.*

- void [Vector58\\_handler](#) (void)

*ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.*

- void [Vector59\\_handler](#) (void)

*ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.*

- void [Vector60\\_handler](#) (void)

*ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.*

- void [Vector61\\_handler](#) (void)

*ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.*

- void [Vector62\\_handler](#) (void)

*ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.*

## Variables

- const [tIsrFunc \\_vectab](#) []

*The interrupt vector table. It contains the reset vector and all other interrupts vectors are remapped to the user program vector table.*

## 7.1422.1 Detailed Description

Demo program interrupt vectors source file.

## 7.1422.2 Macro Definition Documentation

### 7.1422.2.1 VCT\_USER\_PROGRAM\_VECTOR\_TABLE\_STARTADDR

```
#define VCT_USER_PROGRAM_VECTOR_TABLE_STARTADDR (0xE780)
```

Start address of the user program's vector table.

#### Attention

This value must be updated if the memory reserved for the bootloader changes.

Referenced by Vector0\_handler(), Vector10\_handler(), Vector11\_handler(), Vector12\_handler(), Vector13\_handler(), Vector14\_handler(), Vector15\_handler(), Vector16\_handler(), Vector17\_handler(), Vector18\_handler(), Vector19\_handler(), Vector1\_handler(), Vector20\_handler(), Vector21\_handler(), Vector22\_handler(), Vector23\_handler(), Vector24\_handler(), Vector25\_handler(), Vector26\_handler(), Vector27\_handler(), Vector28\_handler(), Vector29\_handler(), Vector2\_handler(), Vector30\_handler(), Vector31\_handler(), Vector32\_handler(), Vector33\_handler(), Vector34\_handler(), Vector35\_handler(), Vector36\_handler(), Vector37\_handler(), Vector38\_handler(), Vector39\_handler(), Vector3\_handler(), Vector40\_handler(), Vector41\_handler(), Vector42\_handler(), Vector43\_handler(), Vector44\_handler(), Vector45\_handler(), Vector46\_handler(), Vector47\_handler(), Vector48\_handler(), Vector49\_handler(), Vector4\_handler(), Vector50\_handler(), Vector51\_handler(), Vector52\_handler(), Vector53\_handler(), Vector54\_handler(), Vector55\_handler(), Vector56\_handler(), Vector57\_handler(), Vector58\_handler(), Vector59\_handler(), Vector5\_handler(), Vector60\_handler(), Vector61\_handler(), Vector62\_handler(), Vector6\_handler(), Vector7\_handler(), Vector8\_handler(), and Vector9\_handler().

## 7.1422.3 Function Documentation

### 7.1422.3.1 Vector0\_handler()

```
void Vector0_handler (
 void)
```

ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.

#### Returns

none.

#### 7.1422.3.2 Vector10\_handler()

```
void Vector10_handler (
 void)
```

ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.

##### Returns

none.

#### 7.1422.3.3 Vector11\_handler()

```
void Vector11_handler (
 void)
```

ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.

##### Returns

none.

#### 7.1422.3.4 Vector12\_handler()

```
void Vector12_handler (
 void)
```

ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.

##### Returns

none.

#### 7.1422.3.5 Vector13\_handler()

```
void Vector13_handler (
 void)
```

ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.

##### Returns

none.

### 7.1422.3.6 Vector14\_handler()

```
void Vector14_handler (
 void)
```

ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.

#### Returns

none.

### 7.1422.3.7 Vector15\_handler()

```
void Vector15_handler (
 void)
```

ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.

#### Returns

none.

### 7.1422.3.8 Vector16\_handler()

```
void Vector16_handler (
 void)
```

ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.

#### Returns

none.

### 7.1422.3.9 Vector17\_handler()

```
void Vector17_handler (
 void)
```

ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.

#### Returns

none.

#### 7.1422.3.10 Vector18\_handler()

```
void Vector18_handler (
 void)
```

ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.

##### Returns

none.

#### 7.1422.3.11 Vector19\_handler()

```
void Vector19_handler (
 void)
```

ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.

##### Returns

none.

#### 7.1422.3.12 Vector1\_handler()

```
void Vector1_handler (
 void)
```

ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.

##### Returns

none.

#### 7.1422.3.13 Vector20\_handler()

```
void Vector20_handler (
 void)
```

ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.

##### Returns

none.

**7.1422.3.14 Vector21\_handler()**

```
void Vector21_handler (
 void)
```

ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.

**Returns**

none.

**7.1422.3.15 Vector22\_handler()**

```
void Vector22_handler (
 void)
```

ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.

**Returns**

none.

**7.1422.3.16 Vector23\_handler()**

```
void Vector23_handler (
 void)
```

ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.

**Returns**

none.

**7.1422.3.17 Vector24\_handler()**

```
void Vector24_handler (
 void)
```

ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.

**Returns**

none.

**7.1422.3.18 Vector25\_handler()**

```
void Vector25_handler (
 void)
```

ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.

**Returns**

none.

**7.1422.3.19 Vector26\_handler()**

```
void Vector26_handler (
 void)
```

ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.

**Returns**

none.

**7.1422.3.20 Vector27\_handler()**

```
void Vector27_handler (
 void)
```

ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.

**Returns**

none.

**7.1422.3.21 Vector28\_handler()**

```
void Vector28_handler (
 void)
```

ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.

**Returns**

none.



**7.1422.3.22 Vector29\_handler()**

```
void Vector29_handler (
 void)
```

ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.

**Returns**

none.

**7.1422.3.23 Vector2\_handler()**

```
void Vector2_handler (
 void)
```

ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.

**Returns**

none.

**7.1422.3.24 Vector30\_handler()**

```
void Vector30_handler (
 void)
```

ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.

**Returns**

none.

**7.1422.3.25 Vector31\_handler()**

```
void Vector31_handler (
 void)
```

ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.

**Returns**

none.

**7.1422.3.26 Vector32\_handler()**

```
void Vector32_handler (
 void)
```

ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.

**Returns**

none.

**7.1422.3.27 Vector33\_handler()**

```
void Vector33_handler (
 void)
```

ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.

**Returns**

none.

**7.1422.3.28 Vector34\_handler()**

```
void Vector34_handler (
 void)
```

ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.

**Returns**

none.

**7.1422.3.29 Vector35\_handler()**

```
void Vector35_handler (
 void)
```

ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.

**Returns**

none.

**7.1422.3.30 Vector36\_handler()**

```
void Vector36_handler (
 void)
```

ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.

**Returns**

none.

**7.1422.3.31 Vector37\_handler()**

```
void Vector37_handler (
 void)
```

ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.

**Returns**

none.

**7.1422.3.32 Vector38\_handler()**

```
void Vector38_handler (
 void)
```

ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.

**Returns**

none.

**7.1422.3.33 Vector39\_handler()**

```
void Vector39_handler (
 void)
```

ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.

**Returns**

none.

**7.1422.3.34 Vector3\_handler()**

```
void Vector3_handler (
 void)
```

ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.

**Returns**

none.

**7.1422.3.35 Vector40\_handler()**

```
void Vector40_handler (
 void)
```

ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.

**Returns**

none.

**7.1422.3.36 Vector41\_handler()**

```
void Vector41_handler (
 void)
```

ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.

**Returns**

none.

**7.1422.3.37 Vector42\_handler()**

```
void Vector42_handler (
 void)
```

ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.

**Returns**

none.

**7.1422.3.38 Vector43\_handler()**

```
void Vector43_handler (
 void)
```

ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.

**Returns**

none.

**7.1422.3.39 Vector44\_handler()**

```
void Vector44_handler (
 void)
```

ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.

**Returns**

none.

**7.1422.3.40 Vector45\_handler()**

```
void Vector45_handler (
 void)
```

ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.

**Returns**

none.

**7.1422.3.41 Vector46\_handler()**

```
void Vector46_handler (
 void)
```

ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.

**Returns**

none.

**7.1422.3.42 Vector47\_handler()**

```
void Vector47_handler (
 void)
```

ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.

**Returns**

none.

**7.1422.3.43 Vector48\_handler()**

```
void Vector48_handler (
 void)
```

ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.

**Returns**

none.

**7.1422.3.44 Vector49\_handler()**

```
void Vector49_handler (
 void)
```

ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.

**Returns**

none.

**7.1422.3.45 Vector4\_handler()**

```
void Vector4_handler (
 void)
```

ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.

**Returns**

none.

**7.1422.3.46 Vector50\_handler()**

```
void Vector50_handler (
 void)
```

ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.

**Returns**

none.

**7.1422.3.47 Vector51\_handler()**

```
void Vector51_handler (
 void)
```

ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.

**Returns**

none.

**7.1422.3.48 Vector52\_handler()**

```
void Vector52_handler (
 void)
```

ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.

**Returns**

none.

**7.1422.3.49 Vector53\_handler()**

```
void Vector53_handler (
 void)
```

ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.

**Returns**

none.

**7.1422.3.50 Vector54\_handler()**

```
void Vector54_handler (
 void)
```

ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.

**Returns**

none.

**7.1422.3.51 Vector55\_handler()**

```
void Vector55_handler (
 void)
```

ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.

**Returns**

none.

**7.1422.3.52 Vector56\_handler()**

```
void Vector56_handler (
 void)
```

ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.

**Returns**

none.

**7.1422.3.53 Vector57\_handler()**

```
void Vector57_handler (
 void)
```

ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.

**Returns**

none.



**7.1422.3.54 Vector58\_handler()**

```
void Vector58_handler (
 void)
```

ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.

**Returns**

none.

**7.1422.3.55 Vector59\_handler()**

```
void Vector59_handler (
 void)
```

ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.

**Returns**

none.

**7.1422.3.56 Vector5\_handler()**

```
void Vector5_handler (
 void)
```

ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.

**Returns**

none.

**7.1422.3.57 Vector60\_handler()**

```
void Vector60_handler (
 void)
```

ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.

**Returns**

none.

**7.1422.3.58 Vector61\_handler()**

```
void Vector61_handler (
 void)
```

ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.

**Returns**

none.

**7.1422.3.59 Vector62\_handler()**

```
void Vector62_handler (
 void)
```

ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.

**Returns**

none.

**7.1422.3.60 Vector6\_handler()**

```
void Vector6_handler (
 void)
```

ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.

**Returns**

none.

**7.1422.3.61 Vector7\_handler()**

```
void Vector7_handler (
 void)
```

ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.

**Returns**

none.

### 7.1422.3.62 Vector8\_handler()

```
void Vector8_handler (
 void)
```

ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.

#### Returns

none.

### 7.1422.3.63 Vector9\_handler()

```
void Vector9_handler (
 void)
```

ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.

#### Returns

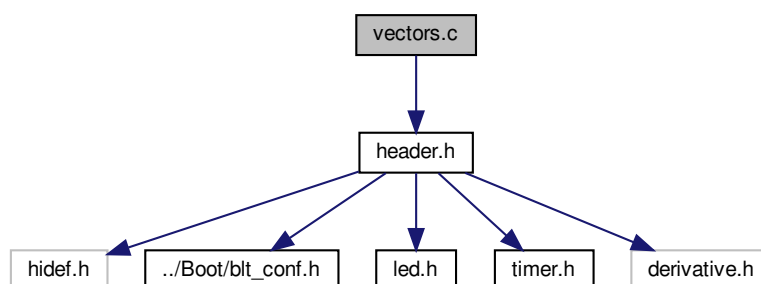
none.

## 7.1423 vectors.c File Reference

Demo program interrupt vectors source file.

```
#include "header.h"
```

Include dependency graph for HCS12\_DevKit\_S12G128\_CodeWarrior/Prog/vectors.c:



#### Typedefs

- `typedef void(* near) (void)`  
*Type for vector table entries.*

## Variables

- const `tlsrFunc _vectab []`  
*Interrupt vector table.*

### 7.1423.1 Detailed Description

Demo program interrupt vectors source file.

### 7.1423.2 Variable Documentation

#### 7.1423.2.1 \_vectab

```
const tlsrFunc _vectab[]
```

Interrupt vector table.

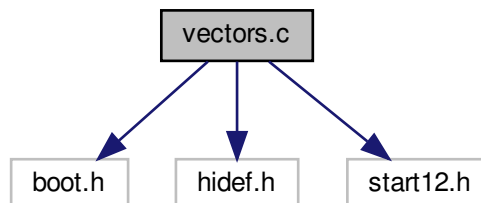
Normally, these are at 0xff80-0xffff, but the bootloader occupies 0xe800 - 0xffff. The bootloader expects the vector table to be at the end of user program flash, which is 0xe780 - 0xe7ff. 2 more bytes are reserved for the checksum that is programmed and verified by the bootloader, so the start address ends up being 0xe77e. Note that this needs to be updated when the size of the bootloader changes, as defined in the `flashLayout[]` table in `flash.c` of the bootloader.

## 7.1424 vectors.c File Reference

Bootloader interrupt vector table source file.

```
#include "boot.h"
#include "hidef.h"
#include "start12.h"
```

Include dependency graph for HCS12\_Evbplus\_Dragon12p\_CodeWarrior/Boot/vectors.c:



## Macros

- `#define VCT_USER_PROGRAM_VECTOR_TABLE_STARTADDR (0xE780)`  
*Start address of the user program's vector table.*
- `#define REG_INITRM (*(volatile blt_int8u *) 0x0010)`  
*INITRM register definition.*
- `#define REG_INITRG (*(volatile blt_int8u *) 0x0011)`  
*INITRG register definition.*
- `#define REG_INITEE (*(volatile blt_int8u *) 0x0012)`  
*INITEE register definition.*

## Typedefs

- `typedef void(* near) (void)`  
*Type for vector table entries.*

## Functions

- `void near main (void)`  
*This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.*
- `void reset_handler (void)`  
*Wrapper for calling the default reset handler which puts the communication module in a disconnected state.*
- `void Vector0_handler (void)`  
*ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.*
- `void Vector1_handler (void)`  
*ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.*
- `void Vector2_handler (void)`  
*ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.*
- `void Vector3_handler (void)`  
*ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.*
- `void Vector4_handler (void)`  
*ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.*
- `void Vector5_handler (void)`  
*ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.*
- `void Vector6_handler (void)`  
*ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.*
- `void Vector7_handler (void)`  
*ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.*
- `void Vector8_handler (void)`  
*ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.*
- `void Vector9_handler (void)`



- void **Vector28\_handler** (void)

- void **Vector29\_handler** (void)

- void **Vector30\_handler** (void)

- void **Vector31\_handler** (void)

- void **Vector32\_handler** (void)

- void **Vector33\_handler** (void)

- void **Vector34\_handler** (void)

- void **Vector35** handler (void)

- void **Vector36\_handler** (void)

- void **Vector37\_handler** (void)

- void **Vector38\_handler** (void)

- void **Vector39\_handler** (void)

- void **Vector40\_handler** (void)

- void **Vector41\_handler** (void)

- void **Vector42\_handler** (void)

- void **Vector43\_handler** (void)

- void **Vector44\_handler** (void)

- void **Vector45\_handler** (void)





## Variables

- const [tlsrFunc\\_vectab](#) []

*The interrupt vector table. It contains the reset vector and all other interrupts vectors are remapped to the user program vector table.*

### 7.1424.1 Detailed Description

Bootloader interrupt vector table source file.

### 7.1424.2 Macro Definition Documentation

#### 7.1424.2.1 VCT\_USER\_PROGRAM\_VECTOR\_TABLE\_STARTADDR

```
#define VCT_USER_PROGRAM_VECTOR_TABLE_STARTADDR (0xE780)
```

Start address of the user program's vector table.

#### Attention

This value must be updated if the memory reserved for the bootloader changes.

Referenced by [Vector0\\_handler\(\)](#), [Vector10\\_handler\(\)](#), [Vector11\\_handler\(\)](#), [Vector12\\_handler\(\)](#), [Vector13\\_↵  
handler\(\)](#), [Vector14\\_handler\(\)](#), [Vector15\\_handler\(\)](#), [Vector16\\_handler\(\)](#), [Vector17\\_handler\(\)](#), [Vector18\\_handler\(\)](#),  
[Vector19\\_handler\(\)](#), [Vector1\\_handler\(\)](#), [Vector20\\_handler\(\)](#), [Vector21\\_handler\(\)](#), [Vector22\\_handler\(\)](#), [Vector23\\_↵  
\\_handler\(\)](#), [Vector24\\_handler\(\)](#), [Vector25\\_handler\(\)](#), [Vector26\\_handler\(\)](#), [Vector27\\_handler\(\)](#), [Vector28\\_handler\(\)](#),  
[Vector29\\_handler\(\)](#), [Vector2\\_handler\(\)](#), [Vector30\\_handler\(\)](#), [Vector31\\_handler\(\)](#), [Vector32\\_handler\(\)](#), [Vector33\\_↵  
\\_handler\(\)](#), [Vector34\\_handler\(\)](#), [Vector35\\_handler\(\)](#), [Vector36\\_handler\(\)](#), [Vector37\\_handler\(\)](#), [Vector38\\_handler\(\)](#),  
[Vector39\\_handler\(\)](#), [Vector3\\_handler\(\)](#), [Vector40\\_handler\(\)](#), [Vector41\\_handler\(\)](#), [Vector42\\_handler\(\)](#), [Vector43\\_↵  
\\_handler\(\)](#), [Vector44\\_handler\(\)](#), [Vector45\\_handler\(\)](#), [Vector46\\_handler\(\)](#), [Vector47\\_handler\(\)](#), [Vector48\\_handler\(\)](#),  
[Vector49\\_handler\(\)](#), [Vector4\\_handler\(\)](#), [Vector50\\_handler\(\)](#), [Vector51\\_handler\(\)](#), [Vector52\\_handler\(\)](#), [Vector53\\_↵  
\\_handler\(\)](#), [Vector54\\_handler\(\)](#), [Vector55\\_handler\(\)](#), [Vector56\\_handler\(\)](#), [Vector57\\_handler\(\)](#), [Vector58\\_handler\(\)](#),  
[Vector59\\_handler\(\)](#), [Vector5\\_handler\(\)](#), [Vector60\\_handler\(\)](#), [Vector61\\_handler\(\)](#), [Vector62\\_handler\(\)](#), [Vector6\\_↵  
handler\(\)](#), [Vector7\\_handler\(\)](#), [Vector8\\_handler\(\)](#), and [Vector9\\_handler\(\)](#).

### 7.1424.3 Function Documentation

#### 7.1424.3.1 main()

```
void near main (
 void)
```

This is the entry point for the bootloader application and is called by the reset interrupt vector after the C-startup routines executed.

##### Returns

Program return code.  
none.  
Program exit code.  
None.

#### 7.1424.3.2 reset\_handler()

```
void reset_handler (
 void)
```

Wrapper for calling the default reset handler which puts the communication module in a disconnected state.

Reset interrupt service routine. Configures the stack, initializes RAM and jumps to function main.

##### Returns

none.

Referenced by `__attribute__()`, and `UnusedISR()`.

#### 7.1424.3.3 Vector0\_handler()

```
void Vector0_handler (
 void)
```

ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.

##### Returns

none.

#### 7.1424.3.4 Vector10\_handler()

```
void Vector10_handler (
 void)
```

ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.

##### Returns

none.

#### 7.1424.3.5 Vector11\_handler()

```
void Vector11_handler (
 void)
```

ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.

##### Returns

none.

#### 7.1424.3.6 Vector12\_handler()

```
void Vector12_handler (
 void)
```

ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.

##### Returns

none.

#### 7.1424.3.7 Vector13\_handler()

```
void Vector13_handler (
 void)
```

ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.

##### Returns

none.

#### 7.1424.3.8 Vector14\_handler()

```
void Vector14_handler (
 void)
```

ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.

##### Returns

none.

#### 7.1424.3.9 Vector15\_handler()

```
void Vector15_handler (
 void)
```

ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.

##### Returns

none.

#### 7.1424.3.10 Vector16\_handler()

```
void Vector16_handler (
 void)
```

ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.

##### Returns

none.

#### 7.1424.3.11 Vector17\_handler()

```
void Vector17_handler (
 void)
```

ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.

##### Returns

none.

**7.1424.3.12 Vector18\_handler()**

```
void Vector18_handler (
 void)
```

ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.

**Returns**

none.

**7.1424.3.13 Vector19\_handler()**

```
void Vector19_handler (
 void)
```

ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.

**Returns**

none.

**7.1424.3.14 Vector1\_handler()**

```
void Vector1_handler (
 void)
```

ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.

**Returns**

none.

**7.1424.3.15 Vector20\_handler()**

```
void Vector20_handler (
 void)
```

ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.

**Returns**

none.

**7.1424.3.16 Vector21\_handler()**

```
void Vector21_handler (
 void)
```

ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.

**Returns**

none.

**7.1424.3.17 Vector22\_handler()**

```
void Vector22_handler (
 void)
```

ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.

**Returns**

none.

**7.1424.3.18 Vector23\_handler()**

```
void Vector23_handler (
 void)
```

ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.

**Returns**

none.

**7.1424.3.19 Vector24\_handler()**

```
void Vector24_handler (
 void)
```

ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.

**Returns**

none.

**7.1424.3.20 Vector25\_handler()**

```
void Vector25_handler (
 void)
```

ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.

**Returns**

none.

**7.1424.3.21 Vector26\_handler()**

```
void Vector26_handler (
 void)
```

ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.

**Returns**

none.

**7.1424.3.22 Vector27\_handler()**

```
void Vector27_handler (
 void)
```

ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.

**Returns**

none.

**7.1424.3.23 Vector28\_handler()**

```
void Vector28_handler (
 void)
```

ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.

**Returns**

none.

**7.1424.3.24 Vector29\_handler()**

```
void Vector29_handler (
 void)
```

ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.

**Returns**

none.

**7.1424.3.25 Vector2\_handler()**

```
void Vector2_handler (
 void)
```

ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.

**Returns**

none.

**7.1424.3.26 Vector30\_handler()**

```
void Vector30_handler (
 void)
```

ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.

**Returns**

none.

**7.1424.3.27 Vector31\_handler()**

```
void Vector31_handler (
 void)
```

ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.

**Returns**

none.



**7.1424.3.28 Vector32\_handler()**

```
void Vector32_handler (
 void)
```

ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.

**Returns**

none.

**7.1424.3.29 Vector33\_handler()**

```
void Vector33_handler (
 void)
```

ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.

**Returns**

none.

**7.1424.3.30 Vector34\_handler()**

```
void Vector34_handler (
 void)
```

ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.

**Returns**

none.

**7.1424.3.31 Vector35\_handler()**

```
void Vector35_handler (
 void)
```

ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.

**Returns**

none.

**7.1424.3.32 Vector36\_handler()**

```
void Vector36_handler (
 void)
```

ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.

**Returns**

none.

**7.1424.3.33 Vector37\_handler()**

```
void Vector37_handler (
 void)
```

ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.

**Returns**

none.

**7.1424.3.34 Vector38\_handler()**

```
void Vector38_handler (
 void)
```

ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.

**Returns**

none.

**7.1424.3.35 Vector39\_handler()**

```
void Vector39_handler (
 void)
```

ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.

**Returns**

none.

**7.1424.3.36 Vector3\_handler()**

```
void Vector3_handler (
 void)
```

ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.

**Returns**

none.

**7.1424.3.37 Vector40\_handler()**

```
void Vector40_handler (
 void)
```

ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.

**Returns**

none.

**7.1424.3.38 Vector41\_handler()**

```
void Vector41_handler (
 void)
```

ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.

**Returns**

none.

**7.1424.3.39 Vector42\_handler()**

```
void Vector42_handler (
 void)
```

ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.

**Returns**

none.

**7.1424.3.40 Vector43\_handler()**

```
void Vector43_handler (
 void)
```

ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.

**Returns**

none.

**7.1424.3.41 Vector44\_handler()**

```
void Vector44_handler (
 void)
```

ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.

**Returns**

none.

**7.1424.3.42 Vector45\_handler()**

```
void Vector45_handler (
 void)
```

ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.

**Returns**

none.

**7.1424.3.43 Vector46\_handler()**

```
void Vector46_handler (
 void)
```

ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.

**Returns**

none.

**7.1424.3.44 Vector47\_handler()**

```
void Vector47_handler (
 void)
```

ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.

**Returns**

none.

**7.1424.3.45 Vector48\_handler()**

```
void Vector48_handler (
 void)
```

ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.

**Returns**

none.

**7.1424.3.46 Vector49\_handler()**

```
void Vector49_handler (
 void)
```

ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.

**Returns**

none.

**7.1424.3.47 Vector4\_handler()**

```
void Vector4_handler (
 void)
```

ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.

**Returns**

none.

**7.1424.3.48 Vector50\_handler()**

```
void Vector50_handler (
 void)
```

ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.

**Returns**

none.

**7.1424.3.49 Vector51\_handler()**

```
void Vector51_handler (
 void)
```

ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.

**Returns**

none.

**7.1424.3.50 Vector52\_handler()**

```
void Vector52_handler (
 void)
```

ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.

**Returns**

none.

**7.1424.3.51 Vector53\_handler()**

```
void Vector53_handler (
 void)
```

ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.

**Returns**

none.

**7.1424.3.52 Vector54\_handler()**

```
void Vector54_handler (
 void)
```

ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.

**Returns**

none.

**7.1424.3.53 Vector55\_handler()**

```
void Vector55_handler (
 void)
```

ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.

**Returns**

none.

**7.1424.3.54 Vector56\_handler()**

```
void Vector56_handler (
 void)
```

ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.

**Returns**

none.

**7.1424.3.55 Vector57\_handler()**

```
void Vector57_handler (
 void)
```

ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.

**Returns**

none.

**7.1424.3.56 Vector58\_handler()**

```
void Vector58_handler (
 void)
```

ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.

**Returns**

none.

**7.1424.3.57 Vector59\_handler()**

```
void Vector59_handler (
 void)
```

ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.

**Returns**

none.

**7.1424.3.58 Vector5\_handler()**

```
void Vector5_handler (
 void)
```

ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.

**Returns**

none.

**7.1424.3.59 Vector60\_handler()**

```
void Vector60_handler (
 void)
```

ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.

**Returns**

none.



**7.1424.3.60 Vector61\_handler()**

```
void Vector61_handler (
 void)
```

ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.

**Returns**

none.

**7.1424.3.61 Vector62\_handler()**

```
void Vector62_handler (
 void)
```

ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.

**Returns**

none.

**7.1424.3.62 Vector6\_handler()**

```
void Vector6_handler (
 void)
```

ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.

**Returns**

none.

**7.1424.3.63 Vector7\_handler()**

```
void Vector7_handler (
 void)
```

ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.

**Returns**

none.

### 7.1424.3.64 Vector8\_handler()

```
void Vector8_handler (
 void)
```

ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.

#### Returns

none.

### 7.1424.3.65 Vector9\_handler()

```
void Vector9_handler (
 void)
```

ISR handler for a specific vector index in the interrupt vector table for linking the actual interrupts vectors to the one in the user program's vector table.

#### Returns

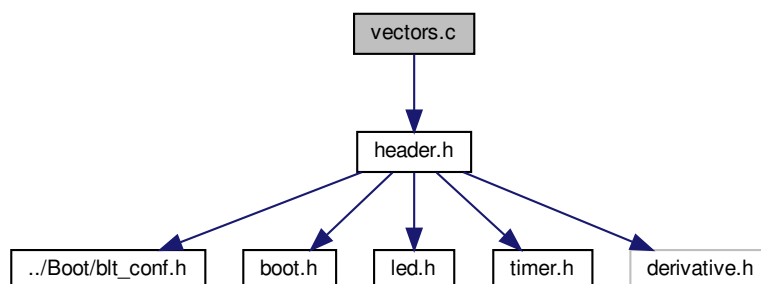
none.

## 7.1425 vectors.c File Reference

Demo program interrupt vectors source file.

```
#include "header.h"
```

Include dependency graph for HCS12\_Evbplus\_Dragon12p\_CodeWarrior/Prog/vectors.c:



## Typedefs

- typedef void(\* [near](#)) (void)  
*Type for vector table entries.*

## Variables

- const [tIsrFunc \\_vectab](#) []  
*Interrupt vector table.*

### 7.1425.1 Detailed Description

Demo program interrupt vectors source file.

### 7.1425.2 Variable Documentation

#### 7.1425.2.1 \_vectab

```
const tIsrFunc _vectab[]
```

Interrupt vector table.

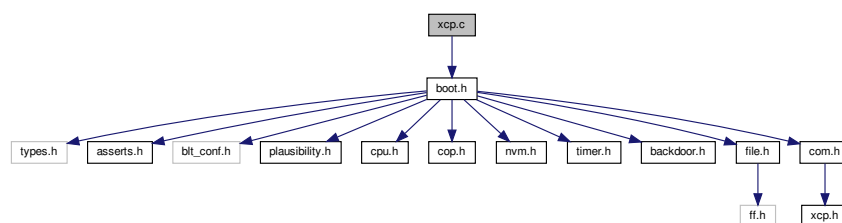
Normally, these are at 0xff80-0xffff, but the bootloader occupies 0xe800 - 0xffff. The bootloader expects the vector table to be at the end of user program flash, which is 0xe780 - 0xe7ff. 2 more bytes are reserved for the checksum that is programmed and verified by the bootloader, so the start address ends up being 0xe77e. Note that this needs to be updated when the size of the bootloader changes, as defined in the flashLayout[] table in flash.c of the bootloader.

## 7.1426 xcp.c File Reference

XCP 1.0 protocol core source file.

```
#include "boot.h"
```

Include dependency graph for xcp.c:



## Data Structures

- struct [tXcpInfo](#)  
*Structure type for grouping XCP internal module information.*

## Functions

- static void [XcpTransmitPacket](#) ([blt\\_int8u](#) \*data, [blt\\_int16s](#) len)  
*Transmits the packet using the xcp transport layer.*
- static [blt\\_int8u](#) [XcpComputeChecksum](#) ([blt\\_int32u](#) address, [blt\\_int32u](#) length, [blt\\_int32u](#) \*checksum)  
*Called by the BUILD\_CHECKSUM command to perform a checksum calculation over the specified memory region.*
- static [blt\\_int8u](#) [XcpGetSeed](#) ([blt\\_int8u](#) resource, [blt\\_int8u](#) \*seed)  
*Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the GET\_SEED command.*
- static [blt\\_int8u](#) [XcpVerifyKey](#) ([blt\\_int8u](#) resource, [blt\\_int8u](#) \*key, [blt\\_int8u](#) len)  
*Called by the UNLOCK command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.*
- static void [XcpProtectResources](#) (void)  
*Utility function to protects all the available resources.*
- static void [XcpSetCtoError](#) ([blt\\_int8u](#) error)  
*Prepares the cto packet data for the specified error.*
- static void [XcpCmdConnect](#) ([blt\\_int8u](#) \*data)  
*XCP command processor function which handles the CONNECT command as defined by the protocol.*
- static void [XcpCmdDisconnect](#) ([blt\\_int8u](#) \*data)  
*XCP command processor function which handles the DISCONNECT command as defined by the protocol.*
- static void [XcpCmdGetStatus](#) ([blt\\_int8u](#) \*data)  
*XCP command processor function which handles the GET\_STATUS command as defined by the protocol.*
- static void [XcpCmdSynch](#) ([blt\\_int8u](#) \*data)  
*XCP command processor function which handles the SYNCH command as defined by the protocol.*
- static void [XcpCmdGetId](#) ([blt\\_int8u](#) \*data)  
*XCP command processor function which handles the GET\_ID command as defined by the protocol.*
- static void [XcpCmdSetMta](#) ([blt\\_int8u](#) \*data)  
*XCP command processor function which handles the SET\_MTA command as defined by the protocol.*
- static void [XcpCmdUpload](#) ([blt\\_int8u](#) \*data)  
*XCP command processor function which handles the UPLOAD command as defined by the protocol.*
- static void [XcpCmdShortUpload](#) ([blt\\_int8u](#) \*data)  
*XCP command processor function which handles the SHORT\_UPLOAD command as defined by the protocol.*
- static void [XcpCmdBuildCheckSum](#) ([blt\\_int8u](#) \*data)  
*XCP command processor function which handles the BUILD\_CHECKSUM command as defined by the protocol.*
- static void [XcpCmdGetSeed](#) ([blt\\_int8u](#) \*data)  
*XCP command processor function which handles the GET\_SEED command as defined by the protocol.*
- static void [XcpCmdUnlock](#) ([blt\\_int8u](#) \*data)  
*XCP command processor function which handles the UNLOCK command as defined by the protocol.*
- static void [XcpCmdProgramMax](#) ([blt\\_int8u](#) \*data)  
*XCP command processor function which handles the PROGRAM\_MAX command as defined by the protocol.*
- static void [XcpCmdProgram](#) ([blt\\_int8u](#) \*data)  
*XCP command processor function which handles the PROGRAM command as defined by the protocol.*
- static void [XcpCmdProgramStart](#) ([blt\\_int8u](#) \*data)  
*XCP command processor function which handles the PROGRAM\_START command as defined by the protocol.*
- static void [XcpCmdProgramClear](#) ([blt\\_int8u](#) \*data)  
*XCP command processor function which handles the PROGRAM\_CLEAR command as defined by the protocol.*
- static void [XcpCmdProgramReset](#) ([blt\\_int8u](#) \*data)  
*XCP command processor function which handles the PROGRAM\_RESET command as defined by the protocol.*
- static void [XcpCmdProgramPrepare](#) ([blt\\_int8u](#) \*data)  
*XCP command processor function which handles the PROGRAM\_PREPARE command as defined by the protocol.*
- [blt\\_int8u](#) [XcpGetSeedHook](#) ([blt\\_int8u](#) resource, [blt\\_int8u](#) \*seed)

*Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the GET\_SEED command.*

- `blt_int8u XcpVerifyKeyHook (blt_int8u resource, blt_int8u *key, blt_int8u len)`

*Called by the UNLOCK command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.*

- `void XcpInit (void)`

*Initializes the XCP driver. Should be called once upon system startup.*

- `blt_bool XcpsConnected (void)`

*Obtains information about the XCP connection state.*

- `void XcpPacketTransmitted (void)`

*Informs the core that a pending packet transmission was completed by the transport layer.*

- `void XcpPacketReceived (blt_int8u *data, blt_int8u len)`

*Informs the core that a new packet was received by the transport layer.*

## Variables

- `static const blt_int8s xcpStationId [] = XCP_STATION_ID_STRING`

*String buffer with station id.*

- `static tXcpInfo xcpInfo`

*Local variable for storing XCP internal module info.*

## 7.1426.1 Detailed Description

XCP 1.0 protocol core source file.

## 7.1426.2 Function Documentation

### 7.1426.2.1 XcpCmdBuildChecksum()

```
static void XcpCmdBuildChecksum (
 blt_int8u * data) [static]
```

XCP command processor function which handles the BUILD\_CHECKSUM command as defined by the protocol.

#### Parameters

|                   |                                                |
|-------------------|------------------------------------------------|
| <code>data</code> | Pointer to a byte buffer with the packet data. |
|-------------------|------------------------------------------------|

#### Returns

none

Referenced by `XcpPacketReceived()`.

#### 7.1426.2.2 XcpCmdConnect()

```
static void XcpCmdConnect (
 blt_int8u * data) [static]
```

XCP command processor function which handles the CONNECT command as defined by the protocol.

##### Parameters

|             |                                                |
|-------------|------------------------------------------------|
| <i>data</i> | Pointer to a byte buffer with the packet data. |
|-------------|------------------------------------------------|

##### Returns

none

Referenced by XcpPacketReceived().

#### 7.1426.2.3 XcpCmdDisconnect()

```
static void XcpCmdDisconnect (
 blt_int8u * data) [static]
```

XCP command processor function which handles the DISCONNECT command as defined by the protocol.

##### Parameters

|             |                                                |
|-------------|------------------------------------------------|
| <i>data</i> | Pointer to a byte buffer with the packet data. |
|-------------|------------------------------------------------|

##### Returns

none

Referenced by XcpPacketReceived().

#### 7.1426.2.4 XcpCmdGetId()

```
static void XcpCmdGetId (
 blt_int8u * data) [static]
```

XCP command processor function which handles the GET\_ID command as defined by the protocol.

##### Parameters

|             |                                                |
|-------------|------------------------------------------------|
| <i>data</i> | Pointer to a byte buffer with the packet data. |
|-------------|------------------------------------------------|

**Returns**

none

Referenced by XcpPacketReceived().

**7.1426.2.5 XcpCmdGetSeed()**

```
static void XcpCmdGetSeed (
 blt_int8u * data) [static]
```

XCP command processor function which handles the GET\_SEED command as defined by the protocol.

**Parameters**

|             |                                                |
|-------------|------------------------------------------------|
| <i>data</i> | Pointer to a byte buffer with the packet data. |
|-------------|------------------------------------------------|

**Returns**

none

Referenced by XcpPacketReceived().

**7.1426.2.6 XcpCmdGetStatus()**

```
static void XcpCmdGetStatus (
 blt_int8u * data) [static]
```

XCP command processor function which handles the GET\_STATUS command as defined by the protocol.

**Parameters**

|             |                                                |
|-------------|------------------------------------------------|
| <i>data</i> | Pointer to a byte buffer with the packet data. |
|-------------|------------------------------------------------|

**Returns**

none

Referenced by XcpPacketReceived().

**7.1426.2.7 XcpCmdProgram()**

```
static void XcpCmdProgram (
 blt_int8u * data) [static]
```

XCP command processor function which handles the PROGRAM command as defined by the protocol.

**Parameters**

|             |                                                |
|-------------|------------------------------------------------|
| <i>data</i> | Pointer to a byte buffer with the packet data. |
|-------------|------------------------------------------------|

**Returns**

none

Referenced by XcpPacketReceived().

**7.1426.2.8 XcpCmdProgramClear()**

```
static void XcpCmdProgramClear (
 blt_int8u * data) [static]
```

XCP command processor function which handles the PROGRAM\_CLEAR command as defined by the protocol.

**Parameters**

|             |                                                |
|-------------|------------------------------------------------|
| <i>data</i> | Pointer to a byte buffer with the packet data. |
|-------------|------------------------------------------------|

**Returns**

none

Referenced by XcpPacketReceived().

**7.1426.2.9 XcpCmdProgramMax()**

```
static void XcpCmdProgramMax (
 blt_int8u * data) [static]
```

XCP command processor function which handles the PROGRAM\_MAX command as defined by the protocol.

**Parameters**

|             |                                                |
|-------------|------------------------------------------------|
| <i>data</i> | Pointer to a byte buffer with the packet data. |
|-------------|------------------------------------------------|

**Returns**

none

Referenced by XcpPacketReceived().



### 7.1426.2.10 XcpCmdProgramPrepare()

```
static void XcpCmdProgramPrepare (
 blt_int8u * data) [static]
```

XCP command processor function which handles the PROGRAM\_PREPARE command as defined by the protocol.

#### Parameters

|             |                                                |
|-------------|------------------------------------------------|
| <i>data</i> | Pointer to a byte buffer with the packet data. |
|-------------|------------------------------------------------|

#### Returns

none

Referenced by XcpPacketReceived().

### 7.1426.2.11 XcpCmdProgramReset()

```
static void XcpCmdProgramReset (
 blt_int8u * data) [static]
```

XCP command processor function which handles the PROGRAM\_RESET command as defined by the protocol.

#### Parameters

|             |                                                |
|-------------|------------------------------------------------|
| <i>data</i> | Pointer to a byte buffer with the packet data. |
|-------------|------------------------------------------------|

#### Returns

none

Referenced by XcpPacketReceived().

### 7.1426.2.12 XcpCmdProgramStart()

```
static void XcpCmdProgramStart (
 blt_int8u * data) [static]
```

XCP command processor function which handles the PROGRAM\_START command as defined by the protocol.

#### Parameters

|             |                                                |
|-------------|------------------------------------------------|
| <i>data</i> | Pointer to a byte buffer with the packet data. |
|-------------|------------------------------------------------|

**Returns**

none

Referenced by XcpPacketReceived().

**7.1426.2.13 XcpCmdSetMta()**

```
static void XcpCmdSetMta (
 blt_int8u * data) [static]
```

XCP command processor function which handles the SET\_MTA command as defined by the protocol.

**Parameters**

|             |                                                |
|-------------|------------------------------------------------|
| <i>data</i> | Pointer to a byte buffer with the packet data. |
|-------------|------------------------------------------------|

**Returns**

none

Referenced by XcpPacketReceived().

**7.1426.2.14 XcpCmdShortUpload()**

```
static void XcpCmdShortUpload (
 blt_int8u * data) [static]
```

XCP command processor function which handles the SHORT\_UPLOAD command as defined by the protocol.

**Parameters**

|             |                                                |
|-------------|------------------------------------------------|
| <i>data</i> | Pointer to a byte buffer with the packet data. |
|-------------|------------------------------------------------|

**Returns**

none

Referenced by XcpPacketReceived().

**7.1426.2.15 XcpCmdSynch()**

```
static void XcpCmdSynch (
 blt_int8u * data) [static]
```

XCP command processor function which handles the SYNCH command as defined by the protocol.

**Parameters**

|             |                                                |
|-------------|------------------------------------------------|
| <i>data</i> | Pointer to a byte buffer with the packet data. |
|-------------|------------------------------------------------|

**Returns**

none

Referenced by XcpPacketReceived().

**7.1426.2.16 XcpCmdUnlock()**

```
static void XcpCmdUnlock (
 blt_int8u * data) [static]
```

XCP command processor function which handles the UNLOCK command as defined by the protocol.

**Parameters**

|             |                                                |
|-------------|------------------------------------------------|
| <i>data</i> | Pointer to a byte buffer with the packet data. |
|-------------|------------------------------------------------|

**Returns**

none

Referenced by XcpPacketReceived().

**7.1426.2.17 XcpCmdUpload()**

```
static void XcpCmdUpload (
 blt_int8u * data) [static]
```

XCP command processor function which handles the UPLOAD command as defined by the protocol.

**Parameters**

|             |                                                |
|-------------|------------------------------------------------|
| <i>data</i> | Pointer to a byte buffer with the packet data. |
|-------------|------------------------------------------------|

**Returns**

none

Referenced by XcpPacketReceived().

### 7.1426.2.18 XcpComputeChecksum()

```
static blt_int8u XcpComputeChecksum (
 blt_int32u address,
 blt_int32u length,
 blt_int32u * checksum) [static]
```

Called by the BUILD\_CHECKSUM command to perform a checksum calculation over the specified memory region.

#### Parameters

|                 |                                                           |
|-----------------|-----------------------------------------------------------|
| <i>address</i>  | The start address of the memory region.                   |
| <i>length</i>   | Length of the memory region in bytes.                     |
| <i>checksum</i> | Pointer to where the calculated checksum is to be stored. |

#### Returns

Checksum type that was used during the checksum calculation.

Referenced by XcpCmdBuildCheckSum().

### 7.1426.2.19 XcpGetSeed()

```
static blt_int8u XcpGetSeed (
 blt_int8u resource,
 blt_int8u * seed) [static]
```

Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the GET\_SEED command.

#### Parameters

|                 |                                                        |
|-----------------|--------------------------------------------------------|
| <i>resource</i> | Resource that the seed is requested for (XCP_RES_XXX). |
| <i>seed</i>     | Pointer to byte buffer where the seed will be stored.  |

#### Returns

Length of the seed in bytes.

Referenced by XcpCmdGetSeed().

### 7.1426.2.20 XcpGetSeedHook()

```
blt_int8u XcpGetSeedHook (
 blt_int8u resource,
 blt_int8u * seed)
```

Provides a seed to the XCP master that will be used for the key generation when the master attempts to unlock the specified resource. Called by the GET\_SEED command.

**Parameters**

|                 |                                                        |
|-----------------|--------------------------------------------------------|
| <i>resource</i> | Resource that the seed is requested for (XCP_RES_XXX). |
| <i>seed</i>     | Pointer to byte buffer where the seed will be stored.  |

**Returns**

Length of the seed in bytes.

Referenced by XcpGetSeed().

**7.1426.2.21 XcpInit()**

```
void XcpInit (
 void)
```

Initializes the XCP driver. Should be called once upon system startup.

**Returns**

none

Referenced by ComInit().

**7.1426.2.22 XcpIsConnected()**

```
blt_bool XcpIsConnected (
 void)
```

Obtains information about the XCP connection state.

**Returns**

BLT\_TRUE is an XCP connection is established, BLT\_FALSE otherwise.

Referenced by ComIsConnected().

**7.1426.2.23 XcpPacketReceived()**

```
void XcpPacketReceived (
 blt_int8u * data,
 blt_int8u len)
```

Informs the core that a new packet was received by the transport layer.

**Parameters**

|             |                                          |
|-------------|------------------------------------------|
| <i>data</i> | Pointer to byte buffer with packet data. |
| <i>len</i>  | Number of bytes in the packet.           |

**Returns**

none

Referenced by ComTask(), and NetApp().

**7.1426.2.24 XcpPacketTransmitted()**

```
void XcpPacketTransmitted (
 void)
```

Informs the core that a pending packet transmission was completed by the transport layer.

**Returns**

none

Referenced by ComTransmitPacket().

**7.1426.2.25 XcpProtectResources()**

```
static void XcpProtectResources (
 void) [static]
```

Utility function to protects all the available resources.

**Returns**

none

Referenced by XcpCmdConnect(), XcpCmdDisconnect(), and XcpCmdUnlock().

**7.1426.2.26 XcpSetCtoError()**

```
static void XcpSetCtoError (
 blt_int8u error) [static]
```

Prepares the cto packet data for the specified error.

## Parameters

|              |                               |
|--------------|-------------------------------|
| <i>error</i> | XCP error code (XCP_ERR_XXX). |
|--------------|-------------------------------|

## Returns

none

Referenced by XcpCmdConnect(), XcpCmdGetSeed(), XcpCmdProgram(), XcpCmdProgramClear(), XcpCmdProgramMax(), XcpCmdProgramPrepare(), XcpCmdProgramReset(), XcpCmdProgramStart(), XcpCmdShortUpload(), XcpCmdSynch(), XcpCmdUnlock(), XcpCmdUpload(), and XcpPacketReceived().

## 7.1426.2.27 XcpTransmitPacket()

```
static void XcpTransmitPacket (
 blt_int8u * data,
 blt_int16s len) [static]
```

Transmits the packet using the xcp transport layer.

## Parameters

|             |                                                   |
|-------------|---------------------------------------------------|
| <i>data</i> | Pointer to the byte buffer with packet data.      |
| <i>len</i>  | Number of data bytes that need to be transmitted. |

## Returns

none

Referenced by XcpPacketReceived().

## 7.1426.2.28 XcpVerifyKey()

```
static blt_int8u XcpVerifyKey (
 blt_int8u resource,
 blt_int8u * key,
 blt_int8u len) [static]
```

Called by the UNLOCK command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

## Parameters

|                 |                                             |
|-----------------|---------------------------------------------|
| <i>resource</i> | resource to unlock (XCP_RES_XXX).           |
| <i>key</i>      | pointer to the byte buffer holding the key. |
| <i>len</i>      | length of the key in bytes.                 |

**Returns**

1 if the key was correct, 0 otherwise.

Referenced by XcpCmdUnlock().

**7.1426.2.29 XcpVerifyKeyHook()**

```
blt_int8u XcpVerifyKeyHook (
 blt_int8u resource,
 blt_int8u * key,
 blt_int8u len)
```

Called by the UNLOCK command and checks if the key to unlock the specified resource was correct. If so, then the resource protection will be removed.

**Parameters**

|                 |                                             |
|-----------------|---------------------------------------------|
| <i>resource</i> | resource to unlock (XCP_RES_XXX).           |
| <i>key</i>      | pointer to the byte buffer holding the key. |
| <i>len</i>      | length of the key in bytes.                 |

**Returns**

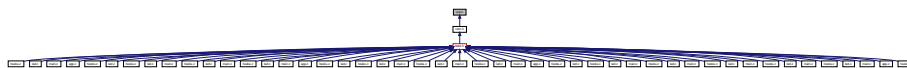
1 if the key was correct, 0 otherwise.

Referenced by XcpVerifyKey().

**7.1427 xcp.h File Reference**

XCP 1.0 protocol core header file.

This graph shows which files directly or indirectly include this file:

**Macros**

- `#define XCP_CTO_PACKET_LEN (ComGetActiveInterfaceMaxRxLen())`  
*Maximum length of the transport layer's command transmit object packet.*
- `#define XCP_DTO_PACKET_LEN (ComGetActiveInterfaceMaxTxLen())`  
*Maximum length of the transport layer's data transmit object packet.*
- `#define XCP_STATION_ID_STRING "OpenBLT"`  
*Name in string format that is used to identify the ECU to the XCP master using the GET\_ID command.*



- #define [XCP\\_MOTOROLA\\_FORMAT](#) (0x00)  
*XCP byte ordering according to the Intel (little-endian).*
- #define [XCP\\_RES\\_CALIBRATION\\_EN](#) (0)  
*Enable (=1) or disable (=0) support for the calibration resource. This is required when data is written to RAM during the XCP session.*
- #define [XCP\\_RES\\_PAGING\\_EN](#) (0)  
*Enable (=1) or disable (=0) support for the paging resource. This is required when switching between application specific calibration pages should be supported. In this case the application specific external functions AppCalSet↔Page and AppCalGetPage must be provided.*
- #define [XCP\\_RES\\_PROGRAMMING\\_EN](#) (1)  
*Enable (=1) or disable (=0) support for the programming resource. This is required when non-volatile memory will be erased or programmed during an XCP session. In this case the following external hardware specific functions must be provided: NvmWrite, NvmErase and CpuStartUserProgram.*
- #define [XCP\\_RES\\_DATA\\_ACQUISITION\\_EN](#) (0)  
*Enable (=1) or disable (=0) support for the data acquisition resource. This note that this feature is currently not supported by the XCP driver.*
- #define [XCP\\_RES\\_DATA\\_STIMULATION\\_EN](#) (0)  
*Enable (=1) or disable (=0) support for the data stimulation resource. This note that this feature is currently not supported by the XCP driver.*
- #define [XCP\\_SEED\\_KEY\\_PROTECTION\\_EN](#) (1)  
*Enable (=1) or disable (=0) support for the seed/key protection feature. If enabled, the XCP master has to perform a GET\_SEED/UNLOCK sequence to obtain access to a resource. The protection algorithm is implemented in Xcp↔GetSeed and XcpVerifyKey.*
- #define [XCP\\_UPLOAD\\_EN](#) (0)  
*Enable (=1) or disable (=0) uploading. By default, XCP always allows memory read operations using the commands UPLOAD and SHORT\_UPLOAD. This is not always desired for security reasons. If disabled, memory reads via XCP always return zero values.*
- #define [XCP\\_PACKET\\_RECEIVED\\_HOOK\\_EN](#) (0)  
*Enable (=1) or disable the hook function that gets called each time an XCP packet was received from the host.*
- #define [XCP\\_VERSION\\_PROTOCOL\\_LAYER](#) (0x0100)  
*XCP protocol layer version number (16-bit).*
- #define [XCP\\_VERSION\\_TRANSPORT\\_LAYER](#) (0x0100)  
*XCP transport layer version number (16-bit).*
- #define [XCP\\_PID\\_RES](#) (0xff)  
*Command response packet identifier.*
- #define [XCP\\_PID\\_ERR](#) (0xfe)  
*Error packet identifier.*
- #define [XCP\\_ERR\\_CMD\\_SYNCH](#) (0x00)  
*Cmd processor synchronization error code.*
- #define [XCP\\_ERR\\_CMD\\_BUSY](#) (0x10)  
*Command was not executed error code.*
- #define [XCP\\_ERR\\_CMD\\_UNKNOWN](#) (0x20)  
*Unknown or unsupported command error code.*
- #define [XCP\\_ERR\\_OUT\\_OF\\_RANGE](#) (0x22)  
*Parameter out of range error code.*
- #define [XCP\\_ERR\\_ACCESS\\_LOCKED](#) (0x25)  
*Protected error code. Seed/key required.*
- #define [XCP\\_ERR\\_PAGE\\_NOT\\_VALID](#) (0x26)  
*Cal page not valid error code.*
- #define [XCP\\_ERR\\_SEQUENCE](#) (0x29)  
*Sequence error code.*
- #define [XCP\\_ERR\\_GENERIC](#) (0x31)  
*Generic error code.*

- #define [XCP\\_CMD\\_CONNECT](#) (0xff)  
*CONNECT command code.*
- #define [XCP\\_CMD\\_DISCONNECT](#) (0xfe)  
*DISCONNECT command code.*
- #define [XCP\\_CMD\\_GET\\_STATUS](#) (0xfd)  
*GET\_STATUS command code.*
- #define [XCP\\_CMD\\_SYNCH](#) (0xfc)  
*SYNCH command code.*
- #define [XCP\\_CMD\\_GET\\_ID](#) (0xfa)  
*GET\_ID command code.*
- #define [XCP\\_CMD\\_GET\\_SEED](#) (0xf8)  
*GET\_SEED command code.*
- #define [XCP\\_CMD\\_UNLOCK](#) (0xf7)  
*UNLOCK command code.*
- #define [XCP\\_CMD\\_SET\\_MTA](#) (0xf6)  
*SET\_MTA command code.*
- #define [XCP\\_CMD\\_UPLOAD](#) (0xf5)  
*UPLOAD command code.*
- #define [XCP\\_CMD\\_SHORT\\_UPLOAD](#) (0xf4)  
*SHORT\_UPLOAD command code.*
- #define [XCP\\_CMD\\_BUILD\\_CHECKSUM](#) (0xf3)  
*BUILD\_CHECKSUM command code.*
- #define [XCP\\_CMD\\_DOWNLOAD](#) (0xf0)  
*DOWNLOAD command code.*
- #define [XCP\\_CMD\\_DOWNLOAD\\_MAX](#) (0xee)  
*DOWNLOAD\_MAX command code.*
- #define [XCP\\_CMD\\_SET\\_CAL\\_PAGE](#) (0xeb)  
*SET\_CALPAGE command code.*
- #define [XCP\\_CMD\\_GET\\_CAL\\_PAGE](#) (0xea)  
*GET\_CALPAGE command code.*
- #define [XCP\\_CMD\\_PROGRAM\\_START](#) (0xd2)  
*PROGRAM\_START command code.*
- #define [XCP\\_CMD\\_PROGRAM\\_CLEAR](#) (0xd1)  
*PROGRAM\_CLEAR command code.*
- #define [XCP\\_CMD\\_PROGRAM](#) (0xd0)  
*PROGRAM command code.*
- #define [XCP\\_CMD\\_PROGRAM\\_RESET](#) (0xcf)  
*PROGRAM\_RESET command code.*
- #define [XCP\\_CMD\\_PROGRAM\\_PREPARE](#) (0xcc)  
*PROGRAM\_PREPARE command code.*
- #define [XCP\\_CMD\\_PROGRAM\\_MAX](#) (0xc9)  
*PROGRAM\_MAX command code.*
- #define [XCP\\_RES\\_PGM](#) (0x10)  
*ProGraMming resource.*
- #define [XCP\\_RES\\_STIM](#) (0x08)  
*data STIMulation resource.*
- #define [XCP\\_RES\\_DAQ](#) (0x04)  
*Data AcQuisition resource.*
- #define [XCP\\_RES\\_CALPAG](#) (0x01)  
*CALibration and PAGing resource.*
- #define [XCP\\_CS\\_ADD11](#) (0x01)

- Add byte into byte checksum.*

  - `#define XCP_CS_ADD12 (0x02)`
- Add byte into word checksum.*

  - `#define XCP_CS_ADD14 (0x03)`
- Add byte into dword checksum.*

  - `#define XCP_CS_ADD22 (0x04)`
- Add word into word checksum.*

  - `#define XCP_CS_ADD24 (0x05)`
- Add word into dword checksum.*

  - `#define XCP_CS_ADD44 (0x06)`
- Add dword into dword checksum.*

  - `#define XCP_CS_CRC16 (0x07)`
- Use 16-bit CRC algorithm.*

  - `#define XCP_CS_CRC16CITT (0x08)`
- Use 16-bit CRC CITT algorithm.*

  - `#define XCP_CS_CRC32 (0x09)`
- Use 32-bit CRC algorithm.*

  - `#define XCP_CS_USER (0xff)`
- Use user defined algorithm.*

  - `#define XCP_SEED_MAX_LEN (BOOT_XCP_SEED_MAX_LEN)`
- Maximum number of bytes of a seed for the seed/key security feature.*

  - `#define XCP_KEY_MAX_LEN (BOOT_XCP_KEY_MAX_LEN)`
- Maximum number of bytes of a key for the seed/key security feature.*

## Functions

- void `XcpInit` (void)

*Initializes the XCP driver. Should be called once upon system startup.*
- `blt_bool XcplsConnected` (void)

*Obtains information about the XCP connection state.*
- void `XcpPacketTransmitted` (void)

*Informs the core that a pending packet transmission was completed by the transport layer.*
- void `XcpPacketReceived` (`blt_int8u *data`, `blt_int8u len`)

*Informs the core that a new packet was received by the transport layer.*

### 7.1427.1 Detailed Description

XCP 1.0 protocol core header file.

### 7.1427.2 Macro Definition Documentation

### 7.1427.2.1 XCP\_PACKET\_RECEIVED\_HOOK\_EN

```
#define XCP_PACKET_RECEIVED_HOOK_EN (0)
```

Enable (=1) or disable the hook function that gets called each time an XCP packet was received from the host.

A master-slave bootloader can be realized by using this hook-function. The mode parameter in the XCP Connect command can be interpreted as a node ID. When trying to connect to a slave, a gateway could be activated that passes the packet on to the slave. When the response packet is received from the slave, [ComTransmitPacket\(\)](#) can be called to pass the response on to the host. At the end of a firmware update procedure, the XCP Program Reset command is called, which can be used to deactivate the gateway. If this hook-function returns BLT\_TRUE, the packet is no longer processed by the XCP module. If it returns BLT\_FALSE, then the packet is processed as usual by the XCP module.

## 7.1427.3 Function Documentation

### 7.1427.3.1 XcpInit()

```
void XcpInit (
 void)
```

Initializes the XCP driver. Should be called once upon system startup.

#### Returns

none

Referenced by ComInit().

### 7.1427.3.2 XcpIsConnected()

```
blt_bool XcpIsConnected (
 void)
```

Obtains information about the XCP connection state.

#### Returns

BLT\_TRUE is an XCP connection is established, BLT\_FALSE otherwise.

Referenced by ComIsConnected().

### 7.1427.3.3 XcpPacketReceived()

```
void XcpPacketReceived (
 blt_int8u * data,
 blt_int8u len)
```

Informs the core that a new packet was received by the transport layer.

**Parameters**

|             |                                          |
|-------------|------------------------------------------|
| <i>data</i> | Pointer to byte buffer with packet data. |
| <i>len</i>  | Number of bytes in the packet.           |

**Returns**

none

Referenced by ComTask(), and NetApp().

**7.1427.3.4 XcpPacketTransmitted()**

```
void XcpPacketTransmitted (
 void)
```

Informs the core that a pending packet transmission was completed by the transport layer.

**Returns**

none

Referenced by ComTransmitPacket().



# Index

## \_\_attribute\_\_

ARMCM3\_LM3S\_EK\_LM3S6965\_GCC/Boot/shared↔\_template/Boot/led.h  
\_params.c, [3068](#)

ARMCM3\_LM3S\_EK\_LM3S6965\_GCC/Prog/shared↔  
\_params.c, [3072](#)

ARMCM4\_STM32F4\_Nucleo\_F429ZI\_CubeID↔  
E/Boot/App/shared\_params.c, [3085](#)

ARMCM4\_STM32F4\_Nucleo\_F429ZI\_CubeID↔  
E/Prog/App/shared\_params.c, [3089](#)

ARMCM4\_STM32F4\_Nucleo\_F429ZI\_GCC/↔  
Boot/shared\_params.c, [3093](#)

ARMCM4\_STM32F4\_Nucleo\_F429ZI\_GCC/↔  
Prog/shared\_params.c, [3098](#)

ARMCM4\_STM32F4\_Nucleo\_F429ZI\_Keil/↔  
Boot/shared\_params.c, [3111](#)

ARMCM4\_STM32F4\_Nucleo\_F429ZI\_Keil/↔  
Prog/shared\_params.c, [3115](#)

ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_GCC/↔  
Boot/shared\_params.c, [3119](#)

ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_GCC/↔  
Prog/shared\_params.c, [3124](#)

ARMCM7\_STM32F7\_Nucleo\_F767ZI\_CubeID↔  
E/Boot/App/shared\_params.c, [3137](#)

ARMCM7\_STM32F7\_Nucleo\_F767ZI\_CubeID↔  
E/Prog/App/shared\_params.c, [3141](#)

ARMCM7\_STM32F7\_Nucleo\_F767ZI\_GCC/↔  
Boot/shared\_params.c, [3145](#)

ARMCM7\_STM32F7\_Nucleo\_F767ZI\_GCC/↔  
Prog/shared\_params.c, [3150](#)

ARMCM7\_STM32F7\_Nucleo\_F767ZI\_Keil/↔  
Boot/shared\_params.c, [3163](#)

ARMCM7\_STM32F7\_Nucleo\_F767ZI\_Keil/↔  
Prog/shared\_params.c, [3167](#)

## \_template/Boot/hooks.c

BackDoorEntryHook, [1621](#)

BackDoorInitHook, [1621](#)

CopInitHook, [1621](#)

CopServiceHook, [1622](#)

CpuUserProgramStartHook, [1622](#)

NvmDoneHook, [1622](#)

NvmEraseHook, [1622](#)

NvmInitHook, [1623](#)

NvmReinitHook, [1623](#)

NvmWriteHook, [1623](#)

XcpGetSeedHook, [1624](#)

XcpVerifyKeyHook, [1624](#)

## \_template/Boot/led.c

LedBlinkExit, [2137](#)

LedBlinkInit, [2137](#)

LedBlinkTask, [2137](#)

LedBlinkExit, [2372](#)

LedBlinkInit, [2372](#)

LedBlinkTask, [2373](#)

\_template/Boot/main.c

Init, [2606](#)

main, [2606](#)

\_template/GCC/cpu\_comp.c

CpuIrqDisable, [1191](#)

CpuIrqEnable, [1191](#)

\_template/Prog/led.c

LedInit, [2139](#)

LedToggle, [2139](#)

\_template/Prog/led.h

LedInit, [2374](#)

LedToggle, [2374](#)

\_template/Prog/main.c

Init, [2608](#)

main, [2608](#)

\_template/can.c

CanGetSpeedConfig, [1064](#)

CanInit, [1064](#)

CanReceivePacket, [1064](#)

canTiming, [1065](#)

CanTransmitPacket, [1065](#)

\_template/cpu.c

CpuInit, [1132](#)

CpuMemCopy, [1132](#)

CpuMemSet, [1132](#)

CpuStartUserProgram, [1133](#)

CpuUserProgramStartHook, [1133](#)

\_template/flash.c

blockInfo, [1275](#)

bootBlockInfo, [1275](#)

FlashAddToBlock, [1270](#)

FlashDone, [1270](#)

FlashErase, [1270](#)

FlashEraseSectors, [1271](#)

FlashGetSectorIdx, [1271](#)

FlashGetUserProgBaseAddress, [1272](#)

FlashInit, [1272](#)

FlashInitBlock, [1272](#)

flashLayout, [1275](#)

FlashReinit, [1273](#)

FlashSwitchBlock, [1273](#)

FlashVerifyChecksum, [1273](#)

FlashWrite, [1274](#)

FlashWriteBlock, [1274](#)

- FlashWriteChecksum, [1274](#)
- \_template/flash.h
  - FlashDone, [1433](#)
  - FlashErase, [1433](#)
  - FlashGetUserProgBaseAddress, [1434](#)
  - FlashInit, [1434](#)
  - FlashReinit, [1434](#)
  - FlashVerifyChecksum, [1435](#)
  - FlashWrite, [1435](#)
  - FlashWriteChecksum, [1435](#)
- \_template/nvm.c
  - NvmDone, [2959](#)
  - NvmDoneHook, [2960](#)
  - NvmErase, [2960](#)
  - NvmEraseHook, [2960](#)
  - NvmGetUserProgBaseAddress, [2961](#)
  - NvmInit, [2961](#)
  - NvmInitHook, [2961](#)
  - NvmReinit, [2962](#)
  - NvmReinitHook, [2962](#)
  - NvmVerifyChecksum, [2962](#)
  - NvmWrite, [2963](#)
  - NvmWriteHook, [2963](#)
- \_template/types.h
  - blt\_addr, [3530](#)
  - blt\_bool, [3530](#)
  - blt\_char, [3530](#)
  - blt\_int16s, [3531](#)
  - blt\_int16u, [3531](#)
  - blt\_int32s, [3531](#)
  - blt\_int32u, [3531](#)
  - blt\_int8s, [3531](#)
  - blt\_int8u, [3531](#)
- \_template/usb.c
  - UsbFifoMgrCreate, [3569](#)
  - UsbFifoMgrInit, [3570](#)
  - UsbFifoMgrRead, [3570](#)
  - UsbFifoMgrScan, [3570](#)
  - UsbFifoMgrWrite, [3571](#)
  - UsbFree, [3571](#)
  - UsbInit, [3571](#)
  - UsbReceiveByte, [3572](#)
  - UsbReceivePacket, [3572](#)
  - UsbReceivePipeBulkOUT, [3573](#)
  - UsbTransmitByte, [3573](#)
  - UsbTransmitPacket, [3573](#)
  - UsbTransmitPipeBulkIN, [3574](#)
- \_vectab
  - HCS12\_DevKit\_S12G128\_CodeWarrior/Prog/vectors.↔  
c, [3664](#)
  - HCS12\_Evbplus\_Dragon12p\_CodeWarrior/↔  
Prog/vectors.c, [3687](#)
- ARMCM0\_S32K11/GCC/cpu\_comp.c
  - CpulrqDisable, [1192](#)
  - CpulrqEnable, [1193](#)
- ARMCM0\_S32K11/IAR/cpu\_comp.c
  - CpulrqDisable, [1194](#)
  - CpulrqEnable, [1194](#)
- ARMCM0\_S32K11/can.c
  - CanDisabledModeEnter, [1068](#)
  - CanDisabledModeExit, [1068](#)
  - CanFreezeModeEnter, [1068](#)
  - CanFreezeModeExit, [1068](#)
  - CanGetSpeedConfig, [1069](#)
  - CanInit, [1069](#)
  - CanReceivePacket, [1069](#)
  - canTiming, [1070](#)
  - CanTransmitPacket, [1070](#)
- ARMCM0\_S32K11/cpu.c
  - CpuInit, [1135](#)
  - CpuMemCopy, [1135](#)
  - CpuMemSet, [1135](#)
  - CpuStartUserProgram, [1136](#)
  - CpuUserProgramStartHook, [1136](#)
- ARMCM0\_S32K11/flash.c
  - blockInfo, [1284](#)
  - bootBlockInfo, [1284](#)
  - FlashAddToBlock, [1278](#)
  - FlashCommandSequence, [1279](#)
  - FlashDone, [1279](#)
  - FlashErase, [1280](#)
  - FlashEraseSectors, [1280](#)
  - FlashGetSectorIdx, [1281](#)
  - FlashGetUserProgBaseAddress, [1281](#)
  - FlashInit, [1281](#)
  - FlashInitBlock, [1281](#)
  - FlashReinit, [1282](#)
  - FlashSwitchBlock, [1282](#)
  - FlashVerifyChecksum, [1282](#)
  - FlashWrite, [1283](#)
  - FlashWriteBlock, [1283](#)
  - FlashWriteChecksum, [1284](#)
- ARMCM0\_S32K11/flash.h
  - FlashDone, [1437](#)
  - FlashErase, [1437](#)
  - FlashGetUserProgBaseAddress, [1438](#)
  - FlashInit, [1438](#)
  - FlashReinit, [1438](#)
  - FlashVerifyChecksum, [1438](#)
  - FlashWrite, [1439](#)
  - FlashWriteChecksum, [1439](#)
- ARMCM0\_S32K11/nvm.c
  - NvmDone, [2965](#)
  - NvmDoneHook, [2965](#)
  - NvmErase, [2966](#)
  - NvmEraseHook, [2966](#)
  - NvmGetUserProgBaseAddress, [2967](#)
  - NvmInit, [2967](#)
  - NvmInitHook, [2967](#)
  - NvmReinit, [2967](#)
  - NvmReinitHook, [2968](#)
  - NvmVerifyChecksum, [2968](#)
  - NvmWrite, [2968](#)
  - NvmWriteHook, [2969](#)
- ARMCM0\_S32K11/types.h
  - blt\_addr, [3532](#)



- blt\_bool, [3532](#)
- blt\_char, [3532](#)
- blt\_int16s, [3532](#)
- blt\_int16u, [3533](#)
- blt\_int32s, [3533](#)
- blt\_int32u, [3533](#)
- blt\_int8s, [3533](#)
- blt\_int8u, [3533](#)
- ARMCM0\_STM32F0/GCC/cpu\_comp.c
  - CpuIrqDisable, [1195](#)
  - CpuIrqEnable, [1195](#)
- ARMCM0\_STM32F0/IAR/cpu\_comp.c
  - CpuIrqDisable, [1196](#)
  - CpuIrqEnable, [1196](#)
- ARMCM0\_STM32F0/Keil/cpu\_comp.c
  - CpuIrqDisable, [1197](#)
  - CpuIrqEnable, [1198](#)
- ARMCM0\_STM32F0/can.c
  - CanGetSpeedConfig, [1072](#)
  - CanInit, [1073](#)
  - CanReceivePacket, [1073](#)
  - canTiming, [1074](#)
  - CanTransmitPacket, [1073](#)
- ARMCM0\_STM32F0/cpu.c
  - CpuInit, [1138](#)
  - CpuMemCopy, [1138](#)
  - CpuMemSet, [1138](#)
  - CpuStartUserProgram, [1139](#)
  - CpuUserProgramStartHook, [1139](#)
- ARMCM0\_STM32F0/flash.c
  - blockInfo, [1292](#)
  - bootBlockInfo, [1292](#)
  - FlashAddToBlock, [1287](#)
  - FlashDone, [1287](#)
  - FlashErase, [1287](#)
  - FlashEraseSectors, [1288](#)
  - FlashGetSector, [1288](#)
  - FlashGetSectorBaseAddr, [1288](#)
  - FlashGetSectorSize, [1289](#)
  - FlashGetUserProgBaseAddress, [1289](#)
  - FlashInit, [1289](#)
  - FlashInitBlock, [1290](#)
  - flashLayout, [1293](#)
  - FlashReinit, [1290](#)
  - FlashSwitchBlock, [1290](#)
  - FlashVerifyChecksum, [1291](#)
  - FlashWrite, [1291](#)
  - FlashWriteBlock, [1291](#)
  - FlashWriteChecksum, [1292](#)
- ARMCM0\_STM32F0/flash.h
  - FlashDone, [1440](#)
  - FlashErase, [1441](#)
  - FlashGetUserProgBaseAddress, [1441](#)
  - FlashInit, [1442](#)
  - FlashReinit, [1442](#)
  - FlashVerifyChecksum, [1442](#)
  - FlashWrite, [1442](#)
  - FlashWriteChecksum, [1443](#)
- ARMCM0\_STM32F0/nvm.c
  - NvmDone, [2971](#)
  - NvmDoneHook, [2971](#)
  - NvmErase, [2971](#)
  - NvmEraseHook, [2972](#)
  - NvmGetUserProgBaseAddress, [2972](#)
  - NvmInit, [2972](#)
  - NvmInitHook, [2973](#)
  - NvmReinit, [2973](#)
  - NvmReinitHook, [2973](#)
  - NvmVerifyChecksum, [2973](#)
  - NvmWrite, [2974](#)
  - NvmWriteHook, [2974](#)
- ARMCM0\_STM32F0/types.h
  - blt\_addr, [3534](#)
  - blt\_bool, [3534](#)
  - blt\_char, [3534](#)
  - blt\_int16s, [3534](#)
  - blt\_int16u, [3535](#)
  - blt\_int32s, [3535](#)
  - blt\_int32u, [3535](#)
  - blt\_int8s, [3535](#)
  - blt\_int8u, [3535](#)
- ARMCM0\_STM32F0\_Discovery\_STM32F051\_Cubel↔
  - DE/Boot/App/app.c
    - AppInit, [388](#)
    - AppTask, [388](#)
- ARMCM0\_STM32F0\_Discovery\_STM32F051\_Cubel↔
  - DE/Boot/App/app.h
    - AppInit, [431](#)
    - AppTask, [432](#)
- ARMCM0\_STM32F0\_Discovery\_STM32F051\_Cubel↔
  - DE/Boot/App/flash\_layout.c
    - flashLayout, [1513](#)
- ARMCM0\_STM32F0\_Discovery\_STM32F051\_Cubel↔
  - DE/Boot/App/hooks.c
    - BackDoorEntryHook, [1626](#)
    - BackDoorInitHook, [1626](#)
    - CopInitHook, [1626](#)
    - CopServiceHook, [1627](#)
    - CpuUserProgramStartHook, [1627](#)
    - NvmDoneHook, [1627](#)
    - NvmEraseHook, [1627](#)
    - NvmInitHook, [1628](#)
    - NvmReinitHook, [1628](#)
    - NvmWriteHook, [1628](#)
    - XcpGetSeedHook, [1629](#)
    - XcpVerifyKeyHook, [1629](#)
- ARMCM0\_STM32F0\_Discovery\_STM32F051\_Cubel↔
  - DE/Boot/App/led.c
    - LedBlinkExit, [2140](#)
    - LedBlinkInit, [2140](#)
    - LedBlinkTask, [2141](#)
- ARMCM0\_STM32F0\_Discovery\_STM32F051\_Cubel↔
  - DE/Boot/App/led.h
    - LedBlinkExit, [2375](#)
    - LedBlinkInit, [2375](#)
    - LedBlinkTask, [2375](#)

- ARMCM0\_STM32F0\_Discovery\_STM32F051\_Cubel↔  
DE/Prog/App/app.c  
  ApplInit, [389](#)  
  AppTask, [389](#)
- ARMCM0\_STM32F0\_Discovery\_STM32F051\_Cubel↔  
DE/Prog/App/app.h  
  ApplInit, [433](#)  
  AppTask, [433](#)
- ARMCM0\_STM32F0\_Discovery\_STM32F051\_Cubel↔  
DE/Prog/App/led.c  
  LedInit, [2142](#)  
  LedToggle, [2142](#)
- ARMCM0\_STM32F0\_Discovery\_STM32F051\_Cubel↔  
DE/Prog/App/led.h  
  LedInit, [2376](#)  
  LedToggle, [2377](#)
- ARMCM0\_STM32F0\_Discovery\_STM32F051\_GCC↔  
Boot/flash\_layout.c  
  flashLayout, [1514](#)
- ARMCM0\_STM32F0\_Discovery\_STM32F051\_GCC↔  
Boot/hooks.c  
  BackDoorEntryHook, [1631](#)  
  BackDoorInitHook, [1631](#)  
  CopInitHook, [1631](#)  
  CopServiceHook, [1632](#)  
  CpuUserProgramStartHook, [1632](#)  
  NvmDoneHook, [1632](#)  
  NvmEraseHook, [1632](#)  
  NvmInitHook, [1633](#)  
  NvmReinitHook, [1633](#)  
  NvmWriteHook, [1633](#)  
  XcpGetSeedHook, [1634](#)  
  XcpVerifyKeyHook, [1634](#)
- ARMCM0\_STM32F0\_Discovery\_STM32F051\_GCC↔  
Boot/led.c  
  LedBlinkExit, [2143](#)  
  LedBlinkInit, [2143](#)  
  LedBlinkTask, [2144](#)
- ARMCM0\_STM32F0\_Discovery\_STM32F051\_GCC↔  
Boot/led.h  
  LedBlinkExit, [2378](#)  
  LedBlinkInit, [2378](#)  
  LedBlinkTask, [2378](#)
- ARMCM0\_STM32F0\_Discovery\_STM32F051\_GCC↔  
Boot/main.c  
  HAL\_MspDelInit, [2609](#)  
  HAL\_MsplInit, [2609](#)  
  Init, [2609](#)  
  main, [2610](#)  
  SystemClock\_Config, [2610](#)
- ARMCM0\_STM32F0\_Discovery\_STM32F051\_GCC↔  
Prog/led.c  
  LedInit, [2145](#)  
  LedToggle, [2145](#)
- ARMCM0\_STM32F0\_Discovery\_STM32F051\_GCC↔  
Prog/led.h  
  LedInit, [2379](#)  
  LedToggle, [2380](#)
- ARMCM0\_STM32F0\_Discovery\_STM32F051\_GCC↔  
Prog/main.c  
  HAL\_MspDelInit, [2611](#)  
  HAL\_MsplInit, [2612](#)  
  Init, [2612](#)  
  main, [2612](#)  
  SystemClock\_Config, [2612](#)
- ARMCM0\_STM32F0\_Discovery\_STM32F051\_IAR↔  
Boot/flash\_layout.c  
  flashLayout, [1515](#)
- ARMCM0\_STM32F0\_Discovery\_STM32F051\_IAR↔  
Boot/hooks.c  
  BackDoorEntryHook, [1636](#)  
  BackDoorInitHook, [1636](#)  
  CopInitHook, [1636](#)  
  CopServiceHook, [1637](#)  
  CpuUserProgramStartHook, [1637](#)  
  NvmDoneHook, [1637](#)  
  NvmEraseHook, [1637](#)  
  NvmInitHook, [1638](#)  
  NvmReinitHook, [1638](#)  
  NvmWriteHook, [1638](#)  
  XcpGetSeedHook, [1639](#)  
  XcpVerifyKeyHook, [1639](#)
- ARMCM0\_STM32F0\_Discovery\_STM32F051\_IAR↔  
Boot/led.c  
  LedBlinkExit, [2146](#)  
  LedBlinkInit, [2146](#)  
  LedBlinkTask, [2147](#)
- ARMCM0\_STM32F0\_Discovery\_STM32F051\_IAR↔  
Boot/led.h  
  LedBlinkExit, [2381](#)  
  LedBlinkInit, [2381](#)  
  LedBlinkTask, [2381](#)
- ARMCM0\_STM32F0\_Discovery\_STM32F051\_IAR↔  
Boot/main.c  
  HAL\_MspDelInit, [2614](#)  
  HAL\_MsplInit, [2614](#)  
  Init, [2614](#)  
  main, [2614](#)  
  SystemClock\_Config, [2615](#)
- ARMCM0\_STM32F0\_Discovery\_STM32F051\_IAR↔  
Prog/led.c  
  LedInit, [2148](#)  
  LedToggle, [2148](#)
- ARMCM0\_STM32F0\_Discovery\_STM32F051\_IAR↔  
Prog/led.h  
  LedInit, [2382](#)  
  LedToggle, [2383](#)
- ARMCM0\_STM32F0\_Discovery\_STM32F051\_IAR↔  
Prog/main.c  
  HAL\_MspDelInit, [2616](#)  
  HAL\_MsplInit, [2616](#)  
  Init, [2616](#)  
  main, [2617](#)  
  SystemClock\_Config, [2617](#)
- ARMCM0\_STM32F0\_Discovery\_STM32F051\_Keil↔  
Boot/flash\_layout.c

- flashLayout, [1516](#)
- ARMCM0\_STM32F0\_Discovery\_STM32F051\_Keil/↔
  - Boot/hooks.c
  - BackDoorEntryHook, [1641](#)
  - BackDoorInitHook, [1641](#)
  - CopInitHook, [1641](#)
  - CopServiceHook, [1642](#)
  - CpuUserProgramStartHook, [1642](#)
  - NvmDoneHook, [1642](#)
  - NvmEraseHook, [1642](#)
  - NvmInitHook, [1643](#)
  - NvmReinitHook, [1643](#)
  - NvmWriteHook, [1643](#)
  - XcpGetSeedHook, [1644](#)
  - XcpVerifyKeyHook, [1644](#)
- ARMCM0\_STM32F0\_Discovery\_STM32F051\_Keil/↔
  - Boot/led.c
  - LedBlinkExit, [2149](#)
  - LedBlinkInit, [2149](#)
  - LedBlinkTask, [2150](#)
- ARMCM0\_STM32F0\_Discovery\_STM32F051\_Keil/↔
  - Boot/led.h
  - LedBlinkExit, [2384](#)
  - LedBlinkInit, [2384](#)
  - LedBlinkTask, [2384](#)
- ARMCM0\_STM32F0\_Discovery\_STM32F051\_Keil/↔
  - Boot/main.c
  - HAL\_MspDeInit, [2618](#)
  - HAL\_MspInit, [2619](#)
  - Init, [2619](#)
  - main, [2619](#)
  - SystemClock\_Config, [2619](#)
- ARMCM0\_STM32F0\_Discovery\_STM32F051\_Keil/↔
  - Prog/led.c
  - LedInit, [2151](#)
  - LedToggle, [2151](#)
- ARMCM0\_STM32F0\_Discovery\_STM32F051\_Keil/↔
  - Prog/led.h
  - LedInit, [2385](#)
  - LedToggle, [2386](#)
- ARMCM0\_STM32F0\_Discovery\_STM32F051\_Keil/↔
  - Prog/main.c
  - HAL\_MspDeInit, [2621](#)
  - HAL\_MspInit, [2621](#)
  - Init, [2621](#)
  - main, [2621](#)
  - SystemClock\_Config, [2622](#)
- ARMCM0\_STM32F0\_Nucleo\_F091RC\_CubeIDE/↔
  - Boot/App/app.c
  - AppInit, [390](#)
  - AppTask, [390](#)
- ARMCM0\_STM32F0\_Nucleo\_F091RC\_CubeIDE/↔
  - Boot/App/app.h
  - AppInit, [434](#)
  - AppTask, [434](#)
- ARMCM0\_STM32F0\_Nucleo\_F091RC\_CubeIDE/↔
  - Boot/App/hooks.c
  - BackDoorEntryHook, [1646](#)
  - BackDoorInitHook, [1646](#)
  - CopInitHook, [1646](#)
  - CopServiceHook, [1647](#)
  - CpuUserProgramStartHook, [1647](#)
  - NvmDoneHook, [1647](#)
  - NvmEraseHook, [1647](#)
  - NvmInitHook, [1648](#)
  - NvmReinitHook, [1648](#)
  - NvmWriteHook, [1648](#)
  - XcpGetSeedHook, [1649](#)
  - XcpVerifyKeyHook, [1649](#)
- ARMCM0\_STM32F0\_Nucleo\_F091RC\_CubeIDE/↔
  - Boot/App/led.c
  - LedBlinkExit, [2152](#)
  - LedBlinkInit, [2152](#)
  - LedBlinkTask, [2153](#)
- ARMCM0\_STM32F0\_Nucleo\_F091RC\_CubeIDE/↔
  - Boot/App/led.h
  - LedBlinkExit, [2387](#)
  - LedBlinkInit, [2387](#)
  - LedBlinkTask, [2387](#)
- ARMCM0\_STM32F0\_Nucleo\_F091RC\_CubeIDE/↔
  - Prog/App/app.c
  - AppInit, [391](#)
  - AppTask, [392](#)
- ARMCM0\_STM32F0\_Nucleo\_F091RC\_CubeIDE/↔
  - Prog/App/app.h
  - AppInit, [435](#)
  - AppTask, [435](#)
- ARMCM0\_STM32F0\_Nucleo\_F091RC\_CubeIDE/↔
  - Prog/App/led.c
  - LedInit, [2154](#)
  - LedToggle, [2154](#)
- ARMCM0\_STM32F0\_Nucleo\_F091RC\_CubeIDE/↔
  - Prog/App/led.h
  - LedInit, [2388](#)
  - LedToggle, [2389](#)
- ARMCM0\_STM32F0\_Nucleo\_F091RC\_GCC/Boot/hooks.↔
  - c
  - BackDoorEntryHook, [1651](#)
  - BackDoorInitHook, [1651](#)
  - CopInitHook, [1651](#)
  - CopServiceHook, [1652](#)
  - CpuUserProgramStartHook, [1652](#)
  - NvmDoneHook, [1652](#)
  - NvmEraseHook, [1652](#)
  - NvmInitHook, [1653](#)
  - NvmReinitHook, [1653](#)
  - NvmWriteHook, [1653](#)
  - XcpGetSeedHook, [1654](#)
  - XcpVerifyKeyHook, [1654](#)
- ARMCM0\_STM32F0\_Nucleo\_F091RC\_GCC/Boot/led.↔
  - c
  - LedBlinkExit, [2155](#)
  - LedBlinkInit, [2155](#)
  - LedBlinkTask, [2156](#)
- ARMCM0\_STM32F0\_Nucleo\_F091RC\_GCC/Boot/led.↔
  - h

- LedBlinkExit, [2390](#)
- LedBlinkInit, [2390](#)
- LedBlinkTask, [2390](#)
- ARMCM0\_STM32F0\_Nucleo\_F091RC\_GCC/Boot/main.↔
  - c
  - HAL\_MspDeInit, [2623](#)
  - HAL\_MspInit, [2623](#)
  - Init, [2623](#)
  - main, [2624](#)
  - SystemClock\_Config, [2624](#)
- ARMCM0\_STM32F0\_Nucleo\_F091RC\_GCC/Prog/led.↔
  - c
  - LedInit, [2157](#)
  - LedToggle, [2157](#)
- ARMCM0\_STM32F0\_Nucleo\_F091RC\_GCC/Prog/led.↔
  - h
  - LedInit, [2391](#)
  - LedToggle, [2392](#)
- ARMCM0\_STM32F0\_Nucleo\_F091RC\_GCC/Prog/main.↔
  - c
  - HAL\_MspDeInit, [2625](#)
  - HAL\_MspInit, [2626](#)
  - Init, [2626](#)
  - main, [2626](#)
  - SystemClock\_Config, [2626](#)
- ARMCM0\_STM32F0\_Nucleo\_F091RC\_IAR/Boot/hooks.↔
  - c
  - BackDoorEntryHook, [1656](#)
  - BackDoorInitHook, [1656](#)
  - CopInitHook, [1656](#)
  - CopServiceHook, [1657](#)
  - CpuUserProgramStartHook, [1657](#)
  - NvmDoneHook, [1657](#)
  - NvmEraseHook, [1657](#)
  - NvmInitHook, [1658](#)
  - NvmReinitHook, [1658](#)
  - NvmWriteHook, [1658](#)
  - XcpGetSeedHook, [1659](#)
  - XcpVerifyKeyHook, [1659](#)
- ARMCM0\_STM32F0\_Nucleo\_F091RC\_IAR/Boot/led.c
  - LedBlinkExit, [2158](#)
  - LedBlinkInit, [2158](#)
  - LedBlinkTask, [2159](#)
- ARMCM0\_STM32F0\_Nucleo\_F091RC\_IAR/Boot/led.h
  - LedBlinkExit, [2393](#)
  - LedBlinkInit, [2393](#)
  - LedBlinkTask, [2393](#)
- ARMCM0\_STM32F0\_Nucleo\_F091RC\_IAR/Boot/main.↔
  - c
  - HAL\_MspDeInit, [2628](#)
  - HAL\_MspInit, [2628](#)
  - Init, [2628](#)
  - main, [2628](#)
  - SystemClock\_Config, [2629](#)
- ARMCM0\_STM32F0\_Nucleo\_F091RC\_IAR/Prog/led.c
  - LedInit, [2160](#)
  - LedToggle, [2160](#)
- ARMCM0\_STM32F0\_Nucleo\_F091RC\_IAR/Prog/led.h
  - LedInit, [2394](#)
  - LedToggle, [2395](#)
- ARMCM0\_STM32F0\_Nucleo\_F091RC\_IAR/Prog/main.↔
  - c
  - HAL\_MspDeInit, [2630](#)
  - HAL\_MspInit, [2630](#)
  - Init, [2630](#)
  - main, [2631](#)
  - SystemClock\_Config, [2631](#)
- ARMCM0\_STM32F0\_Nucleo\_F091RC\_Keil/Boot/hooks.↔
  - c
  - BackDoorEntryHook, [1661](#)
  - BackDoorInitHook, [1661](#)
  - CopInitHook, [1661](#)
  - CopServiceHook, [1662](#)
  - CpuUserProgramStartHook, [1662](#)
  - NvmDoneHook, [1662](#)
  - NvmEraseHook, [1662](#)
  - NvmInitHook, [1663](#)
  - NvmReinitHook, [1663](#)
  - NvmWriteHook, [1663](#)
  - XcpGetSeedHook, [1664](#)
  - XcpVerifyKeyHook, [1664](#)
- ARMCM0\_STM32F0\_Nucleo\_F091RC\_Keil/Boot/led.c
  - LedBlinkExit, [2161](#)
  - LedBlinkInit, [2161](#)
  - LedBlinkTask, [2162](#)
- ARMCM0\_STM32F0\_Nucleo\_F091RC\_Keil/Boot/led.h
  - LedBlinkExit, [2396](#)
  - LedBlinkInit, [2396](#)
  - LedBlinkTask, [2396](#)
- ARMCM0\_STM32F0\_Nucleo\_F091RC\_Keil/Boot/main.↔
  - c
  - HAL\_MspDeInit, [2632](#)
  - HAL\_MspInit, [2633](#)
  - Init, [2633](#)
  - main, [2633](#)
  - SystemClock\_Config, [2633](#)
- ARMCM0\_STM32F0\_Nucleo\_F091RC\_Keil/Prog/led.c
  - LedInit, [2163](#)
  - LedToggle, [2163](#)
- ARMCM0\_STM32F0\_Nucleo\_F091RC\_Keil/Prog/led.h
  - LedInit, [2397](#)
  - LedToggle, [2398](#)
- ARMCM0\_STM32F0\_Nucleo\_F091RC\_Keil/Prog/main.↔
  - c
  - HAL\_MspDeInit, [2635](#)
  - HAL\_MspInit, [2635](#)
  - Init, [2635](#)
  - main, [2635](#)
  - SystemClock\_Config, [2636](#)
- ARMCM0\_STM32G0/GCC/cpu\_comp.c
  - CpulrqDisable, [1199](#)
  - CpulrqEnable, [1199](#)
- ARMCM0\_STM32G0/IAR/cpu\_comp.c
  - CpulrqDisable, [1200](#)
  - CpulrqEnable, [1200](#)
- ARMCM0\_STM32G0/Keil/cpu\_comp.c
  - CpulrqDisable, [1200](#)
  - CpulrqEnable, [1200](#)

- CpuIrqDisable, [1201](#)
- CpuIrqEnable, [1201](#)
- ARMCM0\_STM32G0/cpu.c
  - CpuInit, [1141](#)
  - CpuMemCopy, [1141](#)
  - CpuMemSet, [1141](#)
  - CpuStartUserProgram, [1142](#)
  - CpuUserProgramStartHook, [1142](#)
- ARMCM0\_STM32G0/flash.c
  - blockInfo, [1301](#)
  - bootBlockInfo, [1301](#)
  - FlashAddToBlock, [1295](#)
  - FlashDone, [1296](#)
  - FlashErase, [1296](#)
  - FlashEraseSectors, [1296](#)
  - FlashGetSector, [1297](#)
  - FlashGetSectorBaseAddr, [1297](#)
  - FlashGetSectorSize, [1298](#)
  - FlashGetUserProgBaseAddress, [1298](#)
  - FlashInit, [1298](#)
  - FlashInitBlock, [1298](#)
  - flashLayout, [1302](#)
  - FlashReinit, [1299](#)
  - FlashSwitchBlock, [1299](#)
  - FlashVerifyChecksum, [1300](#)
  - FlashWrite, [1300](#)
  - FlashWriteBlock, [1300](#)
  - FlashWriteChecksum, [1301](#)
- ARMCM0\_STM32G0/flash.h
  - FlashDone, [1444](#)
  - FlashErase, [1444](#)
  - FlashGetUserProgBaseAddress, [1445](#)
  - FlashInit, [1445](#)
  - FlashReinit, [1445](#)
  - FlashVerifyChecksum, [1446](#)
  - FlashWrite, [1446](#)
  - FlashWriteChecksum, [1446](#)
- ARMCM0\_STM32G0/nvm.c
  - NvmDone, [2976](#)
  - NvmDoneHook, [2976](#)
  - NvmErase, [2976](#)
  - NvmEraseHook, [2977](#)
  - NvmGetUserProgBaseAddress, [2977](#)
  - NvmInit, [2977](#)
  - NvmInitHook, [2978](#)
  - NvmReinit, [2978](#)
  - NvmReinitHook, [2978](#)
  - NvmVerifyChecksum, [2978](#)
  - NvmWrite, [2979](#)
  - NvmWriteHook, [2979](#)
- ARMCM0\_STM32G0/types.h
  - blt\_addr, [3536](#)
  - blt\_bool, [3536](#)
  - blt\_char, [3536](#)
  - blt\_int16s, [3536](#)
  - blt\_int16u, [3537](#)
  - blt\_int32s, [3537](#)
  - blt\_int32u, [3537](#)
  - blt\_int64s, [3537](#)
  - blt\_int64u, [3537](#)
  - blt\_int8s, [3537](#)
  - blt\_int8u, [3537](#)
- ARMCM0\_STM32G0\_Nucleo\_G071RB\_CubeIDE/↔
  - Boot/App/app.c
    - AppInit, [393](#)
    - AppTask, [393](#)
  - ARMCM0\_STM32G0\_Nucleo\_G071RB\_CubeIDE/↔
    - Boot/App/app.h
      - AppInit, [436](#)
      - AppTask, [436](#)
    - ARMCM0\_STM32G0\_Nucleo\_G071RB\_CubeIDE/↔
      - Boot/App/hooks.c
        - BackDoorEntryHook, [1666](#)
        - BackDoorInitHook, [1666](#)
        - CopInitHook, [1666](#)
        - CopServiceHook, [1667](#)
        - CpuUserProgramStartHook, [1667](#)
        - NvmDoneHook, [1667](#)
        - NvmEraseHook, [1667](#)
        - NvmInitHook, [1668](#)
        - NvmReinitHook, [1668](#)
        - NvmWriteHook, [1668](#)
        - XcpGetSeedHook, [1669](#)
        - XcpVerifyKeyHook, [1669](#)
      - ARMCM0\_STM32G0\_Nucleo\_G071RB\_CubeIDE/↔
        - Boot/App/led.c
          - LedBlinkExit, [2164](#)
          - LedBlinkInit, [2164](#)
          - LedBlinkTask, [2165](#)
        - ARMCM0\_STM32G0\_Nucleo\_G071RB\_CubeIDE/↔
          - Boot/App/led.h
            - LedBlinkExit, [2399](#)
            - LedBlinkInit, [2399](#)
            - LedBlinkTask, [2399](#)
          - ARMCM0\_STM32G0\_Nucleo\_G071RB\_CubeIDE/↔
            - Prog/App/app.c
              - AppInit, [394](#)
              - AppTask, [394](#)
            - ARMCM0\_STM32G0\_Nucleo\_G071RB\_CubeIDE/↔
              - Prog/App/app.h
                - AppInit, [438](#)
                - AppTask, [438](#)
              - ARMCM0\_STM32G0\_Nucleo\_G071RB\_CubeIDE/↔
                - Prog/App/led.c
                  - LedInit, [2166](#)
                  - LedToggle, [2166](#)
                - ARMCM0\_STM32G0\_Nucleo\_G071RB\_CubeIDE/↔
                  - Prog/App/led.h
                    - LedInit, [2400](#)
                    - LedToggle, [2401](#)
                  - ARMCM0\_STM32G0\_Nucleo\_G071RB\_GCC/↔
                    - Boot/hooks.c
                      - BackDoorEntryHook, [1671](#)
                      - BackDoorInitHook, [1671](#)
                      - CopInitHook, [1671](#)
                      - CopServiceHook, [1672](#)

- CpuUserProgramStartHook, [1672](#)
- NvmDoneHook, [1672](#)
- NvmEraseHook, [1672](#)
- NvmInitHook, [1673](#)
- NvmReinitHook, [1673](#)
- NvmWriteHook, [1673](#)
- XcpGetSeedHook, [1674](#)
- XcpVerifyKeyHook, [1674](#)
- ARMCM0\_STM32G0\_Nucleo\_G071RB\_GCC/↔
  - Boot/led.c
  - LedBlinkExit, [2167](#)
  - LedBlinkInit, [2167](#)
  - LedBlinkTask, [2168](#)
- ARMCM0\_STM32G0\_Nucleo\_G071RB\_GCC/↔
  - Boot/led.h
  - LedBlinkExit, [2402](#)
  - LedBlinkInit, [2402](#)
  - LedBlinkTask, [2402](#)
- ARMCM0\_STM32G0\_Nucleo\_G071RB\_GCC/↔
  - Boot/main.c
  - HAL\_MspDeInit, [2637](#)
  - HAL\_MspInit, [2637](#)
  - Init, [2637](#)
  - main, [2638](#)
  - SystemClock\_Config, [2638](#)
- ARMCM0\_STM32G0\_Nucleo\_G071RB\_GCC/↔
  - Prog/led.c
  - LedInit, [2169](#)
  - LedToggle, [2169](#)
- ARMCM0\_STM32G0\_Nucleo\_G071RB\_GCC/↔
  - Prog/led.h
  - LedInit, [2403](#)
  - LedToggle, [2404](#)
- ARMCM0\_STM32G0\_Nucleo\_G071RB\_GCC/↔
  - Prog/main.c
  - HAL\_MspDeInit, [2639](#)
  - HAL\_MspInit, [2640](#)
  - Init, [2640](#)
  - main, [2640](#)
  - SystemClock\_Config, [2640](#)
  - VectorBase\_Config, [2641](#)
- ARMCM0\_STM32G0\_Nucleo\_G071RB\_IAR/Boot/hooks.↔
  - c
  - BackDoorEntryHook, [1676](#)
  - BackDoorInitHook, [1676](#)
  - CopInitHook, [1676](#)
  - CopServiceHook, [1677](#)
  - CpuUserProgramStartHook, [1677](#)
  - NvmDoneHook, [1677](#)
  - NvmEraseHook, [1677](#)
  - NvmInitHook, [1678](#)
  - NvmReinitHook, [1678](#)
  - NvmWriteHook, [1678](#)
  - XcpGetSeedHook, [1679](#)
  - XcpVerifyKeyHook, [1679](#)
- ARMCM0\_STM32G0\_Nucleo\_G071RB\_IAR/Boot/led.↔
  - c
  - LedBlinkExit, [2170](#)
- LedBlinkInit, [2170](#)
- LedBlinkTask, [2171](#)
- ARMCM0\_STM32G0\_Nucleo\_G071RB\_IAR/Boot/led.↔
  - h
  - LedBlinkExit, [2405](#)
  - LedBlinkInit, [2405](#)
  - LedBlinkTask, [2405](#)
- ARMCM0\_STM32G0\_Nucleo\_G071RB\_IAR/Boot/main.↔
  - c
  - HAL\_MspDeInit, [2642](#)
  - HAL\_MspInit, [2642](#)
  - Init, [2642](#)
  - main, [2643](#)
  - SystemClock\_Config, [2643](#)
- ARMCM0\_STM32G0\_Nucleo\_G071RB\_IAR/Prog/led.↔
  - c
  - LedInit, [2172](#)
  - LedToggle, [2172](#)
- ARMCM0\_STM32G0\_Nucleo\_G071RB\_IAR/Prog/led.↔
  - h
  - LedInit, [2406](#)
  - LedToggle, [2407](#)
- ARMCM0\_STM32G0\_Nucleo\_G071RB\_IAR/Prog/main.↔
  - c
  - HAL\_MspDeInit, [2644](#)
  - HAL\_MspInit, [2645](#)
  - Init, [2645](#)
  - main, [2645](#)
  - SystemClock\_Config, [2645](#)
  - VectorBase\_Config, [2646](#)
- ARMCM0\_STM32G0\_Nucleo\_G071RB\_Keil/Boot/hooks.↔
  - c
  - BackDoorEntryHook, [1681](#)
  - BackDoorInitHook, [1681](#)
  - CopInitHook, [1681](#)
  - CopServiceHook, [1682](#)
  - CpuUserProgramStartHook, [1682](#)
  - NvmDoneHook, [1682](#)
  - NvmEraseHook, [1682](#)
  - NvmInitHook, [1683](#)
  - NvmReinitHook, [1683](#)
  - NvmWriteHook, [1683](#)
  - XcpGetSeedHook, [1684](#)
  - XcpVerifyKeyHook, [1684](#)
- ARMCM0\_STM32G0\_Nucleo\_G071RB\_Keil/Boot/led.↔
  - c
  - LedBlinkExit, [2173](#)
  - LedBlinkInit, [2173](#)
  - LedBlinkTask, [2174](#)
- ARMCM0\_STM32G0\_Nucleo\_G071RB\_Keil/Boot/led.↔
  - h
  - LedBlinkExit, [2408](#)
  - LedBlinkInit, [2408](#)
  - LedBlinkTask, [2408](#)
- ARMCM0\_STM32G0\_Nucleo\_G071RB\_Keil/Boot/main.↔
  - c
  - HAL\_MspDeInit, [2647](#)
  - HAL\_MspInit, [2647](#)



- Init, [2647](#)
- main, [2648](#)
- SystemClock\_Config, [2648](#)
- ARMCM0\_STM32G0\_Nucleo\_G071RB\_Keil/Prog/led.↔
  - c
  - LedInit, [2175](#)
  - LedToggle, [2175](#)
- ARMCM0\_STM32G0\_Nucleo\_G071RB\_Keil/Prog/led.↔
  - h
  - LedInit, [2409](#)
  - LedToggle, [2410](#)
- ARMCM0\_STM32G0\_Nucleo\_G071RB\_Keil/Prog/main.↔
  - c
  - HAL\_MspDeInit, [2649](#)
  - HAL\_MspInit, [2650](#)
  - Init, [2650](#)
  - main, [2650](#)
  - SystemClock\_Config, [2650](#)
  - VectorBase\_Config, [2651](#)
- ARMCM0\_XMC1/GCC/cpu\_comp.c
  - CpulrqDisable, [1202](#)
  - CpulrqEnable, [1203](#)
- ARMCM0\_XMC1/IAR/cpu\_comp.c
  - CpulrqDisable, [1204](#)
  - CpulrqEnable, [1204](#)
- ARMCM0\_XMC1/can.c
  - CanInit, [1076](#)
  - CanReceivePacket, [1076](#)
  - CanTransmitPacket, [1077](#)
- ARMCM0\_XMC1/cpu.c
  - CpuInit, [1144](#)
  - CpuMemCopy, [1144](#)
  - CpuMemSet, [1144](#)
  - CpuStartUserProgram, [1145](#)
  - CpuUserProgramStartHook, [1145](#)
- ARMCM0\_XMC1/flash.c
  - blockInfo, [1310](#)
  - bootBlockInfo, [1310](#)
  - FlashAddToBlock, [1304](#)
  - FlashDone, [1305](#)
  - FlashErase, [1305](#)
  - FlashEraseSectors, [1305](#)
  - FlashGetSector, [1306](#)
  - FlashGetSectorBaseAddr, [1306](#)
  - FlashGetUserProgBaseAddress, [1307](#)
  - FlashInit, [1307](#)
  - FlashInitBlock, [1307](#)
  - flashLayout, [1310](#)
  - FlashReinit, [1308](#)
  - FlashSwitchBlock, [1308](#)
  - FlashVerifyChecksum, [1308](#)
  - FlashWrite, [1308](#)
  - FlashWriteBlock, [1309](#)
  - FlashWriteChecksum, [1309](#)
- ARMCM0\_XMC1/flash.h
  - FlashDone, [1448](#)
  - FlashErase, [1448](#)
  - FlashGetUserProgBaseAddress, [1449](#)
  - FlashInit, [1449](#)
  - FlashReinit, [1449](#)
  - FlashVerifyChecksum, [1449](#)
  - FlashWrite, [1450](#)
  - FlashWriteChecksum, [1450](#)
- ARMCM0\_XMC1/nvm.c
  - NvmDone, [2981](#)
  - NvmDoneHook, [2981](#)
  - NvmErase, [2981](#)
  - NvmEraseHook, [2982](#)
  - NvmGetUserProgBaseAddress, [2982](#)
  - NvmInit, [2982](#)
  - NvmInitHook, [2983](#)
  - NvmReinit, [2983](#)
  - NvmReinitHook, [2983](#)
  - NvmVerifyChecksum, [2983](#)
  - NvmWrite, [2984](#)
  - NvmWriteHook, [2984](#)
- ARMCM0\_XMC1/types.h
  - blt\_addr, [3538](#)
  - blt\_bool, [3538](#)
  - blt\_char, [3538](#)
  - blt\_int16s, [3539](#)
  - blt\_int16u, [3539](#)
  - blt\_int32s, [3539](#)
  - blt\_int32u, [3539](#)
  - blt\_int8s, [3539](#)
  - blt\_int8u, [3539](#)
- ARMCM0\_XMC1\_XMC1400\_Boot\_Kit\_GCC/Boot/hooks.↔
  - c
  - BackDoorEntryHook, [1686](#)
  - BackDoorInitHook, [1686](#)
  - CopInitHook, [1686](#)
  - CopServiceHook, [1687](#)
  - CpuUserProgramStartHook, [1687](#)
  - NvmDoneHook, [1687](#)
  - NvmEraseHook, [1687](#)
  - NvmInitHook, [1688](#)
  - NvmReinitHook, [1688](#)
  - NvmWriteHook, [1688](#)
  - XcpGetSeedHook, [1689](#)
  - XcpVerifyKeyHook, [1689](#)
- ARMCM0\_XMC1\_XMC1400\_Boot\_Kit\_GCC/Boot/led.↔
  - c
  - LedBlinkExit, [2176](#)
  - LedBlinkInit, [2176](#)
  - LedBlinkTask, [2177](#)
- ARMCM0\_XMC1\_XMC1400\_Boot\_Kit\_GCC/Boot/led.↔
  - h
  - LedBlinkExit, [2411](#)
  - LedBlinkInit, [2411](#)
  - LedBlinkTask, [2411](#)
- ARMCM0\_XMC1\_XMC1400\_Boot\_Kit\_GCC/Boot/main.↔
  - c
  - main, [2652](#)
  - PostInit, [2652](#)
- ARMCM0\_XMC1\_XMC1400\_Boot\_Kit\_GCC/Prog/led.↔
  - c

- LedInit, [2178](#)
- LedToggle, [2178](#)
- ARMCM0\_XMC1\_XMC1400\_Boot\_Kit\_GCC/Prog/led.c↔
  - h
  - LedInit, [2412](#)
  - LedToggle, [2413](#)
- ARMCM0\_XMC1\_XMC1400\_Boot\_Kit\_GCC/Prog/main.c↔
  - c
  - Init, [2653](#)
  - main, [2653](#)
- ARMCM0\_XMC1\_XMC1400\_Boot\_Kit\_IAR/Boot/hooks.c↔
  - c
  - BackDoorEntryHook, [1691](#)
  - BackDoorInitHook, [1691](#)
  - CopInitHook, [1691](#)
  - CopServiceHook, [1692](#)
  - CpuUserProgramStartHook, [1692](#)
  - NvmDoneHook, [1692](#)
  - NvmEraseHook, [1692](#)
  - NvmlInitHook, [1693](#)
  - NvmReinitHook, [1693](#)
  - NvmWriteHook, [1693](#)
  - XcpGetSeedHook, [1694](#)
  - XcpVerifyKeyHook, [1694](#)
- ARMCM0\_XMC1\_XMC1400\_Boot\_Kit\_IAR/Boot/led.c
  - LedBlinkExit, [2179](#)
  - LedBlinkInit, [2179](#)
  - LedBlinkTask, [2180](#)
- ARMCM0\_XMC1\_XMC1400\_Boot\_Kit\_IAR/Boot/led.h
  - LedBlinkExit, [2414](#)
  - LedBlinkInit, [2414](#)
  - LedBlinkTask, [2414](#)
- ARMCM0\_XMC1\_XMC1400\_Boot\_Kit\_IAR/Boot/main.c↔
  - c
  - Init, [2655](#)
  - main, [2655](#)
  - PostInit, [2655](#)
- ARMCM0\_XMC1\_XMC1400\_Boot\_Kit\_IAR/Prog/led.c
  - LedInit, [2181](#)
  - LedToggle, [2181](#)
- ARMCM0\_XMC1\_XMC1400\_Boot\_Kit\_IAR/Prog/led.h
  - LedInit, [2415](#)
  - LedToggle, [2416](#)
- ARMCM0\_XMC1\_XMC1400\_Boot\_Kit\_IAR/Prog/main.c↔
  - c
  - Init, [2656](#)
  - main, [2656](#)
- ARMCM33\_STM32L5/GCC/cpu\_comp.c
  - CpulrqDisable, [1205](#)
  - CpulrqEnable, [1205](#)
- ARMCM33\_STM32L5/IAR/cpu\_comp.c
  - CpulrqDisable, [1206](#)
  - CpulrqEnable, [1206](#)
- ARMCM33\_STM32L5/Keil/cpu\_comp.c
  - CpulrqDisable, [1207](#)
  - CpulrqEnable, [1208](#)
- ARMCM33\_STM32L5/can.c
  - CanGetSpeedConfig, [1078](#)
- CanInit, [1079](#)
- CanReceivePacket, [1079](#)
- canTiming, [1080](#)
- CanTransmitPacket, [1079](#)
- ARMCM33\_STM32L5/cpu.c
  - CpuInit, [1147](#)
  - CpuMemCopy, [1147](#)
  - CpuMemSet, [1147](#)
  - CpuStartUserProgram, [1148](#)
  - CpuUserProgramStartHook, [1148](#)
- ARMCM33\_STM32L5/flash.c
  - blockInfo, [1319](#)
  - bootBlockInfo, [1319](#)
  - FlashAddToBlock, [1313](#)
  - FlashDone, [1313](#)
  - FlashErase, [1313](#)
  - FlashEraseSectors, [1314](#)
  - FlashGetBank, [1314](#)
  - FlashGetPage, [1314](#)
  - FlashGetPageSize, [1315](#)
  - FlashGetSectorIdx, [1315](#)
  - FlashGetUserProgBaseAddress, [1316](#)
  - FlashInit, [1316](#)
  - FlashInitBlock, [1316](#)
  - FlashIsDualBankMode, [1317](#)
  - flashLayout, [1320](#)
  - FlashReinit, [1317](#)
  - FlashSwitchBlock, [1317](#)
  - FlashVerifyChecksum, [1318](#)
  - FlashWrite, [1318](#)
  - FlashWriteBlock, [1318](#)
  - FlashWriteChecksum, [1319](#)
- ARMCM33\_STM32L5/flash.h
  - FlashDone, [1451](#)
  - FlashErase, [1452](#)
  - FlashGetUserProgBaseAddress, [1452](#)
  - FlashInit, [1453](#)
  - FlashReinit, [1453](#)
  - FlashVerifyChecksum, [1453](#)
  - FlashWrite, [1453](#)
  - FlashWriteChecksum, [1454](#)
- ARMCM33\_STM32L5/nvm.c
  - NvmDone, [2986](#)
  - NvmDoneHook, [2986](#)
  - NvmErase, [2986](#)
  - NvmEraseHook, [2987](#)
  - NvmGetUserProgBaseAddress, [2987](#)
  - NvmlInit, [2987](#)
  - NvmlInitHook, [2988](#)
  - NvmReinit, [2988](#)
  - NvmReinitHook, [2988](#)
  - NvmVerifyChecksum, [2988](#)
  - NvmWrite, [2989](#)
  - NvmWriteHook, [2989](#)
- ARMCM33\_STM32L5/types.h
  - blt\_addr, [3540](#)
  - blt\_bool, [3540](#)
  - blt\_char, [3540](#)



- blt\_int16s, [3540](#)
- blt\_int16u, [3541](#)
- blt\_int32s, [3541](#)
- blt\_int32u, [3541](#)
- blt\_int64s, [3541](#)
- blt\_int64u, [3541](#)
- blt\_int8s, [3541](#)
- blt\_int8u, [3541](#)
- ARMCM33\_STM32L5/usb.c
  - UsbFifoMgrCreate, [3576](#)
  - UsbFifoMgrInit, [3576](#)
  - UsbFifoMgrRead, [3576](#)
  - UsbFifoMgrScan, [3577](#)
  - UsbFifoMgrWrite, [3577](#)
  - UsbFree, [3578](#)
  - UsbInit, [3578](#)
  - UsbReceiveByte, [3578](#)
  - UsbReceivePacket, [3578](#)
  - UsbReceivePipeBulkOUT, [3579](#)
  - UsbTransmitByte, [3579](#)
  - UsbTransmitPacket, [3579](#)
  - UsbTransmitPipeBulkIN, [3580](#)
- ARMCM33\_STM32L5\_Nucleo\_L552ZE\_CubeIDE/↔
  - Boot/App/app.c
    - ApplInit, [395](#)
    - AppTask, [396](#)
- ARMCM33\_STM32L5\_Nucleo\_L552ZE\_CubeIDE/↔
  - Boot/App/app.h
    - ApplInit, [439](#)
    - AppTask, [439](#)
- ARMCM33\_STM32L5\_Nucleo\_L552ZE\_CubeIDE/↔
  - Boot/App/hooks.c
    - BackDoorEntryHook, [1696](#)
    - BackDoorInitHook, [1696](#)
    - CopInitHook, [1696](#)
    - CopServiceHook, [1697](#)
    - CpuUserProgramStartHook, [1697](#)
    - NvmDoneHook, [1697](#)
    - NvmEraseHook, [1697](#)
    - NvmInitHook, [1698](#)
    - NvmReinitHook, [1698](#)
    - NvmWriteHook, [1698](#)
    - UsbConnectHook, [1699](#)
    - UsbEnterLowPowerModeHook, [1699](#)
    - UsbLeaveLowPowerModeHook, [1699](#)
    - XcpGetSeedHook, [1700](#)
    - XcpVerifyKeyHook, [1700](#)
- ARMCM33\_STM32L5\_Nucleo\_L552ZE\_CubeIDE/↔
  - Boot/App/led.c
    - LedBlinkExit, [2182](#)
    - LedBlinkInit, [2182](#)
    - LedBlinkTask, [2183](#)
- ARMCM33\_STM32L5\_Nucleo\_L552ZE\_CubeIDE/↔
  - Boot/App/led.h
    - LedBlinkExit, [2417](#)
    - LedBlinkInit, [2417](#)
    - LedBlinkTask, [2417](#)
- ARMCM33\_STM32L5\_Nucleo\_L552ZE\_CubeIDE/↔
  - Prog/App/app.c
    - ApplInit, [397](#)
    - AppTask, [397](#)
- ARMCM33\_STM32L5\_Nucleo\_L552ZE\_CubeIDE/↔
  - Prog/App/app.h
    - ApplInit, [440](#)
    - AppTask, [440](#)
- ARMCM33\_STM32L5\_Nucleo\_L552ZE\_CubeIDE/↔
  - Prog/App/led.c
    - LedInit, [2184](#)
    - LedToggle, [2184](#)
- ARMCM33\_STM32L5\_Nucleo\_L552ZE\_CubeIDE/↔
  - Prog/App/led.h
    - LedInit, [2418](#)
    - LedToggle, [2419](#)
- ARMCM33\_STM32L5\_Nucleo\_L552ZE\_GCC/↔
  - Boot/hooks.c
    - BackDoorEntryHook, [1702](#)
    - BackDoorInitHook, [1702](#)
    - CopInitHook, [1702](#)
    - CopServiceHook, [1703](#)
    - CpuUserProgramStartHook, [1703](#)
    - NvmDoneHook, [1703](#)
    - NvmEraseHook, [1703](#)
    - NvmInitHook, [1704](#)
    - NvmReinitHook, [1704](#)
    - NvmWriteHook, [1704](#)
    - UsbConnectHook, [1705](#)
    - UsbEnterLowPowerModeHook, [1705](#)
    - UsbLeaveLowPowerModeHook, [1705](#)
    - XcpGetSeedHook, [1705](#)
    - XcpVerifyKeyHook, [1706](#)
- ARMCM33\_STM32L5\_Nucleo\_L552ZE\_GCC/↔
  - Boot/led.c
    - LedBlinkExit, [2185](#)
    - LedBlinkInit, [2185](#)
    - LedBlinkTask, [2186](#)
- ARMCM33\_STM32L5\_Nucleo\_L552ZE\_GCC/↔
  - Boot/led.h
    - LedBlinkExit, [2420](#)
    - LedBlinkInit, [2420](#)
    - LedBlinkTask, [2420](#)
- ARMCM33\_STM32L5\_Nucleo\_L552ZE\_GCC/↔
  - Boot/main.c
    - HAL\_MspDeInit, [2658](#)
    - HAL\_MspInit, [2658](#)
    - Init, [2658](#)
    - main, [2658](#)
    - SystemClock\_Config, [2659](#)
- ARMCM33\_STM32L5\_Nucleo\_L552ZE\_GCC/↔
  - Prog/led.c
    - LedInit, [2187](#)
    - LedToggle, [2187](#)
- ARMCM33\_STM32L5\_Nucleo\_L552ZE\_GCC/↔
  - Prog/led.h
    - LedInit, [2421](#)
    - LedToggle, [2422](#)



- CpuMemCopy, [1150](#)
- CpuMemSet, [1150](#)
- CpuStartUserProgram, [1151](#)
- CpuUserProgramStartHook, [1151](#)
- ARMCM3\_EFM32/flash.c
  - blockInfo, [1328](#)
  - bootBlockInfo, [1328](#)
  - FlashAddToBlock, [1322](#)
  - FlashCalcPageSize, [1323](#)
  - FlashDone, [1323](#)
  - FlashErase, [1323](#)
  - FlashEraseSectors, [1324](#)
  - FlashGetSector, [1324](#)
  - FlashGetSectorBaseAddr, [1324](#)
  - FlashGetSectorSize, [1325](#)
  - FlashGetUserProgBaseAddress, [1325](#)
  - FlashInit, [1325](#)
  - FlashInitBlock, [1326](#)
  - flashLayout, [1329](#)
  - FlashReinit, [1326](#)
  - FlashSwitchBlock, [1326](#)
  - FlashVerifyChecksum, [1327](#)
  - FlashWrite, [1327](#)
  - FlashWriteBlock, [1327](#)
  - FlashWriteChecksum, [1328](#)
- ARMCM3\_EFM32/flash.h
  - FlashDone, [1455](#)
  - FlashErase, [1455](#)
  - FlashGetUserProgBaseAddress, [1456](#)
  - FlashInit, [1456](#)
  - FlashReinit, [1456](#)
  - FlashVerifyChecksum, [1457](#)
  - FlashWrite, [1457](#)
  - FlashWriteChecksum, [1457](#)
- ARMCM3\_EFM32/nvm.c
  - NvmDone, [2991](#)
  - NvmDoneHook, [2991](#)
  - NvmErase, [2991](#)
  - NvmEraseHook, [2992](#)
  - NvmGetUserProgBaseAddress, [2992](#)
  - NvmInit, [2992](#)
  - NvmInitHook, [2993](#)
  - NvmReinit, [2993](#)
  - NvmReinitHook, [2993](#)
  - NvmVerifyChecksum, [2993](#)
  - NvmWrite, [2994](#)
  - NvmWriteHook, [2994](#)
- ARMCM3\_EFM32/types.h
  - blt\_addr, [3542](#)
  - blt\_bool, [3542](#)
  - blt\_char, [3542](#)
  - blt\_int16s, [3543](#)
  - blt\_int16u, [3543](#)
  - blt\_int32s, [3543](#)
  - blt\_int32u, [3543](#)
  - blt\_int8s, [3543](#)
  - blt\_int8u, [3543](#)
- ARMCM3\_EFM32\_Olimex\_EM32G880F128STK\_GC↔
  - C/Boot/cstart.c
    - main, [1247](#)
    - reset\_handler, [1248](#)
- ARMCM3\_EFM32\_Olimex\_EM32G880F128STK\_GC↔
  - C/Boot/hooks.c
    - BackDoorEntryHook, [1720](#)
    - BackDoorInitHook, [1720](#)
    - CopInitHook, [1720](#)
    - CopServiceHook, [1720](#)
    - CpuUserProgramStartHook, [1721](#)
    - NvmDoneHook, [1721](#)
    - NvmEraseHook, [1721](#)
    - NvmInitHook, [1722](#)
    - NvmReinitHook, [1722](#)
    - NvmWriteHook, [1722](#)
    - XcpGetSeedHook, [1723](#)
    - XcpVerifyKeyHook, [1723](#)
- ARMCM3\_EFM32\_Olimex\_EM32G880F128STK\_GC↔
  - C/Boot/main.c
    - Init, [2673](#)
    - main, [2673](#)
- ARMCM3\_EFM32\_Olimex\_EM32G880F128STK\_GC↔
  - C/Boot/vectors.c
    - reset\_handler, [3623](#)
    - UnusedISR, [3624](#)
- ARMCM3\_EFM32\_Olimex\_EM32G880F128STK\_GC↔
  - C/Prog/cstart.c
    - main, [1249](#)
    - reset\_handler, [1249](#)
- ARMCM3\_EFM32\_Olimex\_EM32G880F128STK\_GC↔
  - C/Prog/led.c
    - LedInit, [2194](#)
    - LedToggle, [2194](#)
- ARMCM3\_EFM32\_Olimex\_EM32G880F128STK\_GC↔
  - C/Prog/led.h
    - LedInit, [2429](#)
    - LedToggle, [2429](#)
- ARMCM3\_EFM32\_Olimex\_EM32G880F128STK\_GC↔
  - C/Prog/main.c
    - Init, [2674](#)
    - main, [2674](#)
- ARMCM3\_EFM32\_Olimex\_EM32G880F128STK\_GC↔
  - C/Prog/vectors.c
    - reset\_handler, [3625](#)
    - UnusedISR, [3625](#)
- ARMCM3\_EFM32\_Olimex\_EM32G880F128STK\_IA↔
  - R/Boot/hooks.c
    - BackDoorEntryHook, [1725](#)
    - BackDoorInitHook, [1725](#)
    - CopInitHook, [1725](#)
    - CopServiceHook, [1726](#)
    - CpuUserProgramStartHook, [1726](#)
    - NvmDoneHook, [1726](#)
    - NvmEraseHook, [1726](#)
    - NvmInitHook, [1727](#)
    - NvmReinitHook, [1727](#)
    - NvmWriteHook, [1727](#)

- XcpGetSeedHook, [1728](#)
- XcpVerifyKeyHook, [1728](#)
- ARMCM3\_EFM32\_Olimex\_EM32G880F128STK\_IA↔
  - R/Boot/main.c
  - Init, [2675](#)
  - main, [2675](#)
- ARMCM3\_EFM32\_Olimex\_EM32G880F128STK\_IA↔
  - R/Boot/vectors.c
  - reset\_handler, [3627](#)
  - UnusedISR, [3627](#)
- ARMCM3\_EFM32\_Olimex\_EM32G880F128STK\_IA↔
  - R/Prog/led.c
  - LedInit, [2195](#)
  - LedToggle, [2195](#)
- ARMCM3\_EFM32\_Olimex\_EM32G880F128STK\_IA↔
  - R/Prog/led.h
  - LedInit, [2431](#)
  - LedToggle, [2431](#)
- ARMCM3\_EFM32\_Olimex\_EM32G880F128STK\_IA↔
  - R/Prog/main.c
  - Init, [2676](#)
  - main, [2677](#)
- ARMCM3\_EFM32\_Olimex\_EM32G880F128STK\_IA↔
  - R/Prog/vectors.c
  - UnusedISR, [3628](#)
- ARMCM3\_LM3S/GCC/cpu\_comp.c
  - CpulrqDisable, [1211](#)
  - CpulrqEnable, [1211](#)
- ARMCM3\_LM3S/IAR/cpu\_comp.c
  - CpulrqDisable, [1212](#)
  - CpulrqEnable, [1213](#)
- ARMCM3\_LM3S/can.c
  - CanInit, [1082](#)
  - CanReceivePacket, [1082](#)
  - CanSetBittiming, [1082](#)
  - CanTransmitPacket, [1082](#)
- ARMCM3\_LM3S/cpu.c
  - Cpulnit, [1153](#)
  - CpuMemCopy, [1153](#)
  - CpuMemSet, [1153](#)
  - CpuStartUserProgram, [1154](#)
  - CpuUserProgramStartHook, [1154](#)
- ARMCM3\_LM3S/flash.c
  - blockInfo, [1337](#)
  - bootBlockInfo, [1337](#)
  - FlashAddToBlock, [1331](#)
  - FlashDone, [1332](#)
  - FlashErase, [1332](#)
  - FlashEraseSectors, [1332](#)
  - FlashGetSector, [1333](#)
  - FlashGetSectorBaseAddr, [1333](#)
  - FlashGetSectorSize, [1334](#)
  - FlashGetUserProgBaseAddress, [1334](#)
  - FlashInit, [1334](#)
  - FlashInitBlock, [1334](#)
  - flashLayout, [1338](#)
  - FlashReinit, [1335](#)
  - FlashSwitchBlock, [1335](#)
  - FlashVerifyChecksum, [1336](#)
  - FlashWrite, [1336](#)
  - FlashWriteBlock, [1336](#)
  - FlashWriteChecksum, [1337](#)
- ARMCM3\_LM3S/flash.h
  - FlashDone, [1459](#)
  - FlashErase, [1459](#)
  - FlashGetUserProgBaseAddress, [1460](#)
  - FlashInit, [1460](#)
  - FlashReinit, [1460](#)
  - FlashVerifyChecksum, [1460](#)
  - FlashWrite, [1461](#)
  - FlashWriteChecksum, [1461](#)
- ARMCM3\_LM3S/nvm.c
  - NvmDone, [2996](#)
  - NvmDoneHook, [2996](#)
  - NvmErase, [2996](#)
  - NvmEraseHook, [2997](#)
  - NvmGetUserProgBaseAddress, [2997](#)
  - Nvmlnit, [2997](#)
  - NvmlnitHook, [2998](#)
  - NvmReinit, [2998](#)
  - NvmReinitHook, [2998](#)
  - NvmVerifyChecksum, [2998](#)
  - NvmWrite, [2999](#)
  - NvmWriteHook, [2999](#)
- ARMCM3\_LM3S/types.h
  - blt\_addr, [3544](#)
  - blt\_bool, [3544](#)
  - blt\_char, [3544](#)
  - blt\_int16s, [3544](#)
  - blt\_int16u, [3545](#)
  - blt\_int32s, [3545](#)
  - blt\_int32u, [3545](#)
  - blt\_int8s, [3545](#)
  - blt\_int8u, [3545](#)
- ARMCM3\_LM3S\_EK\_LM3S6965\_GCC/Boot/cstart.c
  - main, [1250](#)
  - reset\_handler, [1250](#)
- ARMCM3\_LM3S\_EK\_LM3S6965\_GCC/Boot/hooks.c
  - BackDoorEntryHook, [1730](#)
  - BackDoorInitHook, [1731](#)
  - canUse, [1736](#)
  - CopInitHook, [1731](#)
  - CopServiceHook, [1731](#)
  - CpuUserProgramStartHook, [1731](#)
  - FileFirmwareUpdateCompletedHook, [1732](#)
  - FileFirmwareUpdateErrorHook, [1732](#)
  - FileFirmwareUpdateLogHook, [1732](#)
  - FileFirmwareUpdateStartedHook, [1733](#)
  - FileGetFirmwareFilenameHook, [1733](#)
  - FileIsFirmwareUpdateRequestedHook, [1733](#)
  - handle, [1736](#)
  - NvmDoneHook, [1733](#)
  - NvmEraseHook, [1734](#)
  - NvmlnitHook, [1734](#)
  - NvmReinitHook, [1734](#)
  - NvmWriteHook, [1735](#)

- XcpGetSeedHook, [1735](#)
- XcpVerifyKeyHook, [1736](#)
- ARMCM3\_LM3S\_EK\_LM3S6965\_GCC/Boot/main.c
  - Init, [2678](#)
  - main, [2678](#)
- ARMCM3\_LM3S\_EK\_LM3S6965\_GCC/Boot/shared↔
  - \_params.c
    - \_\_attribute\_\_, [3068](#)
    - SharedParamsCalculateChecksum, [3069](#)
    - SharedParamsInit, [3069](#)
    - SharedParamsReadByIndex, [3069](#)
    - SharedParamsValidateBuffer, [3070](#)
    - SharedParamsVerifyChecksum, [3070](#)
    - SharedParamsWriteByIndex, [3070](#)
    - SharedParamsWriteChecksum, [3071](#)
- ARMCM3\_LM3S\_EK\_LM3S6965\_GCC/Boot/shared↔
  - \_params.h
    - SharedParamsInit, [3171](#)
    - SharedParamsReadByIndex, [3171](#)
    - SharedParamsWriteByIndex, [3172](#)
- ARMCM3\_LM3S\_EK\_LM3S6965\_GCC/Boot/vectors.c
  - reset\_handler, [3629](#)
  - UnusedISR, [3629](#)
- ARMCM3\_LM3S\_EK\_LM3S6965\_GCC/Prog/cstart.c
  - main, [1251](#)
  - reset\_handler, [1251](#)
- ARMCM3\_LM3S\_EK\_LM3S6965\_GCC/Prog/led.c
  - LedInit, [2196](#)
  - LedToggle, [2197](#)
- ARMCM3\_LM3S\_EK\_LM3S6965\_GCC/Prog/led.h
  - LedInit, [2432](#)
  - LedToggle, [2432](#)
- ARMCM3\_LM3S\_EK\_LM3S6965\_GCC/Prog/main.c
  - Init, [2679](#)
  - main, [2679](#)
- ARMCM3\_LM3S\_EK\_LM3S6965\_GCC/Prog/shared↔
  - \_params.c
    - \_\_attribute\_\_, [3072](#)
    - SharedParamsCalculateChecksum, [3073](#)
    - SharedParamsInit, [3073](#)
    - SharedParamsReadByIndex, [3074](#)
    - SharedParamsValidateBuffer, [3074](#)
    - SharedParamsVerifyChecksum, [3074](#)
    - SharedParamsWriteByIndex, [3075](#)
    - SharedParamsWriteChecksum, [3075](#)
- ARMCM3\_LM3S\_EK\_LM3S6965\_GCC/Prog/shared↔
  - \_params.h
    - SharedParamsInit, [3174](#)
    - SharedParamsReadByIndex, [3174](#)
    - SharedParamsWriteByIndex, [3174](#)
- ARMCM3\_LM3S\_EK\_LM3S6965\_GCC/Prog/time.c
  - TimerDeinit, [3220](#)
  - TimerGet, [3220](#)
  - TimerISRHandler, [3220](#)
  - TimerInit, [3220](#)
  - TimerSet, [3221](#)
- ARMCM3\_LM3S\_EK\_LM3S6965\_GCC/Prog/time.h
  - TimerDeinit, [3231](#)
  - TimerGet, [3231](#)
  - TimerISRHandler, [3231](#)
  - TimerInit, [3231](#)
  - TimerSet, [3232](#)
- ARMCM3\_LM3S\_EK\_LM3S6965\_GCC/Prog/vectors.c
  - reset\_handler, [3631](#)
  - UnusedISR, [3631](#)
- ARMCM3\_LM3S\_EK\_LM3S6965\_IAR/Boot/hooks.c
  - BackDoorEntryHook, [1738](#)
  - BackDoorInitHook, [1738](#)
  - canUse, [1744](#)
  - CopInitHook, [1739](#)
  - CopServiceHook, [1739](#)
  - CpuUserProgramStartHook, [1739](#)
  - FileFirmwareUpdateCompletedHook, [1739](#)
  - FileFirmwareUpdateErrorHook, [1740](#)
  - FileFirmwareUpdateLogHook, [1740](#)
  - FileFirmwareUpdateStartedHook, [1740](#)
  - FileGetFirmwareFilenameHook, [1741](#)
  - FileFirmwareUpdateRequestedHook, [1741](#)
  - handle, [1744](#)
  - NvmDoneHook, [1741](#)
  - NvmEraseHook, [1741](#)
  - NvmInitHook, [1742](#)
  - NvmReinitHook, [1742](#)
  - NvmWriteHook, [1742](#)
  - XcpGetSeedHook, [1743](#)
  - XcpVerifyKeyHook, [1743](#)
- ARMCM3\_LM3S\_EK\_LM3S6965\_IAR/Boot/main.c
  - Init, [2680](#)
  - main, [2681](#)
- ARMCM3\_LM3S\_EK\_LM3S6965\_IAR/Boot/shared↔
  - \_params.c
    - shared, [3079](#)
    - SharedParamsCalculateChecksum, [3077](#)
    - SharedParamsInit, [3077](#)
    - SharedParamsReadByIndex, [3077](#)
    - SharedParamsValidateBuffer, [3078](#)
    - SharedParamsVerifyChecksum, [3078](#)
    - SharedParamsWriteByIndex, [3078](#)
    - SharedParamsWriteChecksum, [3079](#)
- ARMCM3\_LM3S\_EK\_LM3S6965\_IAR/Boot/shared↔
  - \_params.h
    - SharedParamsInit, [3176](#)
    - SharedParamsReadByIndex, [3176](#)
    - SharedParamsWriteByIndex, [3177](#)
- ARMCM3\_LM3S\_EK\_LM3S6965\_IAR/Boot/vectors.c
  - reset\_handler, [3632](#)
  - UnusedISR, [3632](#)
- ARMCM3\_LM3S\_EK\_LM3S6965\_IAR/Prog/led.c
  - LedInit, [2198](#)
  - LedToggle, [2198](#)
- ARMCM3\_LM3S\_EK\_LM3S6965\_IAR/Prog/led.h
  - LedInit, [2433](#)
  - LedToggle, [2434](#)
- ARMCM3\_LM3S\_EK\_LM3S6965\_IAR/Prog/main.c
  - Init, [2682](#)
  - main, [2682](#)

- ARMCM3\_LM3S\_EK\_LM3S6965\_IAR/Prog/shared\_↔
  - params.c
  - shared, [3083](#)
  - SharedParamsCalculateChecksum, [3081](#)
  - SharedParamsInit, [3081](#)
  - SharedParamsReadByIndex, [3081](#)
  - SharedParamsValidateBuffer, [3082](#)
  - SharedParamsVerifyChecksum, [3082](#)
  - SharedParamsWriteByIndex, [3082](#)
  - SharedParamsWriteChecksum, [3083](#)
- ARMCM3\_LM3S\_EK\_LM3S6965\_IAR/Prog/shared\_↔
  - params.h
  - SharedParamsInit, [3178](#)
  - SharedParamsReadByIndex, [3178](#)
  - SharedParamsWriteByIndex, [3179](#)
- ARMCM3\_LM3S\_EK\_LM3S6965\_IAR/Prog/time.c
  - TimerDeinit, [3222](#)
  - TimerGet, [3222](#)
  - TimerISRHandler, [3223](#)
  - TimerInit, [3222](#)
  - TimerSet, [3223](#)
- ARMCM3\_LM3S\_EK\_LM3S6965\_IAR/Prog/time.h
  - TimerDeinit, [3233](#)
  - TimerGet, [3233](#)
  - TimerISRHandler, [3234](#)
  - TimerInit, [3233](#)
  - TimerSet, [3234](#)
- ARMCM3\_LM3S\_EK\_LM3S6965\_IAR/Prog/vectors.c
  - UnusedISR, [3633](#)
- ARMCM3\_LM3S\_EK\_LM3S8962\_GCC/Boot/cstart.c
  - main, [1252](#)
  - reset\_handler, [1253](#)
- ARMCM3\_LM3S\_EK\_LM3S8962\_GCC/Boot/flash\_↔
  - layout.c
  - flashLayout, [1517](#)
- ARMCM3\_LM3S\_EK\_LM3S8962\_GCC/Boot/hooks.c
  - BackDoorEntryHook, [1745](#)
  - BackDoorInitHook, [1746](#)
  - CopInitHook, [1746](#)
  - CopServiceHook, [1746](#)
  - CpuUserProgramStartHook, [1746](#)
  - NvmDoneHook, [1747](#)
  - NvmEraseHook, [1747](#)
  - NvmInitHook, [1747](#)
  - NvmReinitHook, [1748](#)
  - NvmWriteHook, [1748](#)
  - XcpGetSeedHook, [1748](#)
  - XcpVerifyKeyHook, [1749](#)
- ARMCM3\_LM3S\_EK\_LM3S8962\_GCC/Boot/main.c
  - Init, [2683](#)
  - main, [2683](#)
- ARMCM3\_LM3S\_EK\_LM3S8962\_GCC/Boot/vectors.c
  - reset\_handler, [3635](#)
  - UnusedISR, [3635](#)
- ARMCM3\_LM3S\_EK\_LM3S8962\_GCC/Prog/cstart.c
  - main, [1254](#)
  - reset\_handler, [1254](#)
- ARMCM3\_LM3S\_EK\_LM3S8962\_GCC/Prog/led.c
  - LedInit, [2199](#)
  - LedToggle, [2199](#)
- ARMCM3\_LM3S\_EK\_LM3S8962\_GCC/Prog/led.h
  - LedInit, [2435](#)
  - LedToggle, [2435](#)
- ARMCM3\_LM3S\_EK\_LM3S8962\_GCC/Prog/main.c
  - Init, [2684](#)
  - main, [2684](#)
- ARMCM3\_LM3S\_EK\_LM3S8962\_GCC/Prog/time.c
  - TimerDeinit, [3224](#)
  - TimerGet, [3224](#)
  - TimerISRHandler, [3225](#)
  - TimerInit, [3225](#)
  - TimerSet, [3225](#)
- ARMCM3\_LM3S\_EK\_LM3S8962\_GCC/Prog/time.h
  - TimerDeinit, [3235](#)
  - TimerGet, [3235](#)
  - TimerISRHandler, [3236](#)
  - TimerInit, [3236](#)
  - TimerSet, [3236](#)
- ARMCM3\_LM3S\_EK\_LM3S8962\_GCC/Prog/vectors.c
  - reset\_handler, [3636](#)
  - UnusedISR, [3636](#)
- ARMCM3\_LM3S\_EK\_LM3S8962\_IAR/Boot/flash\_↔
  - layout.c
  - flashLayout, [1518](#)
- ARMCM3\_LM3S\_EK\_LM3S8962\_IAR/Boot/hooks.c
  - BackDoorEntryHook, [1751](#)
  - BackDoorInitHook, [1751](#)
  - CopInitHook, [1751](#)
  - CopServiceHook, [1751](#)
  - CpuUserProgramStartHook, [1752](#)
  - NvmDoneHook, [1752](#)
  - NvmEraseHook, [1752](#)
  - NvmInitHook, [1753](#)
  - NvmReinitHook, [1753](#)
  - NvmWriteHook, [1753](#)
  - XcpGetSeedHook, [1754](#)
  - XcpVerifyKeyHook, [1754](#)
- ARMCM3\_LM3S\_EK\_LM3S8962\_IAR/Boot/main.c
  - Init, [2685](#)
  - main, [2686](#)
- ARMCM3\_LM3S\_EK\_LM3S8962\_IAR/Boot/vectors.c
  - reset\_handler, [3638](#)
  - UnusedISR, [3638](#)
- ARMCM3\_LM3S\_EK\_LM3S8962\_IAR/Prog/led.c
  - LedInit, [2200](#)
  - LedToggle, [2200](#)
- ARMCM3\_LM3S\_EK\_LM3S8962\_IAR/Prog/led.h
  - LedInit, [2436](#)
  - LedToggle, [2436](#)
- ARMCM3\_LM3S\_EK\_LM3S8962\_IAR/Prog/main.c
  - Init, [2687](#)
  - main, [2687](#)
- ARMCM3\_LM3S\_EK\_LM3S8962\_IAR/Prog/time.c
  - TimerDeinit, [3226](#)
  - TimerGet, [3227](#)
  - TimerISRHandler, [3227](#)



- TimerInit, [3227](#)
- TimerSet, [3227](#)
- ARMCM3\_LM3S\_EK\_LM3S8962\_IAR/Prog/time.h
  - TimerDeinit, [3238](#)
  - TimerGet, [3238](#)
  - TimerISRHandler, [3238](#)
  - TimerInit, [3238](#)
  - TimerSet, [3239](#)
- ARMCM3\_LM3S\_EK\_LM3S8962\_IAR/Prog/vectors.c
  - UnusedISR, [3639](#)
- ARMCM3\_STM32F1/GCC/cpu\_comp.c
  - CpuIrqDisable, [1214](#)
  - CpuIrqEnable, [1214](#)
- ARMCM3\_STM32F1/IAR/cpu\_comp.c
  - CpuIrqDisable, [1215](#)
  - CpuIrqEnable, [1215](#)
- ARMCM3\_STM32F1/Keil/cpu\_comp.c
  - CpuIrqDisable, [1216](#)
  - CpuIrqEnable, [1216](#)
- ARMCM3\_STM32F1/can.c
  - CanGetSpeedConfig, [1084](#)
  - CanInit, [1085](#)
  - CanReceivePacket, [1085](#)
  - canTiming, [1086](#)
  - CanTransmitPacket, [1085](#)
- ARMCM3\_STM32F1/cpu.c
  - CpuInit, [1156](#)
  - CpuMemCopy, [1156](#)
  - CpuMemSet, [1156](#)
  - CpuStartUserProgram, [1157](#)
  - CpuUserProgramStartHook, [1157](#)
- ARMCM3\_STM32F1/flash.c
  - blockInfo, [1344](#)
  - bootBlockInfo, [1344](#)
  - FlashAddToBlock, [1340](#)
  - FlashDone, [1341](#)
  - FlashErase, [1341](#)
  - FlashGetUserProgBaseAddress, [1341](#)
  - FlashInit, [1341](#)
  - FlashInitBlock, [1342](#)
  - flashLayout, [1345](#)
  - FlashReinit, [1342](#)
  - FlashSwitchBlock, [1342](#)
  - FlashVerifyChecksum, [1343](#)
  - FlashWrite, [1343](#)
  - FlashWriteBlock, [1343](#)
  - FlashWriteChecksum, [1344](#)
- ARMCM3\_STM32F1/flash.h
  - FlashDone, [1462](#)
  - FlashErase, [1463](#)
  - FlashGetUserProgBaseAddress, [1463](#)
  - FlashInit, [1464](#)
  - FlashReinit, [1464](#)
  - FlashVerifyChecksum, [1464](#)
  - FlashWrite, [1464](#)
  - FlashWriteChecksum, [1465](#)
- ARMCM3\_STM32F1/nvm.c
  - NvmDone, [3001](#)
  - NvmDoneHook, [3001](#)
  - NvmErase, [3001](#)
  - NvmEraseHook, [3002](#)
  - NvmGetUserProgBaseAddress, [3002](#)
  - NvmInit, [3002](#)
  - NvmInitHook, [3003](#)
  - NvmReinit, [3003](#)
  - NvmReinitHook, [3003](#)
  - NvmVerifyChecksum, [3003](#)
  - NvmWrite, [3004](#)
  - NvmWriteHook, [3004](#)
- ARMCM3\_STM32F1/types.h
  - blt\_addr, [3546](#)
  - blt\_bool, [3546](#)
  - blt\_char, [3546](#)
  - blt\_int16s, [3546](#)
  - blt\_int16u, [3547](#)
  - blt\_int32s, [3547](#)
  - blt\_int32u, [3547](#)
  - blt\_int8s, [3547](#)
  - blt\_int8u, [3547](#)
- ARMCM3\_STM32F1/usb.c
  - UsbFifoMgrCreate, [3582](#)
  - UsbFifoMgrInit, [3582](#)
  - UsbFifoMgrRead, [3583](#)
  - UsbFifoMgrScan, [3583](#)
  - UsbFifoMgrWrite, [3583](#)
  - UsbFree, [3584](#)
  - UsbInit, [3584](#)
  - UsbReceiveByte, [3584](#)
  - UsbReceivePacket, [3585](#)
  - UsbReceivePipeBulkOUT, [3585](#)
  - UsbTransmitByte, [3585](#)
  - UsbTransmitPacket, [3586](#)
  - UsbTransmitPipeBulkIN, [3586](#)
- ARMCM3\_STM32F1\_Nucleo\_F103RB\_CubeIDE/↔
  - Boot/App/app.c
    - AppInit, [398](#)
    - AppTask, [398](#)
- ARMCM3\_STM32F1\_Nucleo\_F103RB\_CubeIDE/↔
  - Boot/App/app.h
    - AppInit, [441](#)
    - AppTask, [441](#)
- ARMCM3\_STM32F1\_Nucleo\_F103RB\_CubeIDE/↔
  - Boot/App/flash\_layout.c
    - flashLayout, [1519](#)
- ARMCM3\_STM32F1\_Nucleo\_F103RB\_CubeIDE/↔
  - Boot/App/hooks.c
    - BackDoorEntryHook, [1756](#)
    - BackDoorInitHook, [1756](#)
    - CopInitHook, [1756](#)
    - CopServiceHook, [1757](#)
    - CpuUserProgramStartHook, [1757](#)
    - NvmDoneHook, [1757](#)
    - NvmEraseHook, [1757](#)
    - NvmInitHook, [1758](#)
    - NvmReinitHook, [1758](#)
    - NvmWriteHook, [1758](#)

- XcpGetSeedHook, [1759](#)
- XcpVerifyKeyHook, [1759](#)
- ARMCM3\_STM32F1\_Nucleo\_F103RB\_CubeIDE/↔
  - Boot/App/led.c
  - LedBlinkExit, [2202](#)
  - LedBlinkInit, [2202](#)
  - LedBlinkTask, [2202](#)
- ARMCM3\_STM32F1\_Nucleo\_F103RB\_CubeIDE/↔
  - Boot/App/led.h
  - LedBlinkExit, [2437](#)
  - LedBlinkInit, [2438](#)
  - LedBlinkTask, [2438](#)
- ARMCM3\_STM32F1\_Nucleo\_F103RB\_CubeIDE/↔
  - Prog/App/app.c
  - AppInit, [399](#)
  - AppTask, [399](#)
- ARMCM3\_STM32F1\_Nucleo\_F103RB\_CubeIDE/↔
  - Prog/App/app.h
  - AppInit, [443](#)
  - AppTask, [443](#)
- ARMCM3\_STM32F1\_Nucleo\_F103RB\_CubeIDE/↔
  - Prog/App/led.c
  - LedInit, [2203](#)
  - LedToggle, [2203](#)
- ARMCM3\_STM32F1\_Nucleo\_F103RB\_CubeIDE/↔
  - Prog/App/led.h
  - LedInit, [2439](#)
  - LedToggle, [2439](#)
- ARMCM3\_STM32F1\_Nucleo\_F103RB\_GCC/Boot/flash↔
  - \_layout.c
  - flashLayout, [1520](#)
- ARMCM3\_STM32F1\_Nucleo\_F103RB\_GCC/Boot/hooks.↔
  - c
  - BackDoorEntryHook, [1761](#)
  - BackDoorInitHook, [1761](#)
  - CopInitHook, [1761](#)
  - CopServiceHook, [1762](#)
  - CpuUserProgramStartHook, [1762](#)
  - NvmDoneHook, [1762](#)
  - NvmEraseHook, [1762](#)
  - NvmInitHook, [1763](#)
  - NvmReinitHook, [1763](#)
  - NvmWriteHook, [1763](#)
  - XcpGetSeedHook, [1764](#)
  - XcpVerifyKeyHook, [1764](#)
- ARMCM3\_STM32F1\_Nucleo\_F103RB\_GCC/Boot/led.↔
  - c
  - LedBlinkExit, [2205](#)
  - LedBlinkInit, [2205](#)
  - LedBlinkTask, [2205](#)
- ARMCM3\_STM32F1\_Nucleo\_F103RB\_GCC/Boot/led.↔
  - h
  - LedBlinkExit, [2440](#)
  - LedBlinkInit, [2441](#)
  - LedBlinkTask, [2441](#)
- ARMCM3\_STM32F1\_Nucleo\_F103RB\_GCC/Boot/main.↔
  - c
  - HAL\_MspDelInit, [2688](#)
- HAL\_MspInit, [2688](#)
- Init, [2688](#)
- main, [2689](#)
- SystemClock\_Config, [2689](#)
- ARMCM3\_STM32F1\_Nucleo\_F103RB\_GCC/Prog/led.↔
  - c
  - LedInit, [2206](#)
  - LedToggle, [2206](#)
- ARMCM3\_STM32F1\_Nucleo\_F103RB\_GCC/Prog/led.↔
  - h
  - LedInit, [2442](#)
  - LedToggle, [2442](#)
- ARMCM3\_STM32F1\_Nucleo\_F103RB\_GCC/Prog/main.↔
  - c
  - HAL\_MspDelInit, [2690](#)
  - HAL\_MspInit, [2691](#)
  - Init, [2691](#)
  - main, [2691](#)
  - SystemClock\_Config, [2691](#)
  - VectorBase\_Config, [2692](#)
- ARMCM3\_STM32F1\_Nucleo\_F103RB\_IAR/Boot/flash↔
  - \_layout.c
  - flashLayout, [1521](#)
- ARMCM3\_STM32F1\_Nucleo\_F103RB\_IAR/Boot/hooks.↔
  - c
  - BackDoorEntryHook, [1766](#)
  - BackDoorInitHook, [1766](#)
  - CopInitHook, [1766](#)
  - CopServiceHook, [1767](#)
  - CpuUserProgramStartHook, [1767](#)
  - NvmDoneHook, [1767](#)
  - NvmEraseHook, [1767](#)
  - NvmInitHook, [1768](#)
  - NvmReinitHook, [1768](#)
  - NvmWriteHook, [1768](#)
  - XcpGetSeedHook, [1769](#)
  - XcpVerifyKeyHook, [1769](#)
- ARMCM3\_STM32F1\_Nucleo\_F103RB\_IAR/Boot/led.c
  - LedBlinkExit, [2208](#)
  - LedBlinkInit, [2208](#)
  - LedBlinkTask, [2208](#)
- ARMCM3\_STM32F1\_Nucleo\_F103RB\_IAR/Boot/led.h
  - LedBlinkExit, [2443](#)
  - LedBlinkInit, [2444](#)
  - LedBlinkTask, [2444](#)
- ARMCM3\_STM32F1\_Nucleo\_F103RB\_IAR/Boot/main.↔
  - c
  - HAL\_MspDelInit, [2693](#)
  - HAL\_MspInit, [2693](#)
  - Init, [2693](#)
  - main, [2694](#)
  - SystemClock\_Config, [2694](#)
- ARMCM3\_STM32F1\_Nucleo\_F103RB\_IAR/Prog/led.c
  - LedInit, [2209](#)
  - LedToggle, [2209](#)
- ARMCM3\_STM32F1\_Nucleo\_F103RB\_IAR/Prog/led.h
  - LedInit, [2445](#)
  - LedToggle, [2445](#)



ARMCM3\_STM32F1\_Nucleo\_F103RB\_IAR/Prog/main.↔  
     c  
     HAL\_MspDelInit, 2695  
     HAL\_MsplInit, 2696  
     Init, 2696  
     main, 2696  
     SystemClock\_Config, 2696  
     VectorBase\_Config, 2697  
 ARMCM3\_STM32F1\_Nucleo\_F103RB\_Keil/Boot/flash↔  
     \_layout.c  
     flashLayout, 1522  
 ARMCM3\_STM32F1\_Nucleo\_F103RB\_Keil/Boot/hooks.↔  
     c  
     BackDoorEntryHook, 1771  
     BackDoorInitHook, 1771  
     CopInitHook, 1771  
     CopServiceHook, 1772  
     CpuUserProgramStartHook, 1772  
     NvmDoneHook, 1772  
     NvmEraseHook, 1772  
     NvmlInitHook, 1773  
     NvmReinitHook, 1773  
     NvmWriteHook, 1773  
     XcpGetSeedHook, 1774  
     XcpVerifyKeyHook, 1774  
 ARMCM3\_STM32F1\_Nucleo\_F103RB\_Keil/Boot/led.c  
     LedBlinkExit, 2211  
     LedBlinkInit, 2211  
     LedBlinkTask, 2211  
 ARMCM3\_STM32F1\_Nucleo\_F103RB\_Keil/Boot/led.h  
     LedBlinkExit, 2446  
     LedBlinkInit, 2447  
     LedBlinkTask, 2447  
 ARMCM3\_STM32F1\_Nucleo\_F103RB\_Keil/Boot/main.↔  
     c  
     HAL\_MspDelInit, 2698  
     HAL\_MsplInit, 2698  
     Init, 2698  
     main, 2699  
     SystemClock\_Config, 2699  
 ARMCM3\_STM32F1\_Nucleo\_F103RB\_Keil/Prog/led.c  
     LedInit, 2212  
     LedToggle, 2212  
 ARMCM3\_STM32F1\_Nucleo\_F103RB\_Keil/Prog/led.h  
     LedInit, 2448  
     LedToggle, 2448  
 ARMCM3\_STM32F1\_Nucleo\_F103RB\_Keil/Prog/main.↔  
     c  
     HAL\_MspDelInit, 2700  
     HAL\_MsplInit, 2701  
     Init, 2701  
     main, 2701  
     SystemClock\_Config, 2701  
     VectorBase\_Config, 2702  
 ARMCM3\_STM32F1\_Olimex\_STM32H103\_CubeID↔  
     E/Boot/App/app.c  
     AppInit, 400  
     AppTask, 401  
     E/Boot/App/app.h  
     AppInit, 444  
     AppTask, 444  
     E/Boot/App/flash\_layout.c  
     flashLayout, 1523  
     E/Boot/App/hooks.c  
     BackDoorEntryHook, 1776  
     BackDoorInitHook, 1776  
     CopInitHook, 1776  
     CopServiceHook, 1777  
     CpuUserProgramStartHook, 1777  
     NvmDoneHook, 1777  
     NvmEraseHook, 1777  
     NvmlInitHook, 1778  
     NvmReinitHook, 1778  
     NvmWriteHook, 1778  
     UsbConnectHook, 1779  
     UsbEnterLowPowerModeHook, 1779  
     UsbLeaveLowPowerModeHook, 1779  
     XcpGetSeedHook, 1779  
     XcpVerifyKeyHook, 1780  
     E/Boot/App/led.c  
     LedBlinkExit, 2214  
     LedBlinkInit, 2214  
     LedBlinkTask, 2214  
     E/Boot/App/led.h  
     LedBlinkExit, 2449  
     LedBlinkInit, 2450  
     LedBlinkTask, 2450  
     E/Prog/App/app.c  
     AppInit, 402  
     AppTask, 402  
     E/Prog/App/app.h  
     AppInit, 445  
     AppTask, 445  
     E/Prog/App/led.c  
     LedInit, 2215  
     LedToggle, 2216  
     E/Prog/App/led.h  
     LedInit, 2451  
     LedToggle, 2451  
     Boot/flash\_layout.c  
     flashLayout, 1524  
     Boot/hooks.c  
     BackDoorEntryHook, 1782  
     BackDoorInitHook, 1782  
     CopInitHook, 1782

- CopServiceHook, [1782](#)
- CpuUserProgramStartHook, [1783](#)
- NvmDoneHook, [1783](#)
- NvmEraseHook, [1783](#)
- NvmInitHook, [1784](#)
- NvmReinitHook, [1784](#)
- NvmWriteHook, [1784](#)
- UsbConnectHook, [1785](#)
- UsbEnterLowPowerModeHook, [1785](#)
- UsbLeaveLowPowerModeHook, [1785](#)
- XcpGetSeedHook, [1785](#)
- XcpVerifyKeyHook, [1786](#)
- ARMCM3\_STM32F1\_Olimex\_STM32H103\_GCC/↔
  - Boot/led.c
  - LedBlinkExit, [2217](#)
  - LedBlinkInit, [2217](#)
  - LedBlinkTask, [2218](#)
- ARMCM3\_STM32F1\_Olimex\_STM32H103\_GCC/↔
  - Boot/led.h
  - LedBlinkExit, [2452](#)
  - LedBlinkInit, [2453](#)
  - LedBlinkTask, [2453](#)
- ARMCM3\_STM32F1\_Olimex\_STM32H103\_GCC/↔
  - Boot/main.c
  - HAL\_MspDeInit, [2703](#)
  - HAL\_MspInit, [2703](#)
  - Init, [2703](#)
  - main, [2704](#)
  - SystemClock\_Config, [2704](#)
- ARMCM3\_STM32F1\_Olimex\_STM32H103\_GCC/↔
  - Prog/led.c
  - LedInit, [2219](#)
  - LedToggle, [2219](#)
- ARMCM3\_STM32F1\_Olimex\_STM32H103\_GCC/↔
  - Prog/led.h
  - LedInit, [2454](#)
  - LedToggle, [2454](#)
- ARMCM3\_STM32F1\_Olimex\_STM32H103\_GCC/↔
  - Prog/main.c
  - HAL\_MspDeInit, [2705](#)
  - HAL\_MspInit, [2706](#)
  - Init, [2706](#)
  - main, [2706](#)
  - SystemClock\_Config, [2706](#)
  - VectorBase\_Config, [2707](#)
- ARMCM3\_STM32F1\_Olimex\_STM32H103\_IAR/↔
  - Boot/flash\_layout.c
  - flashLayout, [1525](#)
- ARMCM3\_STM32F1\_Olimex\_STM32H103\_IAR/↔
  - Boot/hooks.c
  - BackDoorEntryHook, [1788](#)
  - BackDoorInitHook, [1788](#)
  - CopInitHook, [1788](#)
  - CopServiceHook, [1788](#)
  - CpuUserProgramStartHook, [1789](#)
  - NvmDoneHook, [1789](#)
  - NvmEraseHook, [1789](#)
  - NvmInitHook, [1790](#)
  - NvmReinitHook, [1790](#)
  - NvmWriteHook, [1790](#)
  - UsbConnectHook, [1791](#)
  - UsbEnterLowPowerModeHook, [1791](#)
  - UsbLeaveLowPowerModeHook, [1791](#)
  - XcpGetSeedHook, [1791](#)
  - XcpVerifyKeyHook, [1792](#)
- ARMCM3\_STM32F1\_Olimex\_STM32H103\_IAR/↔
  - Boot/led.c
  - LedBlinkExit, [2220](#)
  - LedBlinkInit, [2220](#)
  - LedBlinkTask, [2221](#)
- ARMCM3\_STM32F1\_Olimex\_STM32H103\_IAR/↔
  - Boot/led.h
  - LedBlinkExit, [2455](#)
  - LedBlinkInit, [2456](#)
  - LedBlinkTask, [2456](#)
- ARMCM3\_STM32F1\_Olimex\_STM32H103\_IAR/↔
  - Boot/main.c
  - HAL\_MspDeInit, [2708](#)
  - HAL\_MspInit, [2708](#)
  - Init, [2708](#)
  - main, [2709](#)
  - SystemClock\_Config, [2709](#)
- ARMCM3\_STM32F1\_Olimex\_STM32H103\_IAR/↔
  - Prog/led.c
  - LedInit, [2222](#)
  - LedToggle, [2222](#)
- ARMCM3\_STM32F1\_Olimex\_STM32H103\_IAR/↔
  - Prog/led.h
  - LedInit, [2457](#)
  - LedToggle, [2457](#)
- ARMCM3\_STM32F1\_Olimex\_STM32H103\_IAR/↔
  - Prog/main.c
  - HAL\_MspDeInit, [2710](#)
  - HAL\_MspInit, [2711](#)
  - Init, [2711](#)
  - main, [2711](#)
  - SystemClock\_Config, [2711](#)
  - VectorBase\_Config, [2712](#)
- ARMCM3\_STM32F1\_Olimex\_STM32H103\_Keil/↔
  - Boot/flash\_layout.c
  - flashLayout, [1526](#)
- ARMCM3\_STM32F1\_Olimex\_STM32H103\_Keil/↔
  - Boot/hooks.c
  - BackDoorEntryHook, [1794](#)
  - BackDoorInitHook, [1794](#)
  - CopInitHook, [1794](#)
  - CopServiceHook, [1794](#)
  - CpuUserProgramStartHook, [1795](#)
  - NvmDoneHook, [1795](#)
  - NvmEraseHook, [1795](#)
  - NvmInitHook, [1796](#)
  - NvmReinitHook, [1796](#)
  - NvmWriteHook, [1796](#)
  - UsbConnectHook, [1797](#)
  - UsbEnterLowPowerModeHook, [1797](#)
  - UsbLeaveLowPowerModeHook, [1797](#)

- XcpGetSeedHook, [1797](#)
- XcpVerifyKeyHook, [1798](#)
- ARMCM3\_STM32F1\_Olimex\_STM32H103\_Keil/↔
  - Boot/led.c
  - LedBlinkExit, [2223](#)
  - LedBlinkInit, [2223](#)
  - LedBlinkTask, [2224](#)
- ARMCM3\_STM32F1\_Olimex\_STM32H103\_Keil/↔
  - Boot/led.h
  - LedBlinkExit, [2458](#)
  - LedBlinkInit, [2459](#)
  - LedBlinkTask, [2459](#)
- ARMCM3\_STM32F1\_Olimex\_STM32H103\_Keil/↔
  - Boot/main.c
  - HAL\_MspDeInit, [2713](#)
  - HAL\_MspInit, [2713](#)
  - Init, [2713](#)
  - main, [2714](#)
  - SystemClock\_Config, [2714](#)
- ARMCM3\_STM32F1\_Olimex\_STM32H103\_Keil/↔
  - Prog/led.c
  - LedInit, [2225](#)
  - LedToggle, [2225](#)
- ARMCM3\_STM32F1\_Olimex\_STM32H103\_Keil/↔
  - Prog/led.h
  - LedInit, [2460](#)
  - LedToggle, [2460](#)
- ARMCM3\_STM32F1\_Olimex\_STM32H103\_Keil/↔
  - Prog/main.c
  - HAL\_MspDeInit, [2715](#)
  - HAL\_MspInit, [2716](#)
  - Init, [2716](#)
  - main, [2716](#)
  - SystemClock\_Config, [2716](#)
  - VectorBase\_Config, [2717](#)
- ARMCM3\_STM32F1\_Olimex\_STM32P103\_CubeID↔
  - E/Boot/App/app.c
  - AppInit, [403](#)
  - AppTask, [403](#)
- ARMCM3\_STM32F1\_Olimex\_STM32P103\_CubeID↔
  - E/Boot/App/app.h
  - AppInit, [446](#)
  - AppTask, [446](#)
- ARMCM3\_STM32F1\_Olimex\_STM32P103\_CubeID↔
  - E/Boot/App/hooks.c
  - BackDoorEntryHook, [1800](#)
  - BackDoorInitHook, [1800](#)
  - canUse, [1806](#)
  - CopInitHook, [1801](#)
  - CopServiceHook, [1801](#)
  - CpuUserProgramStartHook, [1801](#)
  - FileFirmwareUpdateCompletedHook, [1801](#)
  - FileFirmwareUpdateErrorHook, [1802](#)
  - FileFirmwareUpdateLogHook, [1802](#)
  - FileFirmwareUpdateStartedHook, [1802](#)
  - FileGetFirmwareFilenameHook, [1803](#)
  - FileFirmwareUpdateRequestedHook, [1803](#)
  - handle, [1806](#)
- NvmDoneHook, [1803](#)
- NvmEraseHook, [1803](#)
- NvmInitHook, [1804](#)
- NvmReinitHook, [1804](#)
- NvmWriteHook, [1804](#)
- XcpGetSeedHook, [1805](#)
- XcpVerifyKeyHook, [1805](#)
- ARMCM3\_STM32F1\_Olimex\_STM32P103\_CubeID↔
  - E/Boot/App/led.c
  - LedBlinkExit, [2226](#)
  - LedBlinkInit, [2226](#)
  - LedBlinkTask, [2227](#)
- ARMCM3\_STM32F1\_Olimex\_STM32P103\_CubeID↔
  - E/Boot/App/led.h
  - LedBlinkExit, [2461](#)
  - LedBlinkInit, [2462](#)
  - LedBlinkTask, [2462](#)
- ARMCM3\_STM32F1\_Olimex\_STM32P103\_CubeID↔
  - E/Prog/App/app.c
  - AppInit, [404](#)
  - AppTask, [405](#)
- ARMCM3\_STM32F1\_Olimex\_STM32P103\_CubeID↔
  - E/Prog/App/app.h
  - AppInit, [448](#)
  - AppTask, [448](#)
- ARMCM3\_STM32F1\_Olimex\_STM32P103\_CubeID↔
  - E/Prog/App/led.c
  - LedInit, [2228](#)
  - LedToggle, [2228](#)
- ARMCM3\_STM32F1\_Olimex\_STM32P103\_CubeID↔
  - E/Prog/App/led.h
  - LedInit, [2463](#)
  - LedToggle, [2463](#)
- ARMCM3\_STM32F1\_Olimex\_STM32P103\_GCC/↔
  - Boot/hooks.c
  - BackDoorEntryHook, [1808](#)
  - BackDoorInitHook, [1808](#)
  - canUse, [1813](#)
  - CopInitHook, [1808](#)
  - CopServiceHook, [1809](#)
  - CpuUserProgramStartHook, [1809](#)
  - FileFirmwareUpdateCompletedHook, [1809](#)
  - FileFirmwareUpdateErrorHook, [1809](#)
  - FileFirmwareUpdateLogHook, [1810](#)
  - FileFirmwareUpdateStartedHook, [1810](#)
  - FileGetFirmwareFilenameHook, [1810](#)
  - FileFirmwareUpdateRequestedHook, [1810](#)
  - handle, [1813](#)
  - NvmDoneHook, [1811](#)
  - NvmEraseHook, [1811](#)
  - NvmInitHook, [1811](#)
  - NvmReinitHook, [1812](#)
  - NvmWriteHook, [1812](#)
  - XcpGetSeedHook, [1812](#)
  - XcpVerifyKeyHook, [1813](#)
- ARMCM3\_STM32F1\_Olimex\_STM32P103\_GCC/↔
  - Boot/led.c
  - LedBlinkExit, [2229](#)

- LedBlinkInit, [2229](#)
- LedBlinkTask, [2230](#)
- ARMCM3\_STM32F1\_Olimex\_STM32P103\_GCC/↔
  - Boot/led.h
  - LedBlinkExit, [2464](#)
  - LedBlinkInit, [2465](#)
  - LedBlinkTask, [2465](#)
- ARMCM3\_STM32F1\_Olimex\_STM32P103\_GCC/↔
  - Boot/main.c
  - HAL\_MspDeInit, [2718](#)
  - HAL\_MspInit, [2718](#)
  - Init, [2718](#)
  - main, [2719](#)
  - SystemClock\_Config, [2719](#)
- ARMCM3\_STM32F1\_Olimex\_STM32P103\_GCC/↔
  - Prog/led.c
  - LedInit, [2231](#)
  - LedToggle, [2231](#)
- ARMCM3\_STM32F1\_Olimex\_STM32P103\_GCC/↔
  - Prog/led.h
  - LedInit, [2466](#)
  - LedToggle, [2466](#)
- ARMCM3\_STM32F1\_Olimex\_STM32P103\_GCC/↔
  - Prog/main.c
  - HAL\_MspDeInit, [2720](#)
  - HAL\_MspInit, [2721](#)
  - Init, [2721](#)
  - main, [2721](#)
  - SystemClock\_Config, [2721](#)
  - VectorBase\_Config, [2722](#)
- ARMCM3\_STM32F1\_Olimex\_STM32P103\_IAR/↔
  - Boot/hooks.c
  - BackDoorEntryHook, [1815](#)
  - BackDoorInitHook, [1816](#)
  - canUse, [1821](#)
  - CopInitHook, [1816](#)
  - CopServiceHook, [1816](#)
  - CpuUserProgramStartHook, [1816](#)
  - FileFirmwareUpdateCompletedHook, [1817](#)
  - FileFirmwareUpdateErrorHook, [1817](#)
  - FileFirmwareUpdateLogHook, [1817](#)
  - FileFirmwareUpdateStartedHook, [1818](#)
  - FileGetFirmwareFilenameHook, [1818](#)
  - FileFirmwareUpdateRequestedHook, [1818](#)
  - handle, [1821](#)
  - NvmDoneHook, [1818](#)
  - NvmEraseHook, [1819](#)
  - NvmInitHook, [1819](#)
  - NvmReinitHook, [1819](#)
  - NvmWriteHook, [1820](#)
  - XcpGetSeedHook, [1820](#)
  - XcpVerifyKeyHook, [1821](#)
- ARMCM3\_STM32F1\_Olimex\_STM32P103\_IAR/↔
  - Boot/led.c
  - LedBlinkExit, [2232](#)
  - LedBlinkInit, [2232](#)
  - LedBlinkTask, [2233](#)
- ARMCM3\_STM32F1\_Olimex\_STM32P103\_IAR/↔
  - Boot/led.h
  - LedBlinkExit, [2467](#)
  - LedBlinkInit, [2468](#)
  - LedBlinkTask, [2468](#)
- ARMCM3\_STM32F1\_Olimex\_STM32P103\_IAR/↔
  - Boot/main.c
  - HAL\_MspDeInit, [2723](#)
  - HAL\_MspInit, [2723](#)
  - Init, [2723](#)
  - main, [2724](#)
  - SystemClock\_Config, [2724](#)
- ARMCM3\_STM32F1\_Olimex\_STM32P103\_IAR/↔
  - Prog/led.c
  - LedInit, [2234](#)
  - LedToggle, [2234](#)
- ARMCM3\_STM32F1\_Olimex\_STM32P103\_IAR/↔
  - Prog/led.h
  - LedInit, [2469](#)
  - LedToggle, [2469](#)
- ARMCM3\_STM32F1\_Olimex\_STM32P103\_IAR/↔
  - Prog/main.c
  - HAL\_MspDeInit, [2725](#)
  - HAL\_MspInit, [2726](#)
  - Init, [2726](#)
  - main, [2726](#)
  - SystemClock\_Config, [2726](#)
  - VectorBase\_Config, [2727](#)
- ARMCM3\_STM32F1\_Olimex\_STM32P103\_Keil/↔
  - Boot/hooks.c
  - BackDoorEntryHook, [1823](#)
  - BackDoorInitHook, [1823](#)
  - canUse, [1829](#)
  - CopInitHook, [1824](#)
  - CopServiceHook, [1824](#)
  - CpuUserProgramStartHook, [1824](#)
  - FileFirmwareUpdateCompletedHook, [1824](#)
  - FileFirmwareUpdateErrorHook, [1825](#)
  - FileFirmwareUpdateLogHook, [1825](#)
  - FileFirmwareUpdateStartedHook, [1825](#)
  - FileGetFirmwareFilenameHook, [1826](#)
  - FileFirmwareUpdateRequestedHook, [1826](#)
  - handle, [1829](#)
  - NvmDoneHook, [1826](#)
  - NvmEraseHook, [1826](#)
  - NvmInitHook, [1827](#)
  - NvmReinitHook, [1827](#)
  - NvmWriteHook, [1827](#)
  - XcpGetSeedHook, [1828](#)
  - XcpVerifyKeyHook, [1828](#)
- ARMCM3\_STM32F1\_Olimex\_STM32P103\_Keil/↔
  - Boot/led.c
  - LedBlinkExit, [2235](#)
  - LedBlinkInit, [2235](#)
  - LedBlinkTask, [2236](#)
- ARMCM3\_STM32F1\_Olimex\_STM32P103\_Keil/↔
  - Boot/led.h
  - LedBlinkExit, [2470](#)

- LedBlinkInit, [2471](#)
- LedBlinkTask, [2471](#)
- ARMCM3\_STM32F1\_Olimex\_STM32P103\_Keil/↔
  - Boot/main.c
  - HAL\_MspDelInit, [2728](#)
  - HAL\_MsplInit, [2728](#)
  - Init, [2728](#)
  - main, [2729](#)
  - SystemClock\_Config, [2729](#)
- ARMCM3\_STM32F1\_Olimex\_STM32P103\_Keil/↔
  - Prog/led.c
  - LedInit, [2237](#)
  - LedToggle, [2237](#)
- ARMCM3\_STM32F1\_Olimex\_STM32P103\_Keil/↔
  - Prog/led.h
  - LedInit, [2472](#)
  - LedToggle, [2472](#)
- ARMCM3\_STM32F1\_Olimex\_STM32P103\_Keil/↔
  - Prog/main.c
  - HAL\_MspDelInit, [2730](#)
  - HAL\_MsplInit, [2731](#)
  - Init, [2731](#)
  - main, [2731](#)
  - SystemClock\_Config, [2731](#)
  - VectorBase\_Config, [2732](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_CubeIDE/↔
  - Boot/App/app.c
  - AppInit, [406](#)
  - AppTask, [406](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_CubeIDE/↔
  - Boot/App/app.h
  - AppInit, [449](#)
  - AppTask, [449](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_CubeIDE/↔
  - Boot/App/flash\_layout.c
  - flashLayout, [1527](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_CubeIDE/↔
  - Boot/App/hooks.c
  - BackDoorEntryHook, [1831](#)
  - BackDoorInitHook, [1831](#)
  - canUse, [1837](#)
  - CopInitHook, [1831](#)
  - CopServiceHook, [1832](#)
  - CpuUserProgramStartHook, [1832](#)
  - FileFirmwareUpdateCompletedHook, [1832](#)
  - FileFirmwareUpdateErrorHook, [1832](#)
  - FileFirmwareUpdateLogHook, [1833](#)
  - FileFirmwareUpdateStartedHook, [1833](#)
  - FileGetFirmwareFilenameHook, [1833](#)
  - FileFirmwareUpdateRequestedHook, [1833](#)
  - handle, [1837](#)
  - NvmDoneHook, [1834](#)
  - NvmEraseHook, [1834](#)
  - NvmInitHook, [1834](#)
  - NvmReinitHook, [1835](#)
  - NvmWriteHook, [1835](#)
  - UsbConnectHook, [1835](#)
  - UsbEnterLowPowerModeHook, [1836](#)
- UsbLeaveLowPowerModeHook, [1836](#)
- XcpGetSeedHook, [1836](#)
- XcpVerifyKeyHook, [1837](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_CubeIDE/↔
  - Boot/App/led.c
  - LedBlinkExit, [2238](#)
  - LedBlinkInit, [2238](#)
  - LedBlinkTask, [2239](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_CubeIDE/↔
  - Boot/App/led.h
  - LedBlinkExit, [2473](#)
  - LedBlinkInit, [2474](#)
  - LedBlinkTask, [2474](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_CubeIDE/↔
  - Prog/App/app.c
  - AppInit, [407](#)
  - AppTask, [407](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_CubeIDE/↔
  - Prog/App/app.h
  - AppInit, [450](#)
  - AppTask, [450](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_CubeIDE/↔
  - Prog/App/led.c
  - LedInit, [2240](#)
  - LedToggle, [2240](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_CubeIDE/↔
  - Prog/App/led.h
  - LedInit, [2475](#)
  - LedToggle, [2475](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_GCC/↔
  - Boot/flash\_layout.c
  - flashLayout, [1528](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_GCC/↔
  - Boot/hooks.c
  - BackDoorEntryHook, [1839](#)
  - BackDoorInitHook, [1840](#)
  - canUse, [1846](#)
  - CopInitHook, [1840](#)
  - CopServiceHook, [1840](#)
  - CpuUserProgramStartHook, [1840](#)
  - FileFirmwareUpdateCompletedHook, [1841](#)
  - FileFirmwareUpdateErrorHook, [1841](#)
  - FileFirmwareUpdateLogHook, [1841](#)
  - FileFirmwareUpdateStartedHook, [1842](#)
  - FileGetFirmwareFilenameHook, [1842](#)
  - FileFirmwareUpdateRequestedHook, [1842](#)
  - handle, [1846](#)
  - NvmDoneHook, [1842](#)
  - NvmEraseHook, [1843](#)
  - NvmInitHook, [1843](#)
  - NvmReinitHook, [1843](#)
  - NvmWriteHook, [1844](#)
  - UsbConnectHook, [1844](#)
  - UsbEnterLowPowerModeHook, [1845](#)
  - UsbLeaveLowPowerModeHook, [1845](#)
  - XcpGetSeedHook, [1845](#)
  - XcpVerifyKeyHook, [1846](#)

- ARMCM3\_STM32F1\_Olimexino\_STM32\_GCC/↔
  - Boot/led.c
  - LedBlinkExit, [2241](#)
  - LedBlinkInit, [2241](#)
  - LedBlinkTask, [2242](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_GCC/↔
  - Boot/led.h
  - LedBlinkExit, [2476](#)
  - LedBlinkInit, [2477](#)
  - LedBlinkTask, [2477](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_GCC/↔
  - Boot/main.c
  - HAL\_MspDeInit, [2733](#)
  - HAL\_MspInit, [2733](#)
  - Init, [2733](#)
  - main, [2734](#)
  - SystemClock\_Config, [2734](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_GCC/↔
  - Prog/led.c
  - LedInit, [2243](#)
  - LedToggle, [2243](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_GCC/↔
  - Prog/led.h
  - LedInit, [2478](#)
  - LedToggle, [2478](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_GCC/↔
  - Prog/main.c
  - HAL\_MspDeInit, [2735](#)
  - HAL\_MspInit, [2736](#)
  - Init, [2736](#)
  - main, [2736](#)
  - SystemClock\_Config, [2736](#)
  - VectorBase\_Config, [2737](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_IAR/↔
  - Boot/flash\_layout.c
  - flashLayout, [1529](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_IAR/↔
  - Boot/hooks.c
  - BackDoorEntryHook, [1848](#)
  - BackDoorInitHook, [1849](#)
  - canUse, [1855](#)
  - CopInitHook, [1849](#)
  - CopServiceHook, [1849](#)
  - CpuUserProgramStartHook, [1849](#)
  - FileFirmwareUpdateCompletedHook, [1850](#)
  - FileFirmwareUpdateErrorHook, [1850](#)
  - FileFirmwareUpdateLogHook, [1850](#)
  - FileFirmwareUpdateStartedHook, [1851](#)
  - FileGetFirmwareFilenameHook, [1851](#)
  - FileIsFirmwareUpdateRequestedHook, [1851](#)
  - handle, [1855](#)
  - NvmDoneHook, [1851](#)
  - NvmEraseHook, [1852](#)
  - NvmInitHook, [1852](#)
  - NvmReinitHook, [1852](#)
  - NvmWriteHook, [1853](#)
  - UsbConnectHook, [1853](#)
  - UsbEnterLowPowerModeHook, [1854](#)
  - UsbLeaveLowPowerModeHook, [1854](#)
  - XcpGetSeedHook, [1854](#)
  - XcpVerifyKeyHook, [1855](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_IAR/↔
  - Boot/led.c
  - LedBlinkExit, [2244](#)
  - LedBlinkInit, [2244](#)
  - LedBlinkTask, [2245](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_IAR/↔
  - Boot/led.h
  - LedBlinkExit, [2479](#)
  - LedBlinkInit, [2480](#)
  - LedBlinkTask, [2480](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_IAR/↔
  - Boot/main.c
  - HAL\_MspDeInit, [2738](#)
  - HAL\_MspInit, [2738](#)
  - Init, [2738](#)
  - main, [2739](#)
  - SystemClock\_Config, [2739](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_IAR/↔
  - Prog/led.c
  - LedInit, [2246](#)
  - LedToggle, [2246](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_IAR/↔
  - Prog/led.h
  - LedInit, [2481](#)
  - LedToggle, [2481](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_IAR/↔
  - Prog/main.c
  - HAL\_MspDeInit, [2740](#)
  - HAL\_MspInit, [2741](#)
  - Init, [2741](#)
  - main, [2741](#)
  - SystemClock\_Config, [2741](#)
  - VectorBase\_Config, [2742](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_Keil/↔
  - Boot/flash\_layout.c
  - flashLayout, [1530](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_Keil/↔
  - Boot/hooks.c
  - BackDoorEntryHook, [1857](#)
  - BackDoorInitHook, [1858](#)
  - canUse, [1864](#)
  - CopInitHook, [1858](#)
  - CopServiceHook, [1858](#)
  - CpuUserProgramStartHook, [1858](#)
  - FileFirmwareUpdateCompletedHook, [1859](#)
  - FileFirmwareUpdateErrorHook, [1859](#)
  - FileFirmwareUpdateLogHook, [1859](#)
  - FileFirmwareUpdateStartedHook, [1860](#)
  - FileGetFirmwareFilenameHook, [1860](#)
  - FileIsFirmwareUpdateRequestedHook, [1860](#)
  - handle, [1864](#)
  - NvmDoneHook, [1860](#)
  - NvmEraseHook, [1861](#)
  - NvmInitHook, [1861](#)
  - NvmReinitHook, [1861](#)



- NvmWriteHook, [1862](#)
- UsbConnectHook, [1862](#)
- UsbEnterLowPowerModeHook, [1863](#)
- UsbLeaveLowPowerModeHook, [1863](#)
- XcpGetSeedHook, [1863](#)
- XcpVerifyKeyHook, [1864](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_Keil/↔
  - Boot/led.c
  - LedBlinkExit, [2247](#)
  - LedBlinkInit, [2247](#)
  - LedBlinkTask, [2248](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_Keil/↔
  - Boot/led.h
  - LedBlinkExit, [2482](#)
  - LedBlinkInit, [2483](#)
  - LedBlinkTask, [2483](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_Keil/↔
  - Boot/main.c
  - HAL\_MspDeInit, [2743](#)
  - HAL\_MspInit, [2743](#)
  - Init, [2743](#)
  - main, [2744](#)
  - SystemClock\_Config, [2744](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_Keil/↔
  - Prog/led.c
  - LedInit, [2249](#)
  - LedToggle, [2249](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_Keil/↔
  - Prog/led.h
  - LedInit, [2484](#)
  - LedToggle, [2484](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_Keil/↔
  - Prog/main.c
  - HAL\_MspDeInit, [2745](#)
  - HAL\_MspInit, [2746](#)
  - Init, [2746](#)
  - main, [2746](#)
  - SystemClock\_Config, [2746](#)
  - VectorBase\_Config, [2747](#)
- ARMCM3\_STM32F2/GCC/cpu\_comp.c
  - CpulrqDisable, [1217](#)
  - CpulrqEnable, [1218](#)
- ARMCM3\_STM32F2/IAR/cpu\_comp.c
  - CpulrqDisable, [1219](#)
  - CpulrqEnable, [1219](#)
- ARMCM3\_STM32F2/Keil/cpu\_comp.c
  - CpulrqDisable, [1220](#)
  - CpulrqEnable, [1220](#)
- ARMCM3\_STM32F2/can.c
  - CanGetSpeedConfig, [1088](#)
  - CanInit, [1088](#)
  - CanReceivePacket, [1088](#)
  - canTiming, [1089](#)
  - CanTransmitPacket, [1089](#)
- ARMCM3\_STM32F2/cpu.c
  - CpulInit, [1159](#)
  - CpuMemCopy, [1159](#)
  - CpuMemSet, [1159](#)
  - CpuStartUserProgram, [1160](#)
  - CpuUserProgramStartHook, [1160](#)
- ARMCM3\_STM32F2/flash.c
  - blockInfo, [1352](#)
  - bootBlockInfo, [1352](#)
  - FlashAddToBlock, [1347](#)
  - FlashDone, [1348](#)
  - FlashErase, [1348](#)
  - FlashEraseSectors, [1348](#)
  - FlashGetSector, [1349](#)
  - FlashGetUserProgBaseAddress, [1349](#)
  - FlashInit, [1349](#)
  - FlashInitBlock, [1350](#)
  - flashLayout, [1353](#)
  - FlashReinit, [1350](#)
  - FlashSwitchBlock, [1350](#)
  - FlashVerifyChecksum, [1351](#)
  - FlashWrite, [1351](#)
  - FlashWriteBlock, [1351](#)
  - FlashWriteChecksum, [1352](#)
- ARMCM3\_STM32F2/flash.h
  - FlashDone, [1466](#)
  - FlashErase, [1466](#)
  - FlashGetUserProgBaseAddress, [1467](#)
  - FlashInit, [1467](#)
  - FlashReinit, [1467](#)
  - FlashVerifyChecksum, [1468](#)
  - FlashWrite, [1468](#)
  - FlashWriteChecksum, [1468](#)
- ARMCM3\_STM32F2/nvm.c
  - NvmDone, [3006](#)
  - NvmDoneHook, [3006](#)
  - NvmErase, [3006](#)
  - NvmEraseHook, [3007](#)
  - NvmGetUserProgBaseAddress, [3007](#)
  - NvmInit, [3007](#)
  - NvmInitHook, [3008](#)
  - NvmReinit, [3008](#)
  - NvmReinitHook, [3008](#)
  - NvmVerifyChecksum, [3008](#)
  - NvmWrite, [3009](#)
  - NvmWriteHook, [3009](#)
- ARMCM3\_STM32F2/types.h
  - blt\_addr, [3548](#)
  - blt\_bool, [3548](#)
  - blt\_char, [3548](#)
  - blt\_int16s, [3548](#)
  - blt\_int16u, [3549](#)
  - blt\_int32s, [3549](#)
  - blt\_int32u, [3549](#)
  - blt\_int8s, [3549](#)
  - blt\_int8u, [3549](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_CubeID↔
  - E/Boot/App/app.c
  - AppInit, [408](#)
  - AppTask, [409](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_CubeID↔
  - E/Boot/App/app.h

- AppInit, [451](#)
- AppTask, [451](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_CubeID↔
  - E/Boot/App/hooks.c
  - BackDoorEntryHook, [1866](#)
  - BackDoorInitHook, [1866](#)
  - canUse, [1872](#)
  - CopInitHook, [1867](#)
  - CopServiceHook, [1867](#)
  - CpuUserProgramStartHook, [1867](#)
  - FileFirmwareUpdateCompletedHook, [1867](#)
  - FileFirmwareUpdateErrorHook, [1868](#)
  - FileFirmwareUpdateLogHook, [1868](#)
  - FileFirmwareUpdateStartedHook, [1868](#)
  - FileGetFirmwareFilenameHook, [1869](#)
  - FileIsFirmwareUpdateRequestedHook, [1869](#)
  - handle, [1872](#)
  - NvmDoneHook, [1869](#)
  - NvmEraseHook, [1869](#)
  - NvmInitHook, [1870](#)
  - NvmReinitHook, [1870](#)
  - NvmWriteHook, [1870](#)
  - XcpGetSeedHook, [1871](#)
  - XcpVerifyKeyHook, [1871](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_CubeID↔
  - E/Boot/App/led.c
  - LedBlinkExit, [2250](#)
  - LedBlinkInit, [2250](#)
  - LedBlinkTask, [2251](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_CubeID↔
  - E/Boot/App/led.h
  - LedBlinkExit, [2485](#)
  - LedBlinkInit, [2486](#)
  - LedBlinkTask, [2486](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_CubeID↔
  - E/Prog/App/app.c
  - AppInit, [410](#)
  - AppTask, [410](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_CubeID↔
  - E/Prog/App/app.h
  - AppInit, [453](#)
  - AppTask, [453](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_CubeID↔
  - E/Prog/App/led.c
  - LedInit, [2252](#)
  - LedToggle, [2252](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_CubeID↔
  - E/Prog/App/led.h
  - LedInit, [2487](#)
  - LedToggle, [2487](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_GCC/↔
  - Boot/hooks.c
  - BackDoorEntryHook, [1874](#)
  - BackDoorInitHook, [1874](#)
  - canUse, [1879](#)
  - CopInitHook, [1874](#)
  - CopServiceHook, [1875](#)
  - CpuUserProgramStartHook, [1875](#)
  - FileFirmwareUpdateCompletedHook, [1875](#)
  - FileFirmwareUpdateErrorHook, [1875](#)
  - FileFirmwareUpdateLogHook, [1876](#)
  - FileFirmwareUpdateStartedHook, [1876](#)
  - FileGetFirmwareFilenameHook, [1876](#)
  - FileIsFirmwareUpdateRequestedHook, [1876](#)
  - handle, [1879](#)
  - NvmDoneHook, [1877](#)
  - NvmEraseHook, [1877](#)
  - NvmInitHook, [1877](#)
  - NvmReinitHook, [1878](#)
  - NvmWriteHook, [1878](#)
  - XcpGetSeedHook, [1878](#)
  - XcpVerifyKeyHook, [1879](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_GCC/↔
  - Boot/led.c
  - LedBlinkExit, [2253](#)
  - LedBlinkInit, [2253](#)
  - LedBlinkTask, [2254](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_GCC/↔
  - Boot/led.h
  - LedBlinkExit, [2488](#)
  - LedBlinkInit, [2489](#)
  - LedBlinkTask, [2489](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_GCC/↔
  - Boot/main.c
  - HAL\_MspDeInit, [2748](#)
  - HAL\_MspInit, [2748](#)
  - Init, [2748](#)
  - main, [2749](#)
  - SystemClock\_Config, [2749](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_GCC/↔
  - Prog/led.c
  - LedInit, [2255](#)
  - LedToggle, [2255](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_GCC/↔
  - Prog/led.h
  - LedInit, [2490](#)
  - LedToggle, [2490](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_GCC/↔
  - Prog/main.c
  - HAL\_MspDeInit, [2750](#)
  - HAL\_MspInit, [2751](#)
  - Init, [2751](#)
  - main, [2751](#)
  - SystemClock\_Config, [2751](#)
  - VectorBase\_Config, [2752](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_IAR/↔
  - Boot/hooks.c
  - BackDoorEntryHook, [1881](#)
  - BackDoorInitHook, [1882](#)
  - canUse, [1887](#)
  - CopInitHook, [1882](#)
  - CopServiceHook, [1882](#)
  - CpuUserProgramStartHook, [1882](#)
  - FileFirmwareUpdateCompletedHook, [1883](#)
  - FileFirmwareUpdateErrorHook, [1883](#)
  - FileFirmwareUpdateLogHook, [1883](#)



- FileFirmwareUpdateStartedHook, [1884](#)
- FileGetFirmwareFilenameHook, [1884](#)
- FileIsFirmwareUpdateRequestedHook, [1884](#)
- handle, [1887](#)
- NvmDoneHook, [1884](#)
- NvmEraseHook, [1885](#)
- NvmInitHook, [1885](#)
- NvmReinitHook, [1885](#)
- NvmWriteHook, [1886](#)
- XcpGetSeedHook, [1886](#)
- XcpVerifyKeyHook, [1887](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_IAR/↔
  - Boot/led.c
  - LedBlinkExit, [2256](#)
  - LedBlinkInit, [2256](#)
  - LedBlinkTask, [2257](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_IAR/↔
  - Boot/led.h
  - LedBlinkExit, [2491](#)
  - LedBlinkInit, [2492](#)
  - LedBlinkTask, [2492](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_IAR/↔
  - Boot/main.c
  - HAL\_MspDeInit, [2753](#)
  - HAL\_MspInit, [2753](#)
  - Init, [2753](#)
  - main, [2754](#)
  - SystemClock\_Config, [2754](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_IAR/↔
  - Prog/led.c
  - LedInit, [2258](#)
  - LedToggle, [2258](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_IAR/↔
  - Prog/led.h
  - LedInit, [2493](#)
  - LedToggle, [2493](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_IAR/↔
  - Prog/main.c
  - HAL\_MspDeInit, [2755](#)
  - HAL\_MspInit, [2756](#)
  - Init, [2756](#)
  - main, [2756](#)
  - SystemClock\_Config, [2756](#)
  - VectorBase\_Config, [2757](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_Keil/↔
  - Boot/hooks.c
  - BackDoorEntryHook, [1889](#)
  - BackDoorInitHook, [1889](#)
  - canUse, [1895](#)
  - CopInitHook, [1890](#)
  - CopServiceHook, [1890](#)
  - CpuUserProgramStartHook, [1890](#)
  - FileFirmwareUpdateCompletedHook, [1890](#)
  - FileFirmwareUpdateErrorHook, [1891](#)
  - FileFirmwareUpdateLogHook, [1891](#)
  - FileFirmwareUpdateStartedHook, [1891](#)
  - FileGetFirmwareFilenameHook, [1892](#)
  - FileIsFirmwareUpdateRequestedHook, [1892](#)
  - handle, [1895](#)
  - NvmDoneHook, [1892](#)
  - NvmEraseHook, [1892](#)
  - NvmInitHook, [1893](#)
  - NvmReinitHook, [1893](#)
  - NvmWriteHook, [1893](#)
  - XcpGetSeedHook, [1894](#)
  - XcpVerifyKeyHook, [1894](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_Keil/↔
  - Boot/led.c
  - LedBlinkExit, [2259](#)
  - LedBlinkInit, [2259](#)
  - LedBlinkTask, [2260](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_Keil/↔
  - Boot/led.h
  - LedBlinkExit, [2494](#)
  - LedBlinkInit, [2495](#)
  - LedBlinkTask, [2495](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_Keil/↔
  - Boot/main.c
  - HAL\_MspDeInit, [2758](#)
  - HAL\_MspInit, [2758](#)
  - Init, [2758](#)
  - main, [2759](#)
  - SystemClock\_Config, [2759](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_Keil/↔
  - Prog/led.c
  - LedInit, [2261](#)
  - LedToggle, [2261](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_Keil/↔
  - Prog/led.h
  - LedInit, [2496](#)
  - LedToggle, [2496](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_Keil/↔
  - Prog/main.c
  - HAL\_MspDeInit, [2760](#)
  - HAL\_MspInit, [2761](#)
  - Init, [2761](#)
  - main, [2761](#)
  - SystemClock\_Config, [2761](#)
  - VectorBase\_Config, [2762](#)
- ARMCM4\_S32K14/GCC/cpu\_comp.c
  - CpulrqDisable, [1221](#)
  - CpulrqEnable, [1221](#)
- ARMCM4\_S32K14/IAR/cpu\_comp.c
  - CpulrqDisable, [1222](#)
  - CpulrqEnable, [1223](#)
- ARMCM4\_S32K14/can.c
  - CanDisabledModeEnter, [1091](#)
  - CanDisabledModeExit, [1092](#)
  - CanFreezeModeEnter, [1092](#)
  - CanFreezeModeExit, [1092](#)
  - CanGetSpeedConfig, [1093](#)
  - CanInit, [1093](#)
  - CanReceivePacket, [1093](#)
  - canTiming, [1094](#)
  - CanTransmitPacket, [1094](#)
- ARMCM4\_S32K14/cpu.c

- CpuInit, [1162](#)
- CpuMemCopy, [1162](#)
- CpuMemSet, [1162](#)
- CpuStartUserProgram, [1163](#)
- CpuUserProgramStartHook, [1163](#)
- ARMCM4\_S32K14/flash.c
  - blockInfo, [1361](#)
  - bootBlockInfo, [1361](#)
  - FlashAddToBlock, [1356](#)
  - FlashCommandSequence, [1356](#)
  - FlashDone, [1356](#)
  - FlashErase, [1357](#)
  - FlashEraseSectors, [1357](#)
  - FlashGetSectorIdx, [1358](#)
  - FlashGetUserProgBaseAddress, [1358](#)
  - FlashInit, [1358](#)
  - FlashInitBlock, [1358](#)
  - FlashReinit, [1359](#)
  - FlashSwitchBlock, [1359](#)
  - FlashVerifyChecksum, [1359](#)
  - FlashWrite, [1360](#)
  - FlashWriteBlock, [1360](#)
  - FlashWriteChecksum, [1361](#)
- ARMCM4\_S32K14/flash.h
  - FlashDone, [1470](#)
  - FlashErase, [1470](#)
  - FlashGetUserProgBaseAddress, [1471](#)
  - FlashInit, [1471](#)
  - FlashReinit, [1471](#)
  - FlashVerifyChecksum, [1471](#)
  - FlashWrite, [1472](#)
  - FlashWriteChecksum, [1472](#)
- ARMCM4\_S32K14/nvm.c
  - NvmDone, [3011](#)
  - NvmDoneHook, [3011](#)
  - NvmErase, [3011](#)
  - NvmEraseHook, [3012](#)
  - NvmGetUserProgBaseAddress, [3012](#)
  - NvmInit, [3012](#)
  - NvmInitHook, [3013](#)
  - NvmReinit, [3013](#)
  - NvmReinitHook, [3013](#)
  - NvmVerifyChecksum, [3013](#)
  - NvmWrite, [3014](#)
  - NvmWriteHook, [3014](#)
- ARMCM4\_S32K14/types.h
  - blt\_addr, [3550](#)
  - blt\_bool, [3550](#)
  - blt\_char, [3550](#)
  - blt\_int16s, [3550](#)
  - blt\_int16u, [3551](#)
  - blt\_int32s, [3551](#)
  - blt\_int32u, [3551](#)
  - blt\_int8s, [3551](#)
  - blt\_int8u, [3551](#)
- ARMCM4\_S32K14\_S32K144EVB\_GCC/Boot/hooks.c
  - BackDoorEntryHook, [1896](#)
  - BackDoorInitHook, [1897](#)
- CopInitHook, [1897](#)
- CopServiceHook, [1897](#)
- CpuUserProgramStartHook, [1897](#)
- NvmDoneHook, [1898](#)
- NvmEraseHook, [1898](#)
- NvmInitHook, [1898](#)
- NvmReinitHook, [1899](#)
- NvmWriteHook, [1899](#)
- XcpGetSeedHook, [1899](#)
- XcpVerifyKeyHook, [1900](#)
- ARMCM4\_S32K14\_S32K144EVB\_GCC/Boot/led.c
  - LedBlinkExit, [2262](#)
  - LedBlinkInit, [2262](#)
  - LedBlinkTask, [2263](#)
- ARMCM4\_S32K14\_S32K144EVB\_GCC/Boot/led.h
  - LedBlinkExit, [2497](#)
  - LedBlinkInit, [2498](#)
  - LedBlinkTask, [2498](#)
- ARMCM4\_S32K14\_S32K144EVB\_GCC/Boot/main.c
  - Init, [2763](#)
  - main, [2763](#)
  - SystemClockConfig, [2763](#)
- ARMCM4\_S32K14\_S32K144EVB\_GCC/Prog/led.c
  - LedInit, [2264](#)
  - LedToggle, [2264](#)
- ARMCM4\_S32K14\_S32K144EVB\_GCC/Prog/led.h
  - LedInit, [2499](#)
  - LedToggle, [2499](#)
- ARMCM4\_S32K14\_S32K144EVB\_GCC/Prog/main.c
  - Init, [2765](#)
  - main, [2765](#)
  - SystemClockConfig, [2765](#)
- ARMCM4\_S32K14\_S32K144EVB\_IAR/Boot/hooks.c
  - BackDoorEntryHook, [1902](#)
  - BackDoorInitHook, [1902](#)
  - CopInitHook, [1902](#)
  - CopServiceHook, [1902](#)
  - CpuUserProgramStartHook, [1903](#)
  - NvmDoneHook, [1903](#)
  - NvmEraseHook, [1903](#)
  - NvmInitHook, [1904](#)
  - NvmReinitHook, [1904](#)
  - NvmWriteHook, [1904](#)
  - XcpGetSeedHook, [1905](#)
  - XcpVerifyKeyHook, [1905](#)
- ARMCM4\_S32K14\_S32K144EVB\_IAR/Boot/led.c
  - LedBlinkExit, [2265](#)
  - LedBlinkInit, [2265](#)
  - LedBlinkTask, [2266](#)
- ARMCM4\_S32K14\_S32K144EVB\_IAR/Boot/led.h
  - LedBlinkExit, [2500](#)
  - LedBlinkInit, [2501](#)
  - LedBlinkTask, [2501](#)
- ARMCM4\_S32K14\_S32K144EVB\_IAR/Boot/main.c
  - Init, [2766](#)
  - main, [2766](#)
  - SystemClockConfig, [2767](#)
- ARMCM4\_S32K14\_S32K144EVB\_IAR/Prog/led.c

- LedInit, [2267](#)
- LedToggle, [2267](#)
- ARMCM4\_S32K14\_S32K144EVB\_IAR/Prog/led.h
  - LedInit, [2502](#)
  - LedToggle, [2502](#)
- ARMCM4\_S32K14\_S32K144EVB\_IAR/Prog/main.c
  - Init, [2768](#)
  - main, [2768](#)
  - SystemClockConfig, [2768](#)
- ARMCM4\_STM32F3/GCC/cpu\_comp.c
  - CpuIrqDisable, [1224](#)
  - CpuIrqEnable, [1224](#)
- ARMCM4\_STM32F3/IAR/cpu\_comp.c
  - CpuIrqDisable, [1225](#)
  - CpuIrqEnable, [1225](#)
- ARMCM4\_STM32F3/Keil/cpu\_comp.c
  - CpuIrqDisable, [1226](#)
  - CpuIrqEnable, [1226](#)
- ARMCM4\_STM32F3/can.c
  - CanGetSpeedConfig, [1096](#)
  - CanInit, [1097](#)
  - CanReceivePacket, [1097](#)
  - canTiming, [1098](#)
  - CanTransmitPacket, [1097](#)
- ARMCM4\_STM32F3/cpu.c
  - CpuInit, [1165](#)
  - CpuMemCopy, [1165](#)
  - CpuMemSet, [1165](#)
  - CpuStartUserProgram, [1166](#)
  - CpuUserProgramStartHook, [1166](#)
- ARMCM4\_STM32F3/flash.c
  - blockInfo, [1368](#)
  - bootBlockInfo, [1368](#)
  - FlashAddToBlock, [1364](#)
  - FlashDone, [1364](#)
  - FlashErase, [1364](#)
  - FlashGetUserProgBaseAddress, [1365](#)
  - FlashInit, [1365](#)
  - FlashInitBlock, [1365](#)
  - flashLayout, [1368](#)
  - FlashReinit, [1366](#)
  - FlashSwitchBlock, [1366](#)
  - FlashVerifyChecksum, [1366](#)
  - FlashWrite, [1367](#)
  - FlashWriteBlock, [1367](#)
  - FlashWriteChecksum, [1368](#)
- ARMCM4\_STM32F3/flash.h
  - FlashDone, [1473](#)
  - FlashErase, [1474](#)
  - FlashGetUserProgBaseAddress, [1474](#)
  - FlashInit, [1475](#)
  - FlashReinit, [1475](#)
  - FlashVerifyChecksum, [1475](#)
  - FlashWrite, [1475](#)
  - FlashWriteChecksum, [1476](#)
- ARMCM4\_STM32F3/nvm.c
  - NvmDone, [3016](#)
  - NvmDoneHook, [3016](#)
  - NvmErase, [3016](#)
  - NvmEraseHook, [3017](#)
  - NvmGetUserProgBaseAddress, [3017](#)
  - NvmInit, [3017](#)
  - NvmInitHook, [3018](#)
  - NvmReinit, [3018](#)
  - NvmReinitHook, [3018](#)
  - NvmVerifyChecksum, [3018](#)
  - NvmWrite, [3019](#)
  - NvmWriteHook, [3019](#)
- ARMCM4\_STM32F3/types.h
  - blt\_addr, [3552](#)
  - blt\_bool, [3552](#)
  - blt\_char, [3552](#)
  - blt\_int16s, [3552](#)
  - blt\_int16u, [3553](#)
  - blt\_int32s, [3553](#)
  - blt\_int32u, [3553](#)
  - blt\_int8s, [3553](#)
  - blt\_int8u, [3553](#)
- ARMCM4\_STM32F3/usb.c
  - UsbFifoMgrCreate, [3588](#)
  - UsbFifoMgrInit, [3589](#)
  - UsbFifoMgrRead, [3589](#)
  - UsbFifoMgrScan, [3589](#)
  - UsbFifoMgrWrite, [3590](#)
  - UsbFree, [3590](#)
  - UsbInit, [3590](#)
  - UsbReceiveByte, [3591](#)
  - UsbReceivePacket, [3591](#)
  - UsbReceivePipeBulkOUT, [3591](#)
  - UsbTransmitByte, [3592](#)
  - UsbTransmitPacket, [3592](#)
  - UsbTransmitPipeBulkIN, [3592](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_CubeIDE/↔
  - Boot/App/app.c
    - AppInit, [411](#)
    - AppTask, [411](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_CubeIDE/↔
  - Boot/App/app.h
    - AppInit, [454](#)
    - AppTask, [454](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_CubeIDE/↔
  - Boot/App/flash\_layout.c
    - flashLayout, [1531](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_CubeIDE/↔
  - Boot/App/hooks.c
    - BackDoorEntryHook, [1907](#)
    - BackDoorInitHook, [1907](#)
    - CopInitHook, [1907](#)
    - CopServiceHook, [1908](#)
    - CpuUserProgramStartHook, [1908](#)
    - NvmDoneHook, [1908](#)
    - NvmEraseHook, [1908](#)
    - NvmInitHook, [1909](#)
    - NvmReinitHook, [1909](#)
    - NvmWriteHook, [1909](#)
    - UsbConnectHook, [1910](#)

- UsbEnterLowPowerModeHook, [1910](#)
- UsbLeaveLowPowerModeHook, [1910](#)
- XcpGetSeedHook, [1910](#)
- XcpVerifyKeyHook, [1911](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_CubeIDE/↔
  - Boot/App/led.c
  - LedBlinkExit, [2268](#)
  - LedBlinkInit, [2268](#)
  - LedBlinkTask, [2269](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_CubeIDE/↔
  - Boot/App/led.h
  - LedBlinkExit, [2503](#)
  - LedBlinkInit, [2504](#)
  - LedBlinkTask, [2504](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_CubeIDE/↔
  - Prog/App/app.c
  - AppInit, [412](#)
  - AppTask, [412](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_CubeIDE/↔
  - Prog/App/app.h
  - AppInit, [455](#)
  - AppTask, [455](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_CubeIDE/↔
  - Prog/App/led.c
  - LedInit, [2270](#)
  - LedToggle, [2270](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_CubeIDE/↔
  - Prog/App/led.h
  - LedInit, [2505](#)
  - LedToggle, [2505](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_GCC/↔
  - Boot/flash\_layout.c
  - flashLayout, [1532](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_GCC/↔
  - Boot/hooks.c
  - BackDoorEntryHook, [1913](#)
  - BackDoorInitHook, [1913](#)
  - CopInitHook, [1913](#)
  - CopServiceHook, [1913](#)
  - CpuUserProgramStartHook, [1914](#)
  - NvmDoneHook, [1914](#)
  - NvmEraseHook, [1914](#)
  - NvmInitHook, [1915](#)
  - NvmReinitHook, [1915](#)
  - NvmWriteHook, [1915](#)
  - UsbConnectHook, [1916](#)
  - UsbEnterLowPowerModeHook, [1916](#)
  - UsbLeaveLowPowerModeHook, [1916](#)
  - XcpGetSeedHook, [1916](#)
  - XcpVerifyKeyHook, [1917](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_GCC/↔
  - Boot/led.c
  - LedBlinkExit, [2271](#)
  - LedBlinkInit, [2272](#)
  - LedBlinkTask, [2272](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_GCC/↔
  - Boot/led.h
  - LedBlinkExit, [2506](#)
- LedBlinkInit, [2507](#)
- LedBlinkTask, [2507](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_GCC/↔
  - Boot/main.c
  - HAL\_MspDeInit, [2770](#)
  - HAL\_MspInit, [2770](#)
  - Init, [2770](#)
  - main, [2771](#)
  - SystemClock\_Config, [2771](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_GCC/↔
  - Prog/led.c
  - LedInit, [2273](#)
  - LedToggle, [2273](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_GCC/↔
  - Prog/led.h
  - LedInit, [2508](#)
  - LedToggle, [2508](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_GCC/↔
  - Prog/main.c
  - HAL\_MspDeInit, [2773](#)
  - HAL\_MspInit, [2773](#)
  - Init, [2773](#)
  - main, [2773](#)
  - SystemClock\_Config, [2774](#)
  - VectorBase\_Config, [2774](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_IAR/↔
  - Boot/flash\_layout.c
  - flashLayout, [1533](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_IAR/↔
  - Boot/hooks.c
  - BackDoorEntryHook, [1919](#)
  - BackDoorInitHook, [1919](#)
  - CopInitHook, [1919](#)
  - CopServiceHook, [1919](#)
  - CpuUserProgramStartHook, [1920](#)
  - NvmDoneHook, [1920](#)
  - NvmEraseHook, [1920](#)
  - NvmInitHook, [1921](#)
  - NvmReinitHook, [1921](#)
  - NvmWriteHook, [1921](#)
  - UsbConnectHook, [1922](#)
  - UsbEnterLowPowerModeHook, [1922](#)
  - UsbLeaveLowPowerModeHook, [1922](#)
  - XcpGetSeedHook, [1922](#)
  - XcpVerifyKeyHook, [1923](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_IAR/↔
  - Boot/led.c
  - LedBlinkExit, [2275](#)
  - LedBlinkInit, [2275](#)
  - LedBlinkTask, [2275](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_IAR/↔
  - Boot/led.h
  - LedBlinkExit, [2509](#)
  - LedBlinkInit, [2510](#)
  - LedBlinkTask, [2510](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_IAR/↔
  - Boot/main.c
  - HAL\_MspDeInit, [2775](#)

- HAL\_MspInit, [2776](#)
- Init, [2776](#)
- main, [2776](#)
- SystemClock\_Config, [2776](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_IAR/↔
  - Prog/led.c
  - LedInit, [2276](#)
  - LedToggle, [2277](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_IAR/↔
  - Prog/led.h
  - LedInit, [2511](#)
  - LedToggle, [2511](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_IAR/↔
  - Prog/main.c
  - HAL\_MspDeInit, [2778](#)
  - HAL\_MspInit, [2778](#)
  - Init, [2778](#)
  - main, [2779](#)
  - SystemClock\_Config, [2779](#)
  - VectorBase\_Config, [2779](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_Keil/↔
  - Boot/flash\_layout.c
  - flashLayout, [1534](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_Keil/↔
  - Boot/hooks.c
  - BackDoorEntryHook, [1925](#)
  - BackDoorInitHook, [1925](#)
  - CopInitHook, [1925](#)
  - CopServiceHook, [1925](#)
  - CpuUserProgramStartHook, [1926](#)
  - NvmDoneHook, [1926](#)
  - NvmEraseHook, [1926](#)
  - NvmInitHook, [1927](#)
  - NvmReinitHook, [1927](#)
  - NvmWriteHook, [1927](#)
  - UsbConnectHook, [1928](#)
  - UsbEnterLowPowerModeHook, [1928](#)
  - UsbLeaveLowPowerModeHook, [1928](#)
  - XcpGetSeedHook, [1928](#)
  - XcpVerifyKeyHook, [1929](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_Keil/↔
  - Boot/led.c
  - LedBlinkExit, [2278](#)
  - LedBlinkInit, [2278](#)
  - LedBlinkTask, [2279](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_Keil/↔
  - Boot/led.h
  - LedBlinkExit, [2512](#)
  - LedBlinkInit, [2513](#)
  - LedBlinkTask, [2513](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_Keil/↔
  - Boot/main.c
  - HAL\_MspDeInit, [2781](#)
  - HAL\_MspInit, [2781](#)
  - Init, [2781](#)
  - main, [2781](#)
  - SystemClock\_Config, [2782](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_Keil/↔
  - Prog/led.c
  - LedInit, [2280](#)
  - LedToggle, [2280](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_Keil/↔
  - Prog/led.h
  - LedInit, [2514](#)
  - LedToggle, [2514](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_Keil/↔
  - Prog/main.c
  - HAL\_MspDeInit, [2783](#)
  - HAL\_MspInit, [2783](#)
  - Init, [2783](#)
  - main, [2784](#)
  - SystemClock\_Config, [2784](#)
  - VectorBase\_Config, [2784](#)
- ARMCM4\_STM32F3\_Nucleo\_F303K8\_CubeIDE/↔
  - Boot/App/app.c
  - ApplInit, [413](#)
  - AppTask, [414](#)
- ARMCM4\_STM32F3\_Nucleo\_F303K8\_CubeIDE/↔
  - Boot/App/app.h
  - ApplInit, [456](#)
  - AppTask, [456](#)
- ARMCM4\_STM32F3\_Nucleo\_F303K8\_CubeIDE/↔
  - Boot/App/hooks.c
  - BackDoorEntryHook, [1931](#)
  - BackDoorInitHook, [1931](#)
  - CopInitHook, [1931](#)
  - CopServiceHook, [1931](#)
  - CpuUserProgramStartHook, [1932](#)
  - NvmDoneHook, [1932](#)
  - NvmEraseHook, [1932](#)
  - NvmInitHook, [1933](#)
  - NvmReinitHook, [1933](#)
  - NvmWriteHook, [1933](#)
  - XcpGetSeedHook, [1934](#)
  - XcpVerifyKeyHook, [1934](#)
- ARMCM4\_STM32F3\_Nucleo\_F303K8\_CubeIDE/↔
  - Boot/App/led.c
  - LedBlinkExit, [2281](#)
  - LedBlinkInit, [2282](#)
  - LedBlinkTask, [2282](#)
- ARMCM4\_STM32F3\_Nucleo\_F303K8\_CubeIDE/↔
  - Boot/App/led.h
  - LedBlinkExit, [2515](#)
  - LedBlinkInit, [2516](#)
  - LedBlinkTask, [2516](#)
- ARMCM4\_STM32F3\_Nucleo\_F303K8\_CubeIDE/↔
  - Prog/App/app.c
  - ApplInit, [415](#)
  - AppTask, [415](#)
- ARMCM4\_STM32F3\_Nucleo\_F303K8\_CubeIDE/↔
  - Prog/App/app.h
  - ApplInit, [458](#)
  - AppTask, [458](#)
- ARMCM4\_STM32F3\_Nucleo\_F303K8\_CubeIDE/↔
  - Prog/App/led.c

- LedInit, [2283](#)
- LedToggle, [2283](#)
- ARMCM4\_STM32F3\_Nucleo\_F303K8\_CubeIDE/↔
  - Prog/App/led.h
  - LedInit, [2517](#)
  - LedToggle, [2517](#)
- ARMCM4\_STM32F3\_Nucleo\_F303K8\_GCC/Boot/hooks.↔
  - c
    - BackDoorEntryHook, [1936](#)
    - BackDoorInitHook, [1936](#)
    - CopInitHook, [1936](#)
    - CopServiceHook, [1937](#)
    - CpuUserProgramStartHook, [1937](#)
    - NvmDoneHook, [1937](#)
    - NvmEraseHook, [1937](#)
    - NvmInitHook, [1938](#)
    - NvmReinitHook, [1938](#)
    - NvmWriteHook, [1938](#)
    - XcpGetSeedHook, [1939](#)
    - XcpVerifyKeyHook, [1939](#)
- ARMCM4\_STM32F3\_Nucleo\_F303K8\_GCC/Boot/led.↔
  - c
    - LedBlinkExit, [2285](#)
    - LedBlinkInit, [2285](#)
    - LedBlinkTask, [2285](#)
- ARMCM4\_STM32F3\_Nucleo\_F303K8\_GCC/Boot/led.↔
  - h
    - LedBlinkExit, [2518](#)
    - LedBlinkInit, [2519](#)
    - LedBlinkTask, [2519](#)
- ARMCM4\_STM32F3\_Nucleo\_F303K8\_GCC/Boot/main.↔
  - c
    - HAL\_MspDelInit, [2786](#)
    - HAL\_MsplInit, [2786](#)
    - Init, [2786](#)
    - main, [2786](#)
    - SystemClock\_Config, [2787](#)
- ARMCM4\_STM32F3\_Nucleo\_F303K8\_GCC/Prog/led.↔
  - c
    - LedInit, [2286](#)
    - LedToggle, [2286](#)
- ARMCM4\_STM32F3\_Nucleo\_F303K8\_GCC/Prog/led.↔
  - h
    - LedInit, [2520](#)
    - LedToggle, [2520](#)
- ARMCM4\_STM32F3\_Nucleo\_F303K8\_GCC/Prog/main.↔
  - c
    - HAL\_MspDelInit, [2788](#)
    - HAL\_MsplInit, [2788](#)
    - Init, [2788](#)
    - main, [2789](#)
    - SystemClock\_Config, [2789](#)
    - VectorBase\_Config, [2789](#)
- ARMCM4\_STM32F3\_Nucleo\_F303K8\_IAR/Boot/hooks.↔
  - c
    - BackDoorEntryHook, [1941](#)
    - BackDoorInitHook, [1941](#)
    - CopInitHook, [1941](#)
- CopServiceHook, [1942](#)
- CpuUserProgramStartHook, [1942](#)
- NvmDoneHook, [1942](#)
- NvmEraseHook, [1942](#)
- NvmInitHook, [1943](#)
- NvmReinitHook, [1943](#)
- NvmWriteHook, [1943](#)
- XcpGetSeedHook, [1944](#)
- XcpVerifyKeyHook, [1944](#)
- ARMCM4\_STM32F3\_Nucleo\_F303K8\_IAR/Boot/led.c
  - LedBlinkExit, [2288](#)
  - LedBlinkInit, [2288](#)
  - LedBlinkTask, [2288](#)
- ARMCM4\_STM32F3\_Nucleo\_F303K8\_IAR/Boot/led.h
  - LedBlinkExit, [2521](#)
  - LedBlinkInit, [2522](#)
  - LedBlinkTask, [2522](#)
- ARMCM4\_STM32F3\_Nucleo\_F303K8\_IAR/Boot/main.↔
  - c
    - HAL\_MspDelInit, [2791](#)
    - HAL\_MsplInit, [2791](#)
    - Init, [2791](#)
    - main, [2791](#)
    - SystemClock\_Config, [2792](#)
- ARMCM4\_STM32F3\_Nucleo\_F303K8\_IAR/Prog/led.c
  - LedInit, [2289](#)
  - LedToggle, [2289](#)
- ARMCM4\_STM32F3\_Nucleo\_F303K8\_IAR/Prog/led.h
  - LedInit, [2523](#)
  - LedToggle, [2523](#)
- ARMCM4\_STM32F3\_Nucleo\_F303K8\_IAR/Prog/main.↔
  - c
    - HAL\_MspDelInit, [2793](#)
    - HAL\_MsplInit, [2793](#)
    - Init, [2793](#)
    - main, [2794](#)
    - SystemClock\_Config, [2794](#)
    - VectorBase\_Config, [2794](#)
- ARMCM4\_STM32F3\_Nucleo\_F303K8\_Keil/Boot/hooks.↔
  - c
    - BackDoorEntryHook, [1946](#)
    - BackDoorInitHook, [1946](#)
    - CopInitHook, [1946](#)
    - CopServiceHook, [1947](#)
    - CpuUserProgramStartHook, [1947](#)
    - NvmDoneHook, [1947](#)
    - NvmEraseHook, [1947](#)
    - NvmInitHook, [1948](#)
    - NvmReinitHook, [1948](#)
    - NvmWriteHook, [1948](#)
    - XcpGetSeedHook, [1949](#)
    - XcpVerifyKeyHook, [1949](#)
- ARMCM4\_STM32F3\_Nucleo\_F303K8\_Keil/Boot/led.c
  - LedBlinkExit, [2291](#)
  - LedBlinkInit, [2291](#)
  - LedBlinkTask, [2291](#)
- ARMCM4\_STM32F3\_Nucleo\_F303K8\_Keil/Boot/led.h
  - LedBlinkExit, [2524](#)



- LedBlinkInit, [2525](#)
- LedBlinkTask, [2525](#)
- ARMCM4\_STM32F3\_Nucleo\_F303K8\_Keil/Boot/main.c↔
  - HAL\_MspDeInit, [2796](#)
  - HAL\_MspInit, [2796](#)
  - Init, [2796](#)
  - main, [2796](#)
  - SystemClock\_Config, [2797](#)
- ARMCM4\_STM32F3\_Nucleo\_F303K8\_Keil/Prog/led.c
  - LedInit, [2292](#)
  - LedToggle, [2292](#)
- ARMCM4\_STM32F3\_Nucleo\_F303K8\_Keil/Prog/led.h
  - LedInit, [2526](#)
  - LedToggle, [2526](#)
- ARMCM4\_STM32F3\_Nucleo\_F303K8\_Keil/Prog/main.c↔
  - HAL\_MspDeInit, [2798](#)
  - HAL\_MspInit, [2798](#)
  - Init, [2798](#)
  - main, [2799](#)
  - SystemClock\_Config, [2799](#)
  - VectorBase\_Config, [2799](#)
- ARMCM4\_STM32F4/GCC/cpu\_comp.c
  - CpulrqDisable, [1227](#)
  - CpulrqEnable, [1228](#)
- ARMCM4\_STM32F4/IAR/cpu\_comp.c
  - CpulrqDisable, [1229](#)
  - CpulrqEnable, [1229](#)
- ARMCM4\_STM32F4/Keil/cpu\_comp.c
  - CpulrqDisable, [1230](#)
  - CpulrqEnable, [1230](#)
- ARMCM4\_STM32F4/can.c
  - CanGetSpeedConfig, [1100](#)
  - CanInit, [1100](#)
  - CanReceivePacket, [1100](#)
  - canTiming, [1101](#)
  - CanTransmitPacket, [1101](#)
- ARMCM4\_STM32F4/cpu.c
  - CpuInit, [1168](#)
  - CpuMemCopy, [1168](#)
  - CpuMemSet, [1168](#)
  - CpuStartUserProgram, [1169](#)
  - CpuUserProgramStartHook, [1169](#)
- ARMCM4\_STM32F4/flash.c
  - blockInfo, [1376](#)
  - bootBlockInfo, [1376](#)
  - FlashAddToBlock, [1371](#)
  - FlashDone, [1371](#)
  - FlashErase, [1372](#)
  - FlashEraseSectors, [1372](#)
  - FlashGetSector, [1373](#)
  - FlashGetUserProgBaseAddress, [1373](#)
  - FlashInit, [1373](#)
  - FlashInitBlock, [1373](#)
  - flashLayout, [1376](#)
  - FlashReinit, [1374](#)
  - FlashSwitchBlock, [1374](#)
  - FlashVerifyChecksum, [1374](#)
  - FlashWrite, [1375](#)
  - FlashWriteBlock, [1375](#)
  - FlashWriteChecksum, [1376](#)
- ARMCM4\_STM32F4/flash.h
  - FlashDone, [1477](#)
  - FlashErase, [1477](#)
  - FlashGetUserProgBaseAddress, [1478](#)
  - FlashInit, [1478](#)
  - FlashReinit, [1478](#)
  - FlashVerifyChecksum, [1479](#)
  - FlashWrite, [1479](#)
  - FlashWriteChecksum, [1479](#)
- ARMCM4\_STM32F4/nvm.c
  - NvmDone, [3021](#)
  - NvmDoneHook, [3021](#)
  - NvmErase, [3021](#)
  - NvmEraseHook, [3022](#)
  - NvmGetUserProgBaseAddress, [3022](#)
  - NvmlInit, [3022](#)
  - NvmlInitHook, [3023](#)
  - NvmReinit, [3023](#)
  - NvmReinitHook, [3023](#)
  - NvmVerifyChecksum, [3023](#)
  - NvmWrite, [3024](#)
  - NvmWriteHook, [3024](#)
- ARMCM4\_STM32F4/types.h
  - blt\_addr, [3554](#)
  - blt\_bool, [3554](#)
  - blt\_char, [3554](#)
  - blt\_int16s, [3554](#)
  - blt\_int16u, [3555](#)
  - blt\_int32s, [3555](#)
  - blt\_int32u, [3555](#)
  - blt\_int8s, [3555](#)
  - blt\_int8u, [3555](#)
- ARMCM4\_STM32F4/usb.c
  - UsbFifoMgrCreate, [3595](#)
  - UsbFifoMgrInit, [3595](#)
  - UsbFifoMgrRead, [3595](#)
  - UsbFifoMgrScan, [3596](#)
  - UsbFifoMgrWrite, [3596](#)
  - UsbFree, [3597](#)
  - UsbInit, [3597](#)
  - UsbReceiveByte, [3597](#)
  - UsbReceivePacket, [3597](#)
  - UsbReceivePipeBulkOUT, [3598](#)
  - UsbTransmitByte, [3598](#)
  - UsbTransmitPacket, [3598](#)
  - UsbTransmitPipeBulkIN, [3599](#)
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_CubeIDE/↔
  - Boot/App/app.c
    - AppInit, [416](#)
    - AppTask, [416](#)
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_CubeIDE/↔
  - Boot/App/app.h
    - AppInit, [459](#)
    - AppTask, [459](#)

- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_CubeIDE/↔
  - Boot/App/hooks.c
  - BackDoorEntryHook, [1951](#)
  - BackDoorInitHook, [1951](#)
  - CopInitHook, [1951](#)
  - CopServiceHook, [1952](#)
  - CpuUserProgramStartHook, [1952](#)
  - NvmDoneHook, [1952](#)
  - NvmEraseHook, [1952](#)
  - NvmlInitHook, [1953](#)
  - NvmReinitHook, [1953](#)
  - NvmWriteHook, [1953](#)
  - UsbConnectHook, [1954](#)
  - UsbEnterLowPowerModeHook, [1954](#)
  - UsbLeaveLowPowerModeHook, [1954](#)
  - XcpGetSeedHook, [1954](#)
  - XcpVerifyKeyHook, [1955](#)
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_CubeIDE/↔
  - Boot/App/led.c
  - LedBlinkExit, [2294](#)
  - LedBlinkInit, [2294](#)
  - LedBlinkTask, [2294](#)
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_CubeIDE/↔
  - Boot/App/led.h
  - LedBlinkExit, [2527](#)
  - LedBlinkInit, [2528](#)
  - LedBlinkTask, [2528](#)
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_CubeIDE/↔
  - Boot/App/shared\_params.c
  - \_\_attribute\_\_, [3085](#)
  - SharedParamsCalculateChecksum, [3085](#)
  - SharedParamsInit, [3086](#)
  - SharedParamsReadByIndex, [3086](#)
  - SharedParamsValidateBuffer, [3086](#)
  - SharedParamsVerifyChecksum, [3087](#)
  - SharedParamsWriteByIndex, [3087](#)
  - SharedParamsWriteChecksum, [3087](#)
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_CubeIDE/↔
  - Boot/App/shared\_params.h
  - SharedParamsInit, [3181](#)
  - SharedParamsReadByIndex, [3181](#)
  - SharedParamsWriteByIndex, [3181](#)
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_CubeIDE/↔
  - Prog/App/app.c
  - AppInit, [417](#)
  - AppTask, [418](#)
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_CubeIDE/↔
  - Prog/App/app.h
  - AppInit, [460](#)
  - AppTask, [460](#)
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_CubeIDE/↔
  - Prog/App/led.c
  - LedInit, [2295](#)
  - LedToggle, [2295](#)
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_CubeIDE/↔
  - Prog/App/led.h
  - LedInit, [2529](#)
  - LedToggle, [2529](#)
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_CubeIDE/↔
  - Prog/App/shared\_params.c
  - \_\_attribute\_\_, [3089](#)
  - SharedParamsCalculateChecksum, [3090](#)
  - SharedParamsInit, [3090](#)
  - SharedParamsReadByIndex, [3090](#)
  - SharedParamsValidateBuffer, [3091](#)
  - SharedParamsVerifyChecksum, [3091](#)
  - SharedParamsWriteByIndex, [3091](#)
  - SharedParamsWriteChecksum, [3092](#)
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_CubeIDE/↔
  - Prog/App/shared\_params.h
  - SharedParamsInit, [3183](#)
  - SharedParamsReadByIndex, [3183](#)
  - SharedParamsWriteByIndex, [3184](#)
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_GCC/Boot/hooks.↔
  - c
  - BackDoorEntryHook, [1957](#)
  - BackDoorInitHook, [1957](#)
  - CopInitHook, [1957](#)
  - CopServiceHook, [1958](#)
  - CpuUserProgramStartHook, [1958](#)
  - NvmDoneHook, [1958](#)
  - NvmEraseHook, [1958](#)
  - NvmlInitHook, [1959](#)
  - NvmReinitHook, [1959](#)
  - NvmWriteHook, [1959](#)
  - UsbConnectHook, [1960](#)
  - UsbEnterLowPowerModeHook, [1960](#)
  - UsbLeaveLowPowerModeHook, [1960](#)
  - XcpGetSeedHook, [1960](#)
  - XcpVerifyKeyHook, [1961](#)
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_GCC/Boot/led.h
  - LedBlinkExit, [2530](#)
  - LedBlinkInit, [2531](#)
  - LedBlinkTask, [2531](#)
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_GCC/Boot/main.↔
  - c
  - HAL\_MspDeInit, [2801](#)
  - HAL\_MspInit, [2801](#)
  - Init, [2801](#)
  - main, [2801](#)
  - SystemClock\_Config, [2802](#)
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_GCC/Boot/shared.↔
  - \_params.c
  - \_\_attribute\_\_, [3093](#)
  - SharedParamsCalculateChecksum, [3094](#)
  - SharedParamsInit, [3094](#)
  - SharedParamsReadByIndex, [3095](#)
  - SharedParamsValidateBuffer, [3095](#)
  - SharedParamsVerifyChecksum, [3095](#)
  - SharedParamsWriteByIndex, [3096](#)
  - SharedParamsWriteChecksum, [3096](#)
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_GCC/Prog/led.c
  - LedInit, [2297](#)
  - LedToggle, [2297](#)
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_GCC/Prog/main.↔
  - c



- HAL\_MspDeInit, [2803](#)
- HAL\_MspInit, [2803](#)
- Init, [2803](#)
- main, [2804](#)
- SystemClock\_Config, [2804](#)
- VectorBase\_Config, [2804](#)
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_GCC/Prog/shared↔
  - \_params.c
  - \_\_attribute\_\_, [3098](#)
  - SharedParamsCalculateChecksum, [3098](#)
  - SharedParamsInit, [3099](#)
  - SharedParamsReadByIndex, [3099](#)
  - SharedParamsValidateBuffer, [3099](#)
  - SharedParamsVerifyChecksum, [3100](#)
  - SharedParamsWriteByIndex, [3100](#)
  - SharedParamsWriteChecksum, [3100](#)
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_GCC/Prog/shared↔
  - \_params.h
  - SharedParamsInit, [3185](#)
  - SharedParamsReadByIndex, [3185](#)
  - SharedParamsWriteByIndex, [3186](#)
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_IAR/Boot/hooks.↔
  - c
  - BackDoorEntryHook, [1963](#)
  - BackDoorInitHook, [1963](#)
  - CopInitHook, [1963](#)
  - CopServiceHook, [1964](#)
  - CpuUserProgramStartHook, [1964](#)
  - NvmDoneHook, [1964](#)
  - NvmEraseHook, [1964](#)
  - NvmInitHook, [1965](#)
  - NvmReinitHook, [1965](#)
  - NvmWriteHook, [1965](#)
  - UsbConnectHook, [1966](#)
  - UsbEnterLowPowerModeHook, [1966](#)
  - UsbLeaveLowPowerModeHook, [1966](#)
  - XcpGetSeedHook, [1966](#)
  - XcpVerifyKeyHook, [1967](#)
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_IAR/Boot/led.h
  - LedBlinkExit, [2532](#)
  - LedBlinkInit, [2532](#)
  - LedBlinkTask, [2533](#)
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_IAR/Boot/main.↔
  - c
  - HAL\_MspDeInit, [2806](#)
  - HAL\_MspInit, [2806](#)
  - Init, [2806](#)
  - main, [2806](#)
  - SystemClock\_Config, [2807](#)
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_IAR/Boot/shared↔
  - \_params.c
  - shared, [3105](#)
  - SharedParamsCalculateChecksum, [3102](#)
  - SharedParamsInit, [3102](#)
  - SharedParamsReadByIndex, [3103](#)
  - SharedParamsValidateBuffer, [3103](#)
  - SharedParamsVerifyChecksum, [3103](#)
  - SharedParamsWriteByIndex, [3104](#)
- SharedParamsWriteChecksum, [3104](#)
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_IAR/Prog/led.c
  - LedInit, [2298](#)
  - LedToggle, [2298](#)
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_IAR/Prog/main.↔
  - c
  - HAL\_MspDeInit, [2808](#)
  - HAL\_MspInit, [2808](#)
  - Init, [2808](#)
  - main, [2809](#)
  - SystemClock\_Config, [2809](#)
  - VectorBase\_Config, [2809](#)
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_IAR/Prog/shared↔
  - \_params.c
  - shared, [3109](#)
  - SharedParamsCalculateChecksum, [3107](#)
  - SharedParamsInit, [3107](#)
  - SharedParamsReadByIndex, [3107](#)
  - SharedParamsValidateBuffer, [3108](#)
  - SharedParamsVerifyChecksum, [3108](#)
  - SharedParamsWriteByIndex, [3108](#)
  - SharedParamsWriteChecksum, [3109](#)
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_IAR/Prog/shared↔
  - \_params.h
  - SharedParamsInit, [3188](#)
  - SharedParamsReadByIndex, [3188](#)
  - SharedParamsWriteByIndex, [3188](#)
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_Keil/Boot/hooks.↔
  - c
  - BackDoorEntryHook, [1969](#)
  - BackDoorInitHook, [1969](#)
  - CopInitHook, [1969](#)
  - CopServiceHook, [1970](#)
  - CpuUserProgramStartHook, [1970](#)
  - NvmDoneHook, [1970](#)
  - NvmEraseHook, [1970](#)
  - NvmInitHook, [1971](#)
  - NvmReinitHook, [1971](#)
  - NvmWriteHook, [1971](#)
  - UsbConnectHook, [1972](#)
  - UsbEnterLowPowerModeHook, [1972](#)
  - UsbLeaveLowPowerModeHook, [1972](#)
  - XcpGetSeedHook, [1972](#)
  - XcpVerifyKeyHook, [1973](#)
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_Keil/Boot/led.h
  - LedBlinkExit, [2534](#)
  - LedBlinkInit, [2534](#)
  - LedBlinkTask, [2534](#)
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_Keil/Boot/main.↔
  - c
  - HAL\_MspDeInit, [2811](#)
  - HAL\_MspInit, [2811](#)
  - Init, [2811](#)
  - main, [2811](#)
  - SystemClock\_Config, [2812](#)
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_Keil/Boot/shared↔
  - \_params.c
  - \_\_attribute\_\_, [3111](#)

- SharedParamsCalculateChecksum, 3111
- SharedParamsInit, 3112
- SharedParamsReadByIndex, 3112
- SharedParamsValidateBuffer, 3112
- SharedParamsVerifyChecksum, 3113
- SharedParamsWriteByIndex, 3113
- SharedParamsWriteChecksum, 3113
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_Keil/Prog/led.c
  - LedInit, 2299
  - LedToggle, 2299
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_Keil/Prog/main.c↔
  - HAL\_MspDelInit, 2813
  - HAL\_MsplInit, 2813
  - Init, 2813
  - main, 2814
  - SystemClock\_Config, 2814
  - VectorBase\_Config, 2814
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_Keil/Prog/shared↔\_params.c
  - \_\_attribute\_\_, 3115
  - SharedParamsCalculateChecksum, 3116
  - SharedParamsInit, 3116
  - SharedParamsReadByIndex, 3116
  - SharedParamsValidateBuffer, 3117
  - SharedParamsVerifyChecksum, 3117
  - SharedParamsWriteByIndex, 3117
  - SharedParamsWriteChecksum, 3118
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_Keil/Prog/shared↔\_params.h
  - SharedParamsInit, 3190
  - SharedParamsReadByIndex, 3190
  - SharedParamsWriteByIndex, 3191
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_CubeID↔E/Boot/App/app.c
  - AppInit, 419
  - AppTask, 419
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_CubeID↔E/Boot/App/app.h
  - AppInit, 461
  - AppTask, 461
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_CubeID↔E/Boot/App/hooks.c
  - BackDoorEntryHook, 1975
  - BackDoorInitHook, 1975
  - canUse, 1981
  - CopInitHook, 1976
  - CopServiceHook, 1976
  - CpuUserProgramStartHook, 1976
  - FileFirmwareUpdateCompletedHook, 1976
  - FileFirmwareUpdateErrorHook, 1977
  - FileFirmwareUpdateLogHook, 1977
  - FileFirmwareUpdateStartedHook, 1977
  - FileGetFirmwareFilenameHook, 1978
  - FileFirmwareUpdateRequestedHook, 1978
  - handle, 1981
  - NvmDoneHook, 1978
  - NvmEraseHook, 1978
  - NvmInitHook, 1979
  - NvmReinitHook, 1979
  - NvmWriteHook, 1979
  - UsbConnectHook, 1980
  - UsbEnterLowPowerModeHook, 1980
  - UsbLeaveLowPowerModeHook, 1980
  - XcpGetSeedHook, 1980
  - XcpVerifyKeyHook, 1981
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_CubeID↔E/Boot/App/led.c
  - LedBlinkExit, 2301
  - LedBlinkInit, 2301
  - LedBlinkTask, 2301
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_CubeID↔E/Boot/App/led.h
  - LedBlinkExit, 2535
  - LedBlinkInit, 2535
  - LedBlinkTask, 2536
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_CubeID↔E/Prog/App/app.c
  - AppInit, 420
  - AppTask, 420
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_CubeID↔E/Prog/App/app.h
  - AppInit, 463
  - AppTask, 463
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_CubeID↔E/Prog/App/led.c
  - LedInit, 2302
  - LedToggle, 2302
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_CubeID↔E/Prog/App/led.h
  - LedInit, 2537
  - LedToggle, 2537
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_GCC↔Boot/hooks.c
  - BackDoorEntryHook, 1984
  - BackDoorInitHook, 1984
  - canUse, 1990
  - CopInitHook, 1984
  - CopServiceHook, 1984
  - CpuUserProgramStartHook, 1985
  - FileFirmwareUpdateCompletedHook, 1985
  - FileFirmwareUpdateErrorHook, 1985
  - FileFirmwareUpdateLogHook, 1985
  - FileFirmwareUpdateStartedHook, 1986
  - FileGetFirmwareFilenameHook, 1986
  - FileFirmwareUpdateRequestedHook, 1986
  - handle, 1990
  - NvmDoneHook, 1986
  - NvmEraseHook, 1987
  - NvmInitHook, 1987
  - NvmReinitHook, 1987
  - NvmWriteHook, 1988
  - UsbConnectHook, 1988
  - UsbEnterLowPowerModeHook, 1989
  - UsbLeaveLowPowerModeHook, 1989
  - XcpGetSeedHook, 1989

- XcpVerifyKeyHook, [1990](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_GCC/↵
  - Boot/led.c
    - LedBlinkExit, [2304](#)
    - LedBlinkInit, [2304](#)
    - LedBlinkTask, [2304](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_GCC/↵
  - Boot/led.h
    - LedBlinkExit, [2538](#)
    - LedBlinkInit, [2538](#)
    - LedBlinkTask, [2538](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_GCC/↵
  - Boot/main.c
    - HAL\_MspDeInit, [2816](#)
    - HAL\_MspInit, [2816](#)
    - Init, [2816](#)
    - main, [2816](#)
    - SystemClock\_Config, [2817](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_GCC/↵
  - Prog/led.c
    - LedInit, [2305](#)
    - LedToggle, [2305](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_GCC/↵
  - Prog/led.h
    - LedInit, [2539](#)
    - LedToggle, [2540](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_GCC/↵
  - Prog/main.c
    - HAL\_MspDeInit, [2818](#)
    - HAL\_MspInit, [2818](#)
    - Init, [2818](#)
    - main, [2819](#)
    - SystemClock\_Config, [2819](#)
    - VectorBase\_Config, [2819](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_IAR/↵
  - Boot/hooks.c
    - BackDoorEntryHook, [1992](#)
    - BackDoorInitHook, [1993](#)
    - canUse, [1999](#)
    - CopInitHook, [1993](#)
    - CopServiceHook, [1993](#)
    - CpuUserProgramStartHook, [1993](#)
    - FileFirmwareUpdateCompletedHook, [1994](#)
    - FileFirmwareUpdateErrorHook, [1994](#)
    - FileFirmwareUpdateLogHook, [1994](#)
    - FileFirmwareUpdateStartedHook, [1995](#)
    - FileGetFirmwareFilenameHook, [1995](#)
    - FileFirmwareUpdateRequestedHook, [1995](#)
    - handle, [1999](#)
    - NvmDoneHook, [1995](#)
    - NvmEraseHook, [1996](#)
    - NvmInitHook, [1996](#)
    - NvmReinitHook, [1996](#)
    - NvmWriteHook, [1997](#)
    - UsbConnectHook, [1997](#)
    - UsbEnterLowPowerModeHook, [1998](#)
    - UsbLeaveLowPowerModeHook, [1998](#)
    - XcpGetSeedHook, [1998](#)
- XcpVerifyKeyHook, [1999](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_IAR/↵
  - Boot/led.c
    - LedBlinkExit, [2307](#)
    - LedBlinkInit, [2307](#)
    - LedBlinkTask, [2307](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_IAR/↵
  - Boot/led.h
    - LedBlinkExit, [2541](#)
    - LedBlinkInit, [2541](#)
    - LedBlinkTask, [2541](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_IAR/↵
  - Boot/main.c
    - HAL\_MspDeInit, [2821](#)
    - HAL\_MspInit, [2821](#)
    - Init, [2821](#)
    - main, [2821](#)
    - SystemClock\_Config, [2822](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_IAR/↵
  - Prog/led.c
    - LedInit, [2308](#)
    - LedToggle, [2308](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_IAR/↵
  - Prog/led.h
    - LedInit, [2542](#)
    - LedToggle, [2543](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_IAR/↵
  - Prog/main.c
    - HAL\_MspDeInit, [2823](#)
    - HAL\_MspInit, [2823](#)
    - Init, [2823](#)
    - main, [2824](#)
    - SystemClock\_Config, [2824](#)
    - VectorBase\_Config, [2824](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_Keil/↵
  - Boot/hooks.c
    - BackDoorEntryHook, [2001](#)
    - BackDoorInitHook, [2002](#)
    - canUse, [2008](#)
    - CopInitHook, [2002](#)
    - CopServiceHook, [2002](#)
    - CpuUserProgramStartHook, [2002](#)
    - FileFirmwareUpdateCompletedHook, [2003](#)
    - FileFirmwareUpdateErrorHook, [2003](#)
    - FileFirmwareUpdateLogHook, [2003](#)
    - FileFirmwareUpdateStartedHook, [2004](#)
    - FileGetFirmwareFilenameHook, [2004](#)
    - FileFirmwareUpdateRequestedHook, [2004](#)
    - handle, [2008](#)
    - NvmDoneHook, [2004](#)
    - NvmEraseHook, [2005](#)
    - NvmInitHook, [2005](#)
    - NvmReinitHook, [2005](#)
    - NvmWriteHook, [2006](#)
    - UsbConnectHook, [2006](#)
    - UsbEnterLowPowerModeHook, [2007](#)
    - UsbLeaveLowPowerModeHook, [2007](#)
    - XcpGetSeedHook, [2007](#)

- XcpVerifyKeyHook, [2008](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_Keil/↔
  - Boot/led.c
    - LedBlinkExit, [2310](#)
    - LedBlinkInit, [2310](#)
    - LedBlinkTask, [2310](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_Keil/↔
  - Boot/led.h
    - LedBlinkExit, [2544](#)
    - LedBlinkInit, [2544](#)
    - LedBlinkTask, [2544](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_Keil/↔
  - Boot/main.c
    - HAL\_MspDeInit, [2826](#)
    - HAL\_MspInit, [2826](#)
    - Init, [2826](#)
    - main, [2826](#)
    - SystemClock\_Config, [2827](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_Keil/↔
  - Prog/led.c
    - LedInit, [2311](#)
    - LedToggle, [2311](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_Keil/↔
  - Prog/led.h
    - LedInit, [2545](#)
    - LedToggle, [2546](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_Keil/↔
  - Prog/main.c
    - HAL\_MspDeInit, [2828](#)
    - HAL\_MspInit, [2828](#)
    - Init, [2828](#)
    - main, [2829](#)
    - SystemClock\_Config, [2829](#)
    - VectorBase\_Config, [2829](#)
- ARMCM4\_STM32L4/GCC/cpu\_comp.c
  - CpulrqDisable, [1231](#)
  - CpulrqEnable, [1231](#)
- ARMCM4\_STM32L4/IAR/cpu\_comp.c
  - CpulrqDisable, [1232](#)
  - CpulrqEnable, [1233](#)
- ARMCM4\_STM32L4/Keil/cpu\_comp.c
  - CpulrqDisable, [1234](#)
  - CpulrqEnable, [1234](#)
- ARMCM4\_STM32L4/can.c
  - CanGetSpeedConfig, [1103](#)
  - CanInit, [1104](#)
  - CanReceivePacket, [1104](#)
  - canTiming, [1105](#)
  - CanTransmitPacket, [1104](#)
- ARMCM4\_STM32L4/cpu.c
  - CpuInit, [1171](#)
  - CpuMemCopy, [1171](#)
  - CpuMemSet, [1171](#)
  - CpuStartUserProgram, [1172](#)
  - CpuUserProgramStartHook, [1172](#)
- ARMCM4\_STM32L4/flash.c
  - blockInfo, [1385](#)
  - bootBlockInfo, [1385](#)
- FlashAddToBlock, [1379](#)
- FlashDone, [1380](#)
- FlashErase, [1380](#)
- FlashGetBank, [1380](#)
- FlashGetPage, [1382](#)
- FlashGetUserProgBaseAddress, [1382](#)
- FlashInit, [1382](#)
- FlashInitBlock, [1383](#)
- flashLayout, [1386](#)
- FlashReinit, [1383](#)
- FlashSwitchBlock, [1383](#)
- FlashVerifyChecksum, [1384](#)
- FlashWrite, [1384](#)
- FlashWriteBlock, [1384](#)
- FlashWriteChecksum, [1385](#)
- ARMCM4\_STM32L4/flash.h
  - FlashDone, [1481](#)
  - FlashErase, [1481](#)
  - FlashGetUserProgBaseAddress, [1482](#)
  - FlashInit, [1482](#)
  - FlashReinit, [1482](#)
  - FlashVerifyChecksum, [1482](#)
  - FlashWrite, [1483](#)
  - FlashWriteChecksum, [1483](#)
- ARMCM4\_STM32L4/nvm.c
  - NvmDone, [3026](#)
  - NvmDoneHook, [3026](#)
  - NvmErase, [3026](#)
  - NvmEraseHook, [3027](#)
  - NvmGetUserProgBaseAddress, [3027](#)
  - NvmInit, [3027](#)
  - NvmInitHook, [3028](#)
  - NvmReinit, [3028](#)
  - NvmReinitHook, [3028](#)
  - NvmVerifyChecksum, [3028](#)
  - NvmWrite, [3029](#)
  - NvmWriteHook, [3029](#)
- ARMCM4\_STM32L4/types.h
  - blt\_addr, [3556](#)
  - blt\_bool, [3556](#)
  - blt\_char, [3556](#)
  - blt\_int16s, [3556](#)
  - blt\_int16u, [3557](#)
  - blt\_int32s, [3557](#)
  - blt\_int32u, [3557](#)
  - blt\_int8s, [3557](#)
  - blt\_int8u, [3557](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_CubeIDE/↔
  - Boot/App/app.c
    - AppInit, [421](#)
    - AppTask, [422](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_CubeIDE/↔
  - Boot/App/app.h
    - AppInit, [464](#)
    - AppTask, [464](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_CubeIDE/↔
  - Boot/App/hooks.c
    - BackDoorEntryHook, [2010](#)

- BackDoorInitHook, [2010](#)
- CopInitHook, [2010](#)
- CopServiceHook, [2011](#)
- CpuUserProgramStartHook, [2011](#)
- NvmDoneHook, [2011](#)
- NvmEraseHook, [2011](#)
- NvmInitHook, [2012](#)
- NvmReinitHook, [2012](#)
- NvmWriteHook, [2012](#)
- XcpGetSeedHook, [2013](#)
- XcpVerifyKeyHook, [2013](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_CubeIDE/↔
  - Boot/App/led.c
  - LedBlinkExit, [2313](#)
  - LedBlinkInit, [2313](#)
  - LedBlinkTask, [2313](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_CubeIDE/↔
  - Boot/App/led.h
  - LedBlinkExit, [2547](#)
  - LedBlinkInit, [2547](#)
  - LedBlinkTask, [2547](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_CubeIDE/↔
  - Prog/App/app.c
  - AppInit, [423](#)
  - AppTask, [423](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_CubeIDE/↔
  - Prog/App/app.h
  - AppInit, [465](#)
  - AppTask, [465](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_CubeIDE/↔
  - Prog/App/led.c
  - LedInit, [2314](#)
  - LedToggle, [2314](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_CubeIDE/↔
  - Prog/App/led.h
  - LedInit, [2548](#)
  - LedToggle, [2549](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_GCC/Boot/hooks.↔
  - c
  - BackDoorEntryHook, [2015](#)
  - BackDoorInitHook, [2015](#)
  - CopInitHook, [2015](#)
  - CopServiceHook, [2016](#)
  - CpuUserProgramStartHook, [2016](#)
  - NvmDoneHook, [2016](#)
  - NvmEraseHook, [2016](#)
  - NvmInitHook, [2017](#)
  - NvmReinitHook, [2017](#)
  - NvmWriteHook, [2017](#)
  - XcpGetSeedHook, [2018](#)
  - XcpVerifyKeyHook, [2018](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_GCC/Boot/led.↔
  - c
  - LedBlinkExit, [2316](#)
  - LedBlinkInit, [2316](#)
  - LedBlinkTask, [2316](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_GCC/Boot/led.↔
  - h
- LedBlinkExit, [2550](#)
- LedBlinkInit, [2550](#)
- LedBlinkTask, [2550](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_GCC/Boot/main.↔
  - c
  - HAL\_MspDeInit, [2831](#)
  - HAL\_MspInit, [2831](#)
  - Init, [2831](#)
  - main, [2831](#)
  - SystemClock\_Config, [2832](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_GCC/Prog/led.↔
  - c
  - LedInit, [2317](#)
  - LedToggle, [2317](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_GCC/Prog/led.↔
  - h
  - LedInit, [2551](#)
  - LedToggle, [2552](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_GCC/Prog/main.↔
  - c
  - HAL\_MspDeInit, [2833](#)
  - HAL\_MspInit, [2833](#)
  - Init, [2833](#)
  - main, [2834](#)
  - SystemClock\_Config, [2834](#)
  - VectorBase\_Config, [2834](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_IAR/Boot/hooks.↔
  - c
  - BackDoorEntryHook, [2020](#)
  - BackDoorInitHook, [2020](#)
  - CopInitHook, [2020](#)
  - CopServiceHook, [2021](#)
  - CpuUserProgramStartHook, [2021](#)
  - NvmDoneHook, [2021](#)
  - NvmEraseHook, [2021](#)
  - NvmInitHook, [2022](#)
  - NvmReinitHook, [2022](#)
  - NvmWriteHook, [2022](#)
  - XcpGetSeedHook, [2023](#)
  - XcpVerifyKeyHook, [2023](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_IAR/Boot/led.c
  - LedBlinkExit, [2319](#)
  - LedBlinkInit, [2319](#)
  - LedBlinkTask, [2319](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_IAR/Boot/led.h
  - LedBlinkExit, [2553](#)
  - LedBlinkInit, [2553](#)
  - LedBlinkTask, [2553](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_IAR/Boot/main.↔
  - c
  - HAL\_MspDeInit, [2836](#)
  - HAL\_MspInit, [2836](#)
  - Init, [2836](#)
  - main, [2836](#)
  - SystemClock\_Config, [2837](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_IAR/Prog/led.c
  - LedInit, [2320](#)
  - LedToggle, [2320](#)

- ARMCM4\_STM32L4\_Nucleo\_L476RG\_IAR/Prog/led.h
  - LedInit, [2554](#)
  - LedToggle, [2555](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_IAR/Prog/main.c↔
  - HAL\_MspDeInit, [2838](#)
  - HAL\_MspInit, [2838](#)
  - Init, [2838](#)
  - main, [2839](#)
  - SystemClock\_Config, [2839](#)
  - VectorBase\_Config, [2839](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_Keil/Boot/hooks.c↔
  - BackDoorEntryHook, [2025](#)
  - BackDoorInitHook, [2025](#)
  - CopInitHook, [2025](#)
  - CopServiceHook, [2026](#)
  - CpuUserProgramStartHook, [2026](#)
  - NvmDoneHook, [2026](#)
  - NvmEraseHook, [2026](#)
  - NvmInitHook, [2027](#)
  - NvmReinitHook, [2027](#)
  - NvmWriteHook, [2027](#)
  - XcpGetSeedHook, [2028](#)
  - XcpVerifyKeyHook, [2028](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_Keil/Boot/led.c
  - LedBlinkExit, [2322](#)
  - LedBlinkInit, [2322](#)
  - LedBlinkTask, [2322](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_Keil/Boot/led.h
  - LedBlinkExit, [2556](#)
  - LedBlinkInit, [2556](#)
  - LedBlinkTask, [2556](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_Keil/Boot/main.c↔
  - HAL\_MspDeInit, [2841](#)
  - HAL\_MspInit, [2841](#)
  - Init, [2841](#)
  - main, [2841](#)
  - SystemClock\_Config, [2842](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_Keil/Prog/led.c
  - LedInit, [2323](#)
  - LedToggle, [2323](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_Keil/Prog/led.h
  - LedInit, [2557](#)
  - LedToggle, [2558](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_Keil/Prog/main.c↔
  - HAL\_MspDeInit, [2843](#)
  - HAL\_MspInit, [2843](#)
  - Init, [2843](#)
  - main, [2844](#)
  - SystemClock\_Config, [2844](#)
  - VectorBase\_Config, [2844](#)
- ARMCM4\_TM4C/IAR/cpu\_comp.c
  - CpuIrqDisable, [1235](#)
  - CpuIrqEnable, [1235](#)
- ARMCM4\_TM4C/cpu.c
  - CpuInit, [1174](#)
  - CpuMemCopy, [1174](#)
  - CpuMemSet, [1174](#)
  - CpuStartUserProgram, [1175](#)
  - CpuUserProgramStartHook, [1175](#)
- ARMCM4\_TM4C/flash.c
  - blockInfo, [1394](#)
  - bootBlockInfo, [1394](#)
  - FlashAddToBlock, [1388](#)
  - FlashDone, [1389](#)
  - FlashErase, [1389](#)
  - FlashEraseSectors, [1389](#)
  - FlashGetSector, [1390](#)
  - FlashGetSectorBaseAddr, [1390](#)
  - FlashGetSectorSize, [1391](#)
  - FlashGetUserProgBaseAddress, [1391](#)
  - FlashInit, [1391](#)
  - FlashInitBlock, [1391](#)
  - flashLayout, [1395](#)
  - FlashReinit, [1392](#)
  - FlashSwitchBlock, [1392](#)
  - FlashVerifyChecksum, [1393](#)
  - FlashWrite, [1393](#)
  - FlashWriteBlock, [1393](#)
  - FlashWriteChecksum, [1394](#)
- ARMCM4\_TM4C/flash.h
  - FlashDone, [1484](#)
  - FlashErase, [1485](#)
  - FlashGetUserProgBaseAddress, [1485](#)
  - FlashInit, [1486](#)
  - FlashReinit, [1486](#)
  - FlashVerifyChecksum, [1486](#)
  - FlashWrite, [1486](#)
  - FlashWriteChecksum, [1487](#)
- ARMCM4\_TM4C/nvm.c
  - NvmDone, [3031](#)
  - NvmDoneHook, [3031](#)
  - NvmErase, [3031](#)
  - NvmEraseHook, [3032](#)
  - NvmGetUserProgBaseAddress, [3032](#)
  - NvmInit, [3032](#)
  - NvmInitHook, [3033](#)
  - NvmReinit, [3033](#)
  - NvmReinitHook, [3033](#)
  - NvmVerifyChecksum, [3033](#)
  - NvmWrite, [3034](#)
  - NvmWriteHook, [3034](#)
- ARMCM4\_TM4C/types.h
  - blt\_addr, [3558](#)
  - blt\_bool, [3558](#)
  - blt\_char, [3558](#)
  - blt\_int16s, [3558](#)
  - blt\_int16u, [3559](#)
  - blt\_int32s, [3559](#)
  - blt\_int32u, [3559](#)
  - blt\_int8s, [3559](#)
  - blt\_int8u, [3559](#)
- ARMCM4\_TM4C/usb.c



- UsbBulkRxHandler, [3601](#)
- UsbBulkTxHandler, [3601](#)
- UsbFifoMgrCreate, [3602](#)
- UsbFifoMgrInit, [3602](#)
- UsbFifoMgrRead, [3603](#)
- UsbFifoMgrScan, [3603](#)
- UsbFifoMgrWrite, [3603](#)
- UsbFree, [3604](#)
- UsbInit, [3604](#)
- UsbReceiveByte, [3604](#)
- UsbReceivePacket, [3605](#)
- UsbReceivePipeBulkOUT, [3605](#)
- UsbTransmitByte, [3606](#)
- UsbTransmitPacket, [3606](#)
- UsbTransmitPipeBulkIN, [3606](#)
- ARMCM4\_TM4C\_DK\_TM4C123G\_IAR/Boot/hooks.c
  - BackDoorEntryHook, [2031](#)
  - BackDoorInitHook, [2031](#)
  - canUse, [2037](#)
  - CopInitHook, [2031](#)
  - CopServiceHook, [2031](#)
  - CpuUserProgramStartHook, [2032](#)
  - FileFirmwareUpdateCompletedHook, [2032](#)
  - FileFirmwareUpdateErrorHook, [2032](#)
  - FileFirmwareUpdateLogHook, [2032](#)
  - FileFirmwareUpdateStartedHook, [2033](#)
  - FileGetFirmwareFilenameHook, [2033](#)
  - FileIsFirmwareUpdateRequestedHook, [2033](#)
  - handle, [2037](#)
  - NvmDoneHook, [2033](#)
  - NvmEraseHook, [2034](#)
  - NvmInitHook, [2034](#)
  - NvmReinitHook, [2034](#)
  - NvmWriteHook, [2035](#)
  - UsbConnectHook, [2035](#)
  - UsbEnterLowPowerModeHook, [2036](#)
  - UsbLeaveLowPowerModeHook, [2036](#)
  - XcpGetSeedHook, [2036](#)
  - XcpVerifyKeyHook, [2037](#)
- ARMCM4\_TM4C\_DK\_TM4C123G\_IAR/Boot/main.c
  - Init, [2846](#)
  - main, [2846](#)
- ARMCM4\_TM4C\_DK\_TM4C123G\_IAR/Boot/vectors.c
  - reset\_handler, [3640](#)
  - UnusedISR, [3640](#)
- ARMCM4\_TM4C\_DK\_TM4C123G\_IAR/Prog/led.c
  - LedInit, [2324](#)
  - LedToggle, [2325](#)
- ARMCM4\_TM4C\_DK\_TM4C123G\_IAR/Prog/led.h
  - LedInit, [2559](#)
  - LedToggle, [2559](#)
- ARMCM4\_TM4C\_DK\_TM4C123G\_IAR/Prog/main.c
  - Init, [2847](#)
  - main, [2847](#)
- ARMCM4\_TM4C\_DK\_TM4C123G\_IAR/Prog/time.c
  - TimerDeinit, [3229](#)
  - TimerGet, [3229](#)
  - TimerISRHandler, [3229](#)
  - TimerInit, [3229](#)
  - TimerSet, [3229](#)
- ARMCM4\_TM4C\_DK\_TM4C123G\_IAR/Prog/time.h
  - TimerDeinit, [3240](#)
  - TimerGet, [3240](#)
  - TimerISRHandler, [3241](#)
  - TimerInit, [3240](#)
  - TimerSet, [3241](#)
- ARMCM4\_TM4C\_DK\_TM4C123G\_IAR/Prog/vectors.c
  - ResetISR, [3642](#)
  - UnusedISR, [3642](#)
- ARMCM4\_XMC4/GCC/cpu\_comp.c
  - CpulrqDisable, [1236](#)
  - CpulrqEnable, [1236](#)
- ARMCM4\_XMC4/IAR/cpu\_comp.c
  - CpulrqDisable, [1237](#)
  - CpulrqEnable, [1238](#)
- ARMCM4\_XMC4/can.c
  - CanInit, [1107](#)
  - CanReceivePacket, [1107](#)
  - CanTransmitPacket, [1107](#)
- ARMCM4\_XMC4/cpu.c
  - CpuInit, [1177](#)
  - CpuMemCopy, [1177](#)
  - CpuMemSet, [1177](#)
  - CpuStartUserProgram, [1178](#)
  - CpuUserProgramStartHook, [1178](#)
- ARMCM4\_XMC4/flash.c
  - blockInfo, [1403](#)
  - bootBlockInfo, [1404](#)
  - FlashAddToBlock, [1397](#)
  - FlashDone, [1398](#)
  - FlashErase, [1398](#)
  - FlashEraseSectors, [1399](#)
  - FlashGetSector, [1399](#)
  - FlashGetSectorBaseAddr, [1399](#)
  - FlashGetUserProgBaseAddress, [1400](#)
  - FlashInit, [1400](#)
  - FlashInitBlock, [1400](#)
  - flashLayout, [1404](#)
  - FlashReinit, [1401](#)
  - FlashSwitchBlock, [1401](#)
  - FlashTranslateToNonCachedAddress, [1401](#)
  - FlashVerifyChecksum, [1402](#)
  - FlashWrite, [1402](#)
  - FlashWriteBlock, [1403](#)
  - FlashWriteChecksum, [1403](#)
- ARMCM4\_XMC4/flash.h
  - FlashDone, [1488](#)
  - FlashErase, [1488](#)
  - FlashGetUserProgBaseAddress, [1489](#)
  - FlashInit, [1489](#)
  - FlashReinit, [1489](#)
  - FlashVerifyChecksum, [1490](#)
  - FlashWrite, [1490](#)
  - FlashWriteChecksum, [1490](#)
- ARMCM4\_XMC4/nvm.c
  - NvmDone, [3036](#)

- NvmDoneHook, [3036](#)
- NvmErase, [3036](#)
- NvmEraseHook, [3037](#)
- NvmGetUserProgBaseAddress, [3037](#)
- NvmInit, [3037](#)
- NvmInitHook, [3038](#)
- NvmReinit, [3038](#)
- NvmReinitHook, [3038](#)
- NvmVerifyChecksum, [3038](#)
- NvmWrite, [3039](#)
- NvmWriteHook, [3039](#)
- ARMCM4\_XMC4/types.h
  - blt\_addr, [3560](#)
  - blt\_bool, [3560](#)
  - blt\_char, [3560](#)
  - blt\_int16s, [3560](#)
  - blt\_int16u, [3561](#)
  - blt\_int32s, [3561](#)
  - blt\_int32u, [3561](#)
  - blt\_int8s, [3561](#)
  - blt\_int8u, [3561](#)
- ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_GCC/↵
  - Boot/hooks.c
    - BackDoorEntryHook, [2039](#)
    - BackDoorInitHook, [2039](#)
    - canUse, [2045](#)
    - CopInitHook, [2040](#)
    - CopServiceHook, [2040](#)
    - CpuUserProgramStartHook, [2040](#)
    - FileFirmwareUpdateCompletedHook, [2040](#)
    - FileFirmwareUpdateErrorHook, [2041](#)
    - FileFirmwareUpdateLogHook, [2041](#)
    - FileFirmwareUpdateStartedHook, [2041](#)
    - FileGetFirmwareFilenameHook, [2042](#)
    - FileIsFirmwareUpdateRequestedHook, [2042](#)
    - handle, [2045](#)
    - NvmDoneHook, [2042](#)
    - NvmEraseHook, [2042](#)
    - NvmInitHook, [2043](#)
    - NvmReinitHook, [2043](#)
    - NvmWriteHook, [2043](#)
    - XcpGetSeedHook, [2044](#)
    - XcpVerifyKeyHook, [2044](#)
- ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_GCC/↵
  - Boot/led.c
    - LedBlinkExit, [2326](#)
    - LedBlinkInit, [2326](#)
    - LedBlinkTask, [2327](#)
- ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_GCC/↵
  - Boot/led.h
    - LedBlinkExit, [2560](#)
    - LedBlinkInit, [2560](#)
    - LedBlinkTask, [2561](#)
- ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_GCC/↵
  - Boot/main.c
    - Init, [2848](#)
    - main, [2848](#)
    - PostInit, [2848](#)
- ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_GCC/↵
  - Boot/shared\_params.c
    - \_\_attribute\_\_, [3119](#)
    - SharedParamsCalculateChecksum, [3120](#)
    - SharedParamsInit, [3120](#)
    - SharedParamsReadByIndex, [3121](#)
    - SharedParamsValidateBuffer, [3121](#)
    - SharedParamsVerifyChecksum, [3121](#)
    - SharedParamsWriteByIndex, [3122](#)
    - SharedParamsWriteChecksum, [3122](#)
- ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_GCC/↵
  - Boot/shared\_params.h
    - SharedParamsInit, [3192](#)
    - SharedParamsReadByIndex, [3192](#)
    - SharedParamsWriteByIndex, [3193](#)
- ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_GCC/↵
  - Prog/led.c
    - LedInit, [2328](#)
    - LedToggle, [2328](#)
- ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_GCC/↵
  - Prog/led.h
    - LedInit, [2562](#)
    - LedToggle, [2562](#)
- ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_GCC/↵
  - Prog/main.c
    - Init, [2849](#)
    - main, [2850](#)
- ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_GCC/↵
  - Prog/shared\_params.c
    - \_\_attribute\_\_, [3124](#)
    - SharedParamsCalculateChecksum, [3124](#)
    - SharedParamsInit, [3125](#)
    - SharedParamsReadByIndex, [3125](#)
    - SharedParamsValidateBuffer, [3125](#)
    - SharedParamsVerifyChecksum, [3126](#)
    - SharedParamsWriteByIndex, [3126](#)
    - SharedParamsWriteChecksum, [3126](#)
- ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_GCC/↵
  - Prog/shared\_params.h
    - SharedParamsInit, [3195](#)
    - SharedParamsReadByIndex, [3195](#)
    - SharedParamsWriteByIndex, [3195](#)
- ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_IAR/Boot/hooks.↵
  - c
    - BackDoorEntryHook, [2047](#)
    - BackDoorInitHook, [2047](#)
    - canUse, [2053](#)
    - CopInitHook, [2047](#)
    - CopServiceHook, [2048](#)
    - CpuUserProgramStartHook, [2048](#)
    - FileFirmwareUpdateCompletedHook, [2048](#)
    - FileFirmwareUpdateErrorHook, [2048](#)
    - FileFirmwareUpdateLogHook, [2049](#)
    - FileFirmwareUpdateStartedHook, [2049](#)
    - FileGetFirmwareFilenameHook, [2049](#)
    - FileIsFirmwareUpdateRequestedHook, [2050](#)
    - handle, [2053](#)
    - NvmDoneHook, [2050](#)



- NvmEraseHook, [2050](#)
- NvmInitHook, [2051](#)
- NvmReinitHook, [2051](#)
- NvmWriteHook, [2051](#)
- XcpGetSeedHook, [2052](#)
- XcpVerifyKeyHook, [2052](#)
- ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_IAR/Boot/led.↔
  - c
  - LedBlinkExit, [2329](#)
  - LedBlinkInit, [2329](#)
  - LedBlinkTask, [2330](#)
- ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_IAR/Boot/led.↔
  - h
  - LedBlinkExit, [2563](#)
  - LedBlinkInit, [2563](#)
  - LedBlinkTask, [2563](#)
- ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_IAR/Boot/main.↔
  - c
  - Init, [2851](#)
  - main, [2851](#)
  - PostInit, [2851](#)
- ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_IAR/Boot/shared↔
  - \_params.c
  - shared, [3131](#)
  - SharedParamsCalculateChecksum, [3128](#)
  - SharedParamsInit, [3128](#)
  - SharedParamsReadByIndex, [3129](#)
  - SharedParamsValidateBuffer, [3129](#)
  - SharedParamsVerifyChecksum, [3129](#)
  - SharedParamsWriteByIndex, [3130](#)
  - SharedParamsWriteChecksum, [3130](#)
- ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_IAR/Boot/shared↔
  - \_params.h
  - SharedParamsInit, [3197](#)
  - SharedParamsReadByIndex, [3197](#)
  - SharedParamsWriteByIndex, [3198](#)
- ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_IAR/Prog/led.↔
  - c
  - LedInit, [2331](#)
  - LedToggle, [2331](#)
- ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_IAR/Prog/led.↔
  - h
  - LedInit, [2564](#)
  - LedToggle, [2565](#)
- ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_IAR/Prog/main.↔
  - c
  - Init, [2852](#)
  - main, [2853](#)
- ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_IAR/Prog/shared↔
  - \_params.c
  - shared, [3135](#)
  - SharedParamsCalculateChecksum, [3133](#)
  - SharedParamsInit, [3133](#)
  - SharedParamsReadByIndex, [3133](#)
  - SharedParamsValidateBuffer, [3134](#)
  - SharedParamsVerifyChecksum, [3134](#)
  - SharedParamsWriteByIndex, [3134](#)
  - SharedParamsWriteChecksum, [3135](#)
- ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_IAR/Prog/shared↔
  - \_params.h
  - SharedParamsInit, [3199](#)
  - SharedParamsReadByIndex, [3199](#)
  - SharedParamsWriteByIndex, [3200](#)
- ARMCM7\_STM32F7/GCC/cpu\_comp.c
  - CpulrqDisable, [1239](#)
  - CpulrqEnable, [1239](#)
- ARMCM7\_STM32F7/IAR/cpu\_comp.c
  - CpulrqDisable, [1240](#)
  - CpulrqEnable, [1240](#)
- ARMCM7\_STM32F7/Keil/cpu\_comp.c
  - CpulrqDisable, [1241](#)
  - CpulrqEnable, [1241](#)
- ARMCM7\_STM32F7/can.c
  - CanGetSpeedConfig, [1109](#)
  - CanInit, [1110](#)
  - CanReceivePacket, [1110](#)
  - canTiming, [1111](#)
  - CanTransmitPacket, [1110](#)
- ARMCM7\_STM32F7/cpu.c
  - CpuInit, [1180](#)
  - CpuMemCopy, [1180](#)
  - CpuMemSet, [1180](#)
  - CpuStartUserProgram, [1181](#)
  - CpuUserProgramStartHook, [1181](#)
- ARMCM7\_STM32F7/flash.c
  - blockInfo, [1412](#)
  - bootBlockInfo, [1412](#)
  - FlashAddToBlock, [1407](#)
  - FlashDone, [1407](#)
  - FlashErase, [1408](#)
  - FlashEraseSectors, [1408](#)
  - FlashGetSector, [1409](#)
  - FlashGetUserProgBaseAddress, [1409](#)
  - FlashInit, [1409](#)
  - FlashInitBlock, [1409](#)
  - FlashIsSingleBankMode, [1410](#)
  - flashLayout, [1413](#)
  - FlashReinit, [1410](#)
  - FlashSwitchBlock, [1410](#)
  - FlashVerifyChecksum, [1411](#)
  - FlashWrite, [1411](#)
  - FlashWriteBlock, [1411](#)
  - FlashWriteChecksum, [1412](#)
- ARMCM7\_STM32F7/flash.h
  - FlashDone, [1492](#)
  - FlashErase, [1492](#)
  - FlashGetUserProgBaseAddress, [1493](#)
  - FlashInit, [1493](#)
  - FlashReinit, [1493](#)
  - FlashVerifyChecksum, [1493](#)
  - FlashWrite, [1494](#)
  - FlashWriteChecksum, [1494](#)
- ARMCM7\_STM32F7/nvm.c
  - NvmDone, [3041](#)
  - NvmDoneHook, [3041](#)
  - NvmErase, [3041](#)

- NvmEraseHook, [3042](#)
- NvmGetUserProgBaseAddress, [3042](#)
- NvmInit, [3042](#)
- NvmInitHook, [3043](#)
- NvmReinit, [3043](#)
- NvmReinitHook, [3043](#)
- NvmVerifyChecksum, [3043](#)
- NvmWrite, [3044](#)
- NvmWriteHook, [3044](#)
- ARMCM7\_STM32F7/types.h
  - blt\_addr, [3562](#)
  - blt\_bool, [3562](#)
  - blt\_char, [3562](#)
  - blt\_int16s, [3562](#)
  - blt\_int16u, [3563](#)
  - blt\_int32s, [3563](#)
  - blt\_int32u, [3563](#)
  - blt\_int8s, [3563](#)
  - blt\_int8u, [3563](#)
- ARMCM7\_STM32F7/usb.c
  - UsbFifoMgrCreate, [3609](#)
  - UsbFifoMgrInit, [3609](#)
  - UsbFifoMgrRead, [3609](#)
  - UsbFifoMgrScan, [3610](#)
  - UsbFifoMgrWrite, [3610](#)
  - UsbFree, [3611](#)
  - UsbInit, [3611](#)
  - UsbReceiveByte, [3611](#)
  - UsbReceivePacket, [3611](#)
  - UsbReceivePipeBulkOUT, [3612](#)
  - UsbTransmitByte, [3612](#)
  - UsbTransmitPacket, [3612](#)
  - UsbTransmitPipeBulkIN, [3613](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_CubeIDE/↔
  - Boot/App/app.c
    - AppInit, [424](#)
    - AppTask, [424](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_CubeIDE/↔
  - Boot/App/app.h
    - AppInit, [466](#)
    - AppTask, [466](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_CubeIDE/↔
  - Boot/App/hooks.c
    - BackDoorEntryHook, [2054](#)
    - BackDoorInitHook, [2055](#)
    - CopInitHook, [2055](#)
    - CopServiceHook, [2055](#)
    - CpuUserProgramStartHook, [2055](#)
    - NvmDoneHook, [2056](#)
    - NvmEraseHook, [2056](#)
    - NvmInitHook, [2056](#)
    - NvmReinitHook, [2057](#)
    - NvmWriteHook, [2057](#)
    - UsbConnectHook, [2057](#)
    - UsbEnterLowPowerModeHook, [2058](#)
    - UsbLeaveLowPowerModeHook, [2058](#)
    - XcpGetSeedHook, [2058](#)
    - XcpVerifyKeyHook, [2059](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_CubeIDE/↔
  - Boot/App/led.c
    - LedBlinkExit, [2332](#)
    - LedBlinkInit, [2332](#)
    - LedBlinkTask, [2333](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_CubeIDE/↔
  - Boot/App/led.h
    - LedBlinkExit, [2566](#)
    - LedBlinkInit, [2566](#)
    - LedBlinkTask, [2566](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_CubeIDE/↔
  - Prog/App/app.c
    - AppInit, [425](#)
    - AppTask, [425](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_CubeIDE/↔
  - Prog/App/app.h
    - AppInit, [468](#)
    - AppTask, [468](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_CubeIDE/↔
  - Prog/App/led.c
    - LedInit, [2334](#)
    - LedToggle, [2334](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_CubeIDE/↔
  - Prog/App/led.h
    - LedInit, [2567](#)
    - LedToggle, [2568](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_GCC/Boot/hooks.↔
  - c
    - BackDoorEntryHook, [2061](#)
    - BackDoorInitHook, [2061](#)
    - CopInitHook, [2061](#)
    - CopServiceHook, [2061](#)
    - CpuUserProgramStartHook, [2062](#)
    - NvmDoneHook, [2062](#)
    - NvmEraseHook, [2062](#)
    - NvmInitHook, [2063](#)
    - NvmReinitHook, [2063](#)
    - NvmWriteHook, [2063](#)
    - UsbConnectHook, [2064](#)
    - UsbEnterLowPowerModeHook, [2064](#)
    - UsbLeaveLowPowerModeHook, [2064](#)
    - XcpGetSeedHook, [2064](#)
    - XcpVerifyKeyHook, [2065](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_GCC/Boot/led.↔
  - c
    - LedBlinkExit, [2335](#)
    - LedBlinkInit, [2335](#)
    - LedBlinkTask, [2336](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_GCC/Boot/led.↔
  - h
    - LedBlinkExit, [2569](#)
    - LedBlinkInit, [2569](#)
    - LedBlinkTask, [2569](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_GCC/Boot/main.↔
  - c
    - HAL\_MspDelInit, [2854](#)
    - HAL\_MspInit, [2854](#)
    - Init, [2854](#)

- main, [2855](#)
- SystemClock\_Config, [2855](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_GCC/Prog/led.↔
  - c
  - LedInit, [2337](#)
  - LedToggle, [2337](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_GCC/Prog/led.↔
  - h
  - LedInit, [2570](#)
  - LedToggle, [2571](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_GCC/Prog/main.↔
  - c
  - HAL\_MspDeInit, [2856](#)
  - HAL\_MspInit, [2857](#)
  - Init, [2857](#)
  - main, [2857](#)
  - SystemClock\_Config, [2857](#)
  - VectorBase\_Config, [2858](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_IAR/Boot/hooks.↔
  - c
  - BackDoorEntryHook, [2067](#)
  - BackDoorInitHook, [2067](#)
  - CopInitHook, [2067](#)
  - CopServiceHook, [2067](#)
  - CpuUserProgramStartHook, [2068](#)
  - NvmDoneHook, [2068](#)
  - NvmEraseHook, [2068](#)
  - NvmInitHook, [2069](#)
  - NvmReinitHook, [2069](#)
  - NvmWriteHook, [2069](#)
  - UsbConnectHook, [2070](#)
  - UsbEnterLowPowerModeHook, [2070](#)
  - UsbLeaveLowPowerModeHook, [2070](#)
  - XcpGetSeedHook, [2070](#)
  - XcpVerifyKeyHook, [2071](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_IAR/Boot/led.c
  - LedBlinkExit, [2338](#)
  - LedBlinkInit, [2338](#)
  - LedBlinkTask, [2339](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_IAR/Boot/led.h
  - LedBlinkExit, [2572](#)
  - LedBlinkInit, [2572](#)
  - LedBlinkTask, [2572](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_IAR/Boot/main.↔
  - c
  - HAL\_MspDeInit, [2859](#)
  - HAL\_MspInit, [2859](#)
  - Init, [2859](#)
  - main, [2860](#)
  - SystemClock\_Config, [2860](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_IAR/Prog/led.c
  - LedInit, [2340](#)
  - LedToggle, [2340](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_IAR/Prog/led.h
  - LedInit, [2573](#)
  - LedToggle, [2574](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_IAR/Prog/main.↔
  - c
  - HAL\_MspDeInit, [2861](#)
  - HAL\_MspInit, [2862](#)
  - Init, [2862](#)
  - main, [2862](#)
  - SystemClock\_Config, [2862](#)
  - VectorBase\_Config, [2863](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_Keil/Boot/hooks.↔
  - c
  - BackDoorEntryHook, [2073](#)
  - BackDoorInitHook, [2073](#)
  - CopInitHook, [2073](#)
  - CopServiceHook, [2073](#)
  - CpuUserProgramStartHook, [2074](#)
  - NvmDoneHook, [2074](#)
  - NvmEraseHook, [2074](#)
  - NvmInitHook, [2075](#)
  - NvmReinitHook, [2075](#)
  - NvmWriteHook, [2075](#)
  - UsbConnectHook, [2076](#)
  - UsbEnterLowPowerModeHook, [2076](#)
  - UsbLeaveLowPowerModeHook, [2076](#)
  - XcpGetSeedHook, [2076](#)
  - XcpVerifyKeyHook, [2077](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_Keil/Boot/led.c
  - LedBlinkExit, [2341](#)
  - LedBlinkInit, [2341](#)
  - LedBlinkTask, [2342](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_Keil/Boot/led.h
  - LedBlinkExit, [2575](#)
  - LedBlinkInit, [2575](#)
  - LedBlinkTask, [2575](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_Keil/Boot/main.↔
  - c
  - HAL\_MspDeInit, [2864](#)
  - HAL\_MspInit, [2864](#)
  - Init, [2864](#)
  - main, [2865](#)
  - SystemClock\_Config, [2865](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_Keil/Prog/led.c
  - LedInit, [2343](#)
  - LedToggle, [2343](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_Keil/Prog/led.h
  - LedInit, [2576](#)
  - LedToggle, [2577](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_Keil/Prog/main.↔
  - c
  - HAL\_MspDeInit, [2866](#)
  - HAL\_MspInit, [2867](#)
  - Init, [2867](#)
  - main, [2867](#)
  - SystemClock\_Config, [2867](#)
  - VectorBase\_Config, [2868](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_CubeIDE/↔
  - Boot/App/app.c
  - AppInit, [427](#)
  - AppTask, [427](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_CubeIDE/↔
  - Boot/App/app.h

- AppInit, [469](#)
- AppTask, [469](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_CubeIDE/↔
  - Boot/App/hooks.c
  - BackDoorEntryHook, [2079](#)
  - BackDoorInitHook, [2079](#)
  - CopInitHook, [2079](#)
  - CopServiceHook, [2079](#)
  - CpuUserProgramStartHook, [2080](#)
  - NvmDoneHook, [2080](#)
  - NvmEraseHook, [2080](#)
  - NvmInitHook, [2081](#)
  - NvmReinitHook, [2081](#)
  - NvmWriteHook, [2081](#)
  - XcpGetSeedHook, [2082](#)
  - XcpVerifyKeyHook, [2082](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_CubeIDE/↔
  - Boot/App/led.c
  - LedBlinkExit, [2344](#)
  - LedBlinkInit, [2344](#)
  - LedBlinkTask, [2345](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_CubeIDE/↔
  - Boot/App/led.h
  - LedBlinkExit, [2578](#)
  - LedBlinkInit, [2578](#)
  - LedBlinkTask, [2578](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_CubeIDE/↔
  - Boot/App/shared\_params.c
  - \_\_attribute\_\_, [3137](#)
  - SharedParamsCalculateChecksum, [3137](#)
  - SharedParamsInit, [3138](#)
  - SharedParamsReadByIndex, [3138](#)
  - SharedParamsValidateBuffer, [3138](#)
  - SharedParamsVerifyChecksum, [3139](#)
  - SharedParamsWriteByIndex, [3139](#)
  - SharedParamsWriteChecksum, [3139](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_CubeIDE/↔
  - Boot/App/shared\_params.h
  - SharedParamsInit, [3202](#)
  - SharedParamsReadByIndex, [3202](#)
  - SharedParamsWriteByIndex, [3202](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_CubeIDE/↔
  - Prog/App/app.c
  - AppInit, [428](#)
  - AppTask, [428](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_CubeIDE/↔
  - Prog/App/app.h
  - AppInit, [470](#)
  - AppTask, [470](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_CubeIDE/↔
  - Prog/App/led.c
  - LedInit, [2346](#)
  - LedToggle, [2346](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_CubeIDE/↔
  - Prog/App/led.h
  - LedInit, [2579](#)
  - LedToggle, [2580](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_CubeIDE/↔
  - Prog/App/shared\_params.c
  - \_\_attribute\_\_, [3141](#)
  - SharedParamsCalculateChecksum, [3142](#)
  - SharedParamsInit, [3142](#)
  - SharedParamsReadByIndex, [3142](#)
  - SharedParamsValidateBuffer, [3143](#)
  - SharedParamsVerifyChecksum, [3143](#)
  - SharedParamsWriteByIndex, [3143](#)
  - SharedParamsWriteChecksum, [3144](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_CubeIDE/↔
  - Prog/App/shared\_params.h
  - SharedParamsInit, [3204](#)
  - SharedParamsReadByIndex, [3204](#)
  - SharedParamsWriteByIndex, [3205](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_GCC/Boot/hooks.↔
  - c
  - BackDoorEntryHook, [2084](#)
  - BackDoorInitHook, [2084](#)
  - CopInitHook, [2084](#)
  - CopServiceHook, [2085](#)
  - CpuUserProgramStartHook, [2085](#)
  - NvmDoneHook, [2085](#)
  - NvmEraseHook, [2085](#)
  - NvmInitHook, [2086](#)
  - NvmReinitHook, [2086](#)
  - NvmWriteHook, [2086](#)
  - XcpGetSeedHook, [2087](#)
  - XcpVerifyKeyHook, [2087](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_GCC/Boot/led.c
  - LedBlinkExit, [2347](#)
  - LedBlinkInit, [2347](#)
  - LedBlinkTask, [2348](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_GCC/Boot/led.h
  - LedBlinkExit, [2581](#)
  - LedBlinkInit, [2581](#)
  - LedBlinkTask, [2581](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_GCC/Boot/main.↔
  - c
  - HAL\_MspDelInit, [2869](#)
  - HAL\_MspInit, [2869](#)
  - Init, [2869](#)
  - main, [2870](#)
  - SystemClock\_Config, [2870](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_GCC/Boot/shared.↔
  - \_params.c
  - \_\_attribute\_\_, [3145](#)
  - SharedParamsCalculateChecksum, [3146](#)
  - SharedParamsInit, [3146](#)
  - SharedParamsReadByIndex, [3147](#)
  - SharedParamsValidateBuffer, [3147](#)
  - SharedParamsVerifyChecksum, [3147](#)
  - SharedParamsWriteByIndex, [3148](#)
  - SharedParamsWriteChecksum, [3148](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_GCC/Boot/shared.↔
  - \_params.h
  - SharedParamsInit, [3206](#)
  - SharedParamsReadByIndex, [3206](#)

- SharedParamsWriteByIndex, [3207](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_GCC/Prog/led.c
  - LedInit, [2349](#)
  - LedToggle, [2349](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_GCC/Prog/led.h
  - LedInit, [2582](#)
  - LedToggle, [2583](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_GCC/Prog/main.↔
  - c
  - HAL\_MspDelInit, [2871](#)
  - HAL\_MsplInit, [2872](#)
  - Init, [2872](#)
  - main, [2872](#)
  - SystemClock\_Config, [2872](#)
  - VectorBase\_Config, [2873](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_GCC/Prog/shared↔
  - \_params.c
  - \_\_attribute\_\_, [3150](#)
  - SharedParamsCalculateChecksum, [3150](#)
  - SharedParamsInit, [3151](#)
  - SharedParamsReadByIndex, [3151](#)
  - SharedParamsValidateBuffer, [3151](#)
  - SharedParamsVerifyChecksum, [3152](#)
  - SharedParamsWriteByIndex, [3152](#)
  - SharedParamsWriteChecksum, [3152](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_GCC/Prog/shared↔
  - \_params.h
  - SharedParamsInit, [3209](#)
  - SharedParamsReadByIndex, [3209](#)
  - SharedParamsWriteByIndex, [3209](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_IAR/Boot/hooks.↔
  - c
  - BackDoorEntryHook, [2089](#)
  - BackDoorInitHook, [2089](#)
  - CopInitHook, [2089](#)
  - CopServiceHook, [2090](#)
  - CpuUserProgramStartHook, [2090](#)
  - NvmDoneHook, [2090](#)
  - NvmEraseHook, [2090](#)
  - NvmInitHook, [2091](#)
  - NvmReinitHook, [2091](#)
  - NvmWriteHook, [2091](#)
  - XcpGetSeedHook, [2092](#)
  - XcpVerifyKeyHook, [2092](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_IAR/Boot/led.c
  - LedBlinkExit, [2350](#)
  - LedBlinkInit, [2350](#)
  - LedBlinkTask, [2351](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_IAR/Boot/led.h
  - LedBlinkExit, [2584](#)
  - LedBlinkInit, [2584](#)
  - LedBlinkTask, [2584](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_IAR/Boot/main.↔
  - c
  - HAL\_MspDelInit, [2874](#)
  - HAL\_MsplInit, [2874](#)
  - Init, [2874](#)
  - main, [2875](#)
- SystemClock\_Config, [2875](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_IAR/Boot/shared↔
  - \_params.c
  - shared, [3157](#)
  - SharedParamsCalculateChecksum, [3154](#)
  - SharedParamsInit, [3154](#)
  - SharedParamsReadByIndex, [3155](#)
  - SharedParamsValidateBuffer, [3155](#)
  - SharedParamsVerifyChecksum, [3155](#)
  - SharedParamsWriteByIndex, [3156](#)
  - SharedParamsWriteChecksum, [3156](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_IAR/Boot/shared↔
  - \_params.h
  - SharedParamsInit, [3211](#)
  - SharedParamsReadByIndex, [3211](#)
  - SharedParamsWriteByIndex, [3212](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_IAR/Prog/led.c
  - LedInit, [2352](#)
  - LedToggle, [2352](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_IAR/Prog/led.h
  - LedInit, [2585](#)
  - LedToggle, [2586](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_IAR/Prog/main.↔
  - c
  - HAL\_MspDelInit, [2876](#)
  - HAL\_MsplInit, [2877](#)
  - Init, [2877](#)
  - main, [2877](#)
  - SystemClock\_Config, [2877](#)
  - VectorBase\_Config, [2878](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_IAR/Prog/shared↔
  - \_params.c
  - shared, [3161](#)
  - SharedParamsCalculateChecksum, [3159](#)
  - SharedParamsInit, [3159](#)
  - SharedParamsReadByIndex, [3159](#)
  - SharedParamsValidateBuffer, [3160](#)
  - SharedParamsVerifyChecksum, [3160](#)
  - SharedParamsWriteByIndex, [3160](#)
  - SharedParamsWriteChecksum, [3161](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_IAR/Prog/shared↔
  - \_params.h
  - SharedParamsInit, [3213](#)
  - SharedParamsReadByIndex, [3213](#)
  - SharedParamsWriteByIndex, [3214](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_Keil/Boot/hooks.↔
  - c
  - BackDoorEntryHook, [2094](#)
  - BackDoorInitHook, [2094](#)
  - CopInitHook, [2094](#)
  - CopServiceHook, [2095](#)
  - CpuUserProgramStartHook, [2095](#)
  - NvmDoneHook, [2095](#)
  - NvmEraseHook, [2095](#)
  - NvmInitHook, [2096](#)
  - NvmReinitHook, [2096](#)
  - NvmWriteHook, [2096](#)
  - XcpGetSeedHook, [2097](#)

- XcpVerifyKeyHook, [2097](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_Keil/Boot/led.c
  - LedBlinkExit, [2353](#)
  - LedBlinkInit, [2353](#)
  - LedBlinkTask, [2354](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_Keil/Boot/led.h
  - LedBlinkExit, [2587](#)
  - LedBlinkInit, [2587](#)
  - LedBlinkTask, [2587](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_Keil/Boot/main.↔
  - c
  - HAL\_MspDelInit, [2879](#)
  - HAL\_MsplInit, [2879](#)
  - Init, [2879](#)
  - main, [2880](#)
  - SystemClock\_Config, [2880](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_Keil/Boot/shared↔
  - \_params.c
  - \_\_attribute\_\_, [3163](#)
  - SharedParamsCalculateChecksum, [3163](#)
  - SharedParamsInit, [3164](#)
  - SharedParamsReadByIndex, [3164](#)
  - SharedParamsValidateBuffer, [3164](#)
  - SharedParamsVerifyChecksum, [3165](#)
  - SharedParamsWriteByIndex, [3165](#)
  - SharedParamsWriteChecksum, [3165](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_Keil/Boot/shared↔
  - \_params.h
  - SharedParamsInit, [3216](#)
  - SharedParamsReadByIndex, [3216](#)
  - SharedParamsWriteByIndex, [3216](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_Keil/Prog/led.c
  - LedInit, [2355](#)
  - LedToggle, [2355](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_Keil/Prog/led.h
  - LedInit, [2588](#)
  - LedToggle, [2589](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_Keil/Prog/main.↔
  - c
  - HAL\_MspDelInit, [2881](#)
  - HAL\_MsplInit, [2882](#)
  - Init, [2882](#)
  - main, [2882](#)
  - SystemClock\_Config, [2882](#)
  - VectorBase\_Config, [2883](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_Keil/Prog/shared↔
  - \_params.c
  - \_\_attribute\_\_, [3167](#)
  - SharedParamsCalculateChecksum, [3168](#)
  - SharedParamsInit, [3168](#)
  - SharedParamsReadByIndex, [3168](#)
  - SharedParamsValidateBuffer, [3169](#)
  - SharedParamsVerifyChecksum, [3169](#)
  - SharedParamsWriteByIndex, [3169](#)
  - SharedParamsWriteChecksum, [3170](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_Keil/Prog/shared↔
  - \_params.h
  - SharedParamsInit, [3218](#)
- SharedParamsReadByIndex, [3218](#)
- SharedParamsWriteByIndex, [3219](#)
- ARMCM7\_STM32H7/GCC/cpu\_comp.c
  - CpuIrqDisable, [1242](#)
  - CpuIrqEnable, [1243](#)
- ARMCM7\_STM32H7/IAR/cpu\_comp.c
  - CpuIrqDisable, [1244](#)
  - CpuIrqEnable, [1244](#)
- ARMCM7\_STM32H7/Keil/cpu\_comp.c
  - CpuIrqDisable, [1245](#)
  - CpuIrqEnable, [1245](#)
- ARMCM7\_STM32H7/can.c
  - CanGetSpeedConfig, [1113](#)
  - CanInit, [1113](#)
  - CanReceivePacket, [1113](#)
  - canTiming, [1114](#)
  - CanTransmitPacket, [1114](#)
- ARMCM7\_STM32H7/cpu.c
  - CpuInit, [1183](#)
  - CpuMemCopy, [1183](#)
  - CpuMemSet, [1183](#)
  - CpuStartUserProgram, [1184](#)
  - CpuUserProgramStartHook, [1184](#)
- ARMCM7\_STM32H7/flash.c
  - blockInfo, [1420](#)
  - bootBlockInfo, [1421](#)
  - FlashAddToBlock, [1416](#)
  - FlashDone, [1416](#)
  - FlashErase, [1416](#)
  - FlashEraseSectors, [1417](#)
  - FlashGetSectorIdx, [1417](#)
  - FlashGetUserProgBaseAddress, [1417](#)
  - FlashInit, [1418](#)
  - FlashInitBlock, [1418](#)
  - flashLayout, [1421](#)
  - FlashReinit, [1418](#)
  - FlashSwitchBlock, [1419](#)
  - FlashVerifyChecksum, [1419](#)
  - FlashWrite, [1419](#)
  - FlashWriteBlock, [1420](#)
  - FlashWriteChecksum, [1420](#)
- ARMCM7\_STM32H7/flash.h
  - FlashDone, [1495](#)
  - FlashErase, [1496](#)
  - FlashGetUserProgBaseAddress, [1496](#)
  - FlashInit, [1497](#)
  - FlashReinit, [1497](#)
  - FlashVerifyChecksum, [1497](#)
  - FlashWrite, [1497](#)
  - FlashWriteChecksum, [1498](#)
- ARMCM7\_STM32H7/nvm.c
  - NvmDone, [3046](#)
  - NvmDoneHook, [3046](#)
  - NvmErase, [3046](#)
  - NvmEraseHook, [3047](#)
  - NvmGetUserProgBaseAddress, [3047](#)
  - NvmInit, [3047](#)
  - NvmInitHook, [3048](#)



- NvmReinit, [3048](#)
- NvmReinitHook, [3048](#)
- NvmVerifyChecksum, [3048](#)
- NvmWrite, [3049](#)
- NvmWriteHook, [3049](#)
- ARMCM7\_STM32H7/types.h
  - blt\_addr, [3564](#)
  - blt\_bool, [3564](#)
  - blt\_char, [3564](#)
  - blt\_int16s, [3564](#)
  - blt\_int16u, [3565](#)
  - blt\_int32s, [3565](#)
  - blt\_int32u, [3565](#)
  - blt\_int8s, [3565](#)
  - blt\_int8u, [3565](#)
- ARMCM7\_STM32H7/usb.c
  - UsbFifoMgrCreate, [3615](#)
  - UsbFifoMgrInit, [3615](#)
  - UsbFifoMgrRead, [3616](#)
  - UsbFifoMgrScan, [3616](#)
  - UsbFifoMgrWrite, [3616](#)
  - UsbFree, [3617](#)
  - UsbInit, [3617](#)
  - UsbReceiveByte, [3617](#)
  - UsbReceivePacket, [3618](#)
  - UsbReceivePipeBulkOUT, [3618](#)
  - UsbTransmitByte, [3618](#)
  - UsbTransmitPacket, [3619](#)
  - UsbTransmitPipeBulkIN, [3619](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_CubeIDE/↔
  - Boot/App/app.c
    - AppInit, [429](#)
    - AppTask, [429](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_CubeIDE/↔
  - Boot/App/app.h
    - AppInit, [471](#)
    - AppTask, [471](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_CubeIDE/↔
  - Boot/App/hooks.c
    - BackDoorEntryHook, [2099](#)
    - BackDoorInitHook, [2099](#)
    - CopInitHook, [2099](#)
    - CopServiceHook, [2100](#)
    - CpuUserProgramStartHook, [2100](#)
    - NvmDoneHook, [2100](#)
    - NvmEraseHook, [2100](#)
    - NvmInitHook, [2101](#)
    - NvmReinitHook, [2101](#)
    - NvmWriteHook, [2101](#)
    - UsbConnectHook, [2102](#)
    - UsbEnterLowPowerModeHook, [2102](#)
    - UsbLeaveLowPowerModeHook, [2102](#)
    - XcpGetSeedHook, [2102](#)
    - XcpVerifyKeyHook, [2103](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_CubeIDE/↔
  - Boot/App/led.c
    - LedBlinkExit, [2356](#)
    - LedBlinkInit, [2356](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_CubeIDE/↔
  - Boot/App/led.h
    - LedBlinkExit, [2590](#)
    - LedBlinkInit, [2590](#)
    - LedBlinkTask, [2590](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_CubeIDE/↔
  - Prog/App/app.c
    - AppInit, [430](#)
    - AppTask, [431](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_CubeIDE/↔
  - Prog/App/app.h
    - AppInit, [473](#)
    - AppTask, [473](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_CubeIDE/↔
  - Prog/App/led.c
    - LedInit, [2358](#)
    - LedToggle, [2358](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_CubeIDE/↔
  - Prog/App/led.h
    - LedInit, [2591](#)
    - LedToggle, [2592](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_GCC/Boot/hooks.↔
  - c
    - BackDoorEntryHook, [2105](#)
    - BackDoorInitHook, [2105](#)
    - CopInitHook, [2105](#)
    - CopServiceHook, [2105](#)
    - CpuUserProgramStartHook, [2106](#)
    - NvmDoneHook, [2106](#)
    - NvmEraseHook, [2106](#)
    - NvmInitHook, [2107](#)
    - NvmReinitHook, [2107](#)
    - NvmWriteHook, [2107](#)
    - UsbConnectHook, [2108](#)
    - UsbEnterLowPowerModeHook, [2108](#)
    - UsbLeaveLowPowerModeHook, [2108](#)
    - XcpGetSeedHook, [2108](#)
    - XcpVerifyKeyHook, [2109](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_GCC/Boot/led.↔
  - c
    - LedBlinkExit, [2359](#)
    - LedBlinkInit, [2359](#)
    - LedBlinkTask, [2360](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_GCC/Boot/led.↔
  - h
    - LedBlinkExit, [2593](#)
    - LedBlinkInit, [2593](#)
    - LedBlinkTask, [2593](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_GCC/Boot/main.↔
  - c
    - HAL\_MspDeInit, [2884](#)
    - HAL\_MspInit, [2884](#)
    - Init, [2884](#)
    - main, [2885](#)
    - SystemClock\_Config, [2885](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_GCC/Prog/led.↔
  - c

- LedInit, [2361](#)
- LedToggle, [2361](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_GCC/Prog/led.↔
  - h
    - LedInit, [2594](#)
    - LedToggle, [2595](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_GCC/Prog/main.↔
  - c
    - HAL\_MspDelnit, [2886](#)
    - HAL\_Msplnit, [2887](#)
    - Init, [2887](#)
    - main, [2887](#)
    - SystemClock\_Config, [2887](#)
    - VectorBase\_Config, [2888](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_IAR/Boot/hooks.↔
  - c
    - BackDoorEntryHook, [2111](#)
    - BackDoorInitHook, [2111](#)
    - CopInitHook, [2111](#)
    - CopServiceHook, [2111](#)
    - CpuUserProgramStartHook, [2112](#)
    - NvmDoneHook, [2112](#)
    - NvmEraseHook, [2112](#)
    - NvmlInitHook, [2113](#)
    - NvmReinitHook, [2113](#)
    - NvmWriteHook, [2113](#)
    - UsbConnectHook, [2114](#)
    - UsbEnterLowPowerModeHook, [2114](#)
    - UsbLeaveLowPowerModeHook, [2114](#)
    - XcpGetSeedHook, [2114](#)
    - XcpVerifyKeyHook, [2115](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_IAR/Boot/led.c
  - LedBlinkExit, [2362](#)
  - LedBlinkInit, [2362](#)
  - LedBlinkTask, [2363](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_IAR/Boot/led.h
  - LedBlinkExit, [2596](#)
  - LedBlinkInit, [2596](#)
  - LedBlinkTask, [2596](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_IAR/Boot/main.↔
  - c
    - HAL\_MspDelnit, [2889](#)
    - HAL\_Msplnit, [2889](#)
    - Init, [2889](#)
    - main, [2890](#)
    - SystemClock\_Config, [2890](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_IAR/Prog/led.c
  - LedInit, [2364](#)
  - LedToggle, [2364](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_IAR/Prog/led.h
  - LedInit, [2597](#)
  - LedToggle, [2598](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_IAR/Prog/main.↔
  - c
    - HAL\_MspDelnit, [2891](#)
    - HAL\_Msplnit, [2892](#)
    - Init, [2892](#)
    - main, [2892](#)
- SystemClock\_Config, [2892](#)
- VectorBase\_Config, [2893](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_Keil/Boot/hooks.↔
  - c
    - BackDoorEntryHook, [2117](#)
    - BackDoorInitHook, [2117](#)
    - CopInitHook, [2117](#)
    - CopServiceHook, [2117](#)
    - CpuUserProgramStartHook, [2118](#)
    - NvmDoneHook, [2118](#)
    - NvmEraseHook, [2118](#)
    - NvmlInitHook, [2119](#)
    - NvmReinitHook, [2119](#)
    - NvmWriteHook, [2119](#)
    - UsbConnectHook, [2120](#)
    - UsbEnterLowPowerModeHook, [2120](#)
    - UsbLeaveLowPowerModeHook, [2120](#)
    - XcpGetSeedHook, [2120](#)
    - XcpVerifyKeyHook, [2121](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_Keil/Boot/led.c
  - LedBlinkExit, [2365](#)
  - LedBlinkInit, [2365](#)
  - LedBlinkTask, [2366](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_Keil/Boot/led.h
  - LedBlinkExit, [2599](#)
  - LedBlinkInit, [2599](#)
  - LedBlinkTask, [2599](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_Keil/Boot/main.↔
  - c
    - HAL\_MspDelnit, [2894](#)
    - HAL\_Msplnit, [2894](#)
    - Init, [2894](#)
    - main, [2895](#)
    - SystemClock\_Config, [2895](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_Keil/Prog/led.c
  - LedInit, [2367](#)
  - LedToggle, [2367](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_Keil/Prog/led.h
  - LedInit, [2600](#)
  - LedToggle, [2601](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_Keil/Prog/main.↔
  - c
    - HAL\_MspDelnit, [2896](#)
    - HAL\_Msplnit, [2897](#)
    - Init, [2897](#)
    - main, [2897](#)
    - SystemClock\_Config, [2897](#)
    - VectorBase\_Config, [2898](#)
- address
  - tSrecLineParseObject, [379](#)
- app.c, [387](#), [388](#), [390–392](#), [394–397](#), [399–401](#), [403–405](#), [407–410](#), [412–414](#), [416–418](#), [420–423](#), [425–427](#), [429](#), [430](#)
- app.h, [431](#), [432](#), [434–438](#), [440–443](#), [445–448](#), [450–453](#), [455–458](#), [460–463](#), [465–468](#), [470–472](#)
- Applnit
  - ARMCM0\_STM32F0\_Discovery\_STM32F051\_↔
    - CubeIDE/Boot/App/app.c, [388](#)



- ARMCM0\_STM32F0\_Discovery\_STM32F051\_↔  
CubeIDE/Boot/App/app.h, [431](#)
- ARMCM0\_STM32F0\_Discovery\_STM32F051\_↔  
CubeIDE/Prog/App/app.c, [389](#)
- ARMCM0\_STM32F0\_Discovery\_STM32F051\_↔  
CubeIDE/Prog/App/app.h, [433](#)
- ARMCM0\_STM32F0\_Nucleo\_F091RC\_CubeID↔  
E/Boot/App/app.c, [390](#)
- ARMCM0\_STM32F0\_Nucleo\_F091RC\_CubeID↔  
E/Boot/App/app.h, [434](#)
- ARMCM0\_STM32F0\_Nucleo\_F091RC\_CubeID↔  
E/Prog/App/app.c, [391](#)
- ARMCM0\_STM32F0\_Nucleo\_F091RC\_CubeID↔  
E/Prog/App/app.h, [435](#)
- ARMCM0\_STM32G0\_Nucleo\_G071RB\_CubeID↔  
E/Boot/App/app.c, [393](#)
- ARMCM0\_STM32G0\_Nucleo\_G071RB\_CubeID↔  
E/Boot/App/app.h, [436](#)
- ARMCM0\_STM32G0\_Nucleo\_G071RB\_CubeID↔  
E/Prog/App/app.c, [394](#)
- ARMCM0\_STM32G0\_Nucleo\_G071RB\_CubeID↔  
E/Prog/App/app.h, [438](#)
- ARMCM33\_STM32L5\_Nucleo\_L552ZE\_CubeID↔  
E/Boot/App/app.c, [395](#)
- ARMCM33\_STM32L5\_Nucleo\_L552ZE\_CubeID↔  
E/Boot/App/app.h, [439](#)
- ARMCM33\_STM32L5\_Nucleo\_L552ZE\_CubeID↔  
E/Prog/App/app.c, [397](#)
- ARMCM33\_STM32L5\_Nucleo\_L552ZE\_CubeID↔  
E/Prog/App/app.h, [440](#)
- ARMCM3\_STM32F1\_Nucleo\_F103RB\_CubeID↔  
E/Boot/App/app.c, [398](#)
- ARMCM3\_STM32F1\_Nucleo\_F103RB\_CubeID↔  
E/Boot/App/app.h, [441](#)
- ARMCM3\_STM32F1\_Nucleo\_F103RB\_CubeID↔  
E/Prog/App/app.c, [399](#)
- ARMCM3\_STM32F1\_Nucleo\_F103RB\_CubeID↔  
E/Prog/App/app.h, [443](#)
- ARMCM3\_STM32F1\_Olimex\_STM32H103\_↔  
CubeIDE/Boot/App/app.c, [400](#)
- ARMCM3\_STM32F1\_Olimex\_STM32H103\_↔  
CubeIDE/Boot/App/app.h, [444](#)
- ARMCM3\_STM32F1\_Olimex\_STM32H103\_↔  
CubeIDE/Prog/App/app.c, [402](#)
- ARMCM3\_STM32F1\_Olimex\_STM32H103\_↔  
CubeIDE/Prog/App/app.h, [445](#)
- ARMCM3\_STM32F1\_Olimex\_STM32P103\_↔  
CubeIDE/Boot/App/app.c, [403](#)
- ARMCM3\_STM32F1\_Olimex\_STM32P103\_↔  
CubeIDE/Boot/App/app.h, [446](#)
- ARMCM3\_STM32F1\_Olimex\_STM32P103\_↔  
CubeIDE/Prog/App/app.c, [404](#)
- ARMCM3\_STM32F1\_Olimex\_STM32P103\_↔  
CubeIDE/Prog/App/app.h, [448](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_CubeID↔  
DE/Boot/App/app.c, [406](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_CubeID↔  
DE/Boot/App/app.h, [449](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_CubeID↔  
DE/Prog/App/app.c, [407](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_CubeID↔  
DE/Prog/App/app.h, [450](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_↔  
CubeIDE/Boot/App/app.c, [408](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_↔  
CubeIDE/Boot/App/app.h, [451](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_↔  
CubeIDE/Prog/App/app.c, [410](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_↔  
CubeIDE/Prog/App/app.h, [453](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_CubeID↔  
IDE/Boot/App/app.c, [411](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_CubeID↔  
IDE/Boot/App/app.h, [454](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_CubeID↔  
IDE/Prog/App/app.c, [412](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_CubeID↔  
IDE/Prog/App/app.h, [455](#)
- ARMCM4\_STM32F3\_Nucleo\_F303K8\_CubeID↔  
E/Boot/App/app.c, [413](#)
- ARMCM4\_STM32F3\_Nucleo\_F303K8\_CubeID↔  
E/Boot/App/app.h, [456](#)
- ARMCM4\_STM32F3\_Nucleo\_F303K8\_CubeID↔  
E/Prog/App/app.c, [415](#)
- ARMCM4\_STM32F3\_Nucleo\_F303K8\_CubeID↔  
E/Prog/App/app.h, [458](#)
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_CubeID↔  
E/Boot/App/app.c, [416](#)
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_CubeID↔  
E/Boot/App/app.h, [459](#)
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_CubeID↔  
E/Prog/App/app.c, [417](#)
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_CubeID↔  
E/Prog/App/app.h, [460](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_↔  
CubeIDE/Boot/App/app.c, [419](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_↔  
CubeIDE/Boot/App/app.h, [461](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_↔  
CubeIDE/Prog/App/app.c, [420](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_↔  
CubeIDE/Prog/App/app.h, [463](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_CubeID↔  
E/Boot/App/app.c, [421](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_CubeID↔  
E/Boot/App/app.h, [464](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_CubeID↔  
E/Prog/App/app.c, [423](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_CubeID↔  
E/Prog/App/app.h, [465](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_CubeID↔  
E/Boot/App/app.c, [424](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_CubeID↔  
E/Boot/App/app.h, [466](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_CubeID↔  
E/Prog/App/app.c, [425](#)

- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_CubeID↔  
E/Prog/App/app.h, [468](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_CubeID↔  
E/Boot/App/app.c, [427](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_CubeID↔  
E/Boot/App/app.h, [469](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_CubeID↔  
E/Prog/App/app.c, [428](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_CubeID↔  
E/Prog/App/app.h, [470](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_CubeID↔  
E/Boot/App/app.c, [429](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_CubeID↔  
E/Boot/App/app.h, [471](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_CubeID↔  
E/Prog/App/app.c, [430](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_CubeID↔  
E/Prog/App/app.h, [473](#)
- AppTask
  - ARMCM0\_STM32F0\_Discovery\_STM32F051↔  
CubeIDE/Boot/App/app.c, [388](#)
  - ARMCM0\_STM32F0\_Discovery\_STM32F051↔  
CubeIDE/Boot/App/app.h, [432](#)
  - ARMCM0\_STM32F0\_Discovery\_STM32F051↔  
CubeIDE/Prog/App/app.c, [389](#)
  - ARMCM0\_STM32F0\_Discovery\_STM32F051↔  
CubeIDE/Prog/App/app.h, [433](#)
  - ARMCM0\_STM32F0\_Nucleo\_F091RC\_CubeID↔  
E/Boot/App/app.c, [390](#)
  - ARMCM0\_STM32F0\_Nucleo\_F091RC\_CubeID↔  
E/Boot/App/app.h, [434](#)
  - ARMCM0\_STM32F0\_Nucleo\_F091RC\_CubeID↔  
E/Prog/App/app.c, [392](#)
  - ARMCM0\_STM32F0\_Nucleo\_F091RC\_CubeID↔  
E/Prog/App/app.h, [435](#)
  - ARMCM0\_STM32G0\_Nucleo\_G071RB\_CubeID↔  
E/Boot/App/app.c, [393](#)
  - ARMCM0\_STM32G0\_Nucleo\_G071RB\_CubeID↔  
E/Boot/App/app.h, [436](#)
  - ARMCM0\_STM32G0\_Nucleo\_G071RB\_CubeID↔  
E/Prog/App/app.c, [394](#)
  - ARMCM0\_STM32G0\_Nucleo\_G071RB\_CubeID↔  
E/Prog/App/app.h, [438](#)
  - ARMCM33\_STM32L5\_Nucleo\_L552ZE\_CubeID↔  
E/Boot/App/app.c, [396](#)
  - ARMCM33\_STM32L5\_Nucleo\_L552ZE\_CubeID↔  
E/Boot/App/app.h, [439](#)
  - ARMCM33\_STM32L5\_Nucleo\_L552ZE\_CubeID↔  
E/Prog/App/app.c, [397](#)
  - ARMCM33\_STM32L5\_Nucleo\_L552ZE\_CubeID↔  
E/Prog/App/app.h, [440](#)
  - ARMCM3\_STM32F1\_Nucleo\_F103RB\_CubeID↔  
E/Boot/App/app.c, [398](#)
  - ARMCM3\_STM32F1\_Nucleo\_F103RB\_CubeID↔  
E/Boot/App/app.h, [441](#)
  - ARMCM3\_STM32F1\_Nucleo\_F103RB\_CubeID↔  
E/Prog/App/app.c, [399](#)
  - ARMCM3\_STM32F1\_Nucleo\_F103RB\_CubeID↔  
E/Prog/App/app.h, [443](#)
  - ARMCM3\_STM32F1\_Olimex\_STM32H103↔  
CubeIDE/Boot/App/app.c, [401](#)
  - ARMCM3\_STM32F1\_Olimex\_STM32H103↔  
CubeIDE/Boot/App/app.h, [444](#)
  - ARMCM3\_STM32F1\_Olimex\_STM32H103↔  
CubeIDE/Prog/App/app.c, [402](#)
  - ARMCM3\_STM32F1\_Olimex\_STM32H103↔  
CubeIDE/Prog/App/app.h, [445](#)
  - ARMCM3\_STM32F1\_Olimex\_STM32P103↔  
CubeIDE/Boot/App/app.c, [403](#)
  - ARMCM3\_STM32F1\_Olimex\_STM32P103↔  
CubeIDE/Boot/App/app.h, [446](#)
  - ARMCM3\_STM32F1\_Olimex\_STM32P103↔  
CubeIDE/Prog/App/app.c, [405](#)
  - ARMCM3\_STM32F1\_Olimex\_STM32P103↔  
CubeIDE/Prog/App/app.h, [448](#)
  - ARMCM3\_STM32F1\_Olimexino\_STM32\_CubeID↔  
DE/Boot/App/app.c, [406](#)
  - ARMCM3\_STM32F1\_Olimexino\_STM32\_CubeID↔  
DE/Boot/App/app.h, [449](#)
  - ARMCM3\_STM32F1\_Olimexino\_STM32\_CubeID↔  
DE/Prog/App/app.c, [407](#)
  - ARMCM3\_STM32F1\_Olimexino\_STM32\_CubeID↔  
DE/Prog/App/app.h, [450](#)
  - ARMCM3\_STM32F2\_Olimex\_STM32P207↔  
CubeIDE/Boot/App/app.c, [409](#)
  - ARMCM3\_STM32F2\_Olimex\_STM32P207↔  
CubeIDE/Boot/App/app.h, [451](#)
  - ARMCM3\_STM32F2\_Olimex\_STM32P207↔  
CubeIDE/Prog/App/app.c, [410](#)
  - ARMCM3\_STM32F2\_Olimex\_STM32P207↔  
CubeIDE/Prog/App/app.h, [453](#)
  - ARMCM4\_STM32F3\_Discovery\_F303VC\_CubeID↔  
IDE/Boot/App/app.c, [411](#)
  - ARMCM4\_STM32F3\_Discovery\_F303VC\_CubeID↔  
IDE/Boot/App/app.h, [454](#)
  - ARMCM4\_STM32F3\_Discovery\_F303VC\_CubeID↔  
IDE/Prog/App/app.c, [412](#)
  - ARMCM4\_STM32F3\_Discovery\_F303VC\_CubeID↔  
IDE/Prog/App/app.h, [455](#)
  - ARMCM4\_STM32F3\_Nucleo\_F303K8\_CubeID↔  
E/Boot/App/app.c, [414](#)
  - ARMCM4\_STM32F3\_Nucleo\_F303K8\_CubeID↔  
E/Boot/App/app.h, [456](#)
  - ARMCM4\_STM32F3\_Nucleo\_F303K8\_CubeID↔  
E/Prog/App/app.c, [415](#)
  - ARMCM4\_STM32F3\_Nucleo\_F303K8\_CubeID↔  
E/Prog/App/app.h, [458](#)
  - ARMCM4\_STM32F4\_Nucleo\_F429ZI\_CubeID↔  
E/Boot/App/app.c, [416](#)
  - ARMCM4\_STM32F4\_Nucleo\_F429ZI\_CubeID↔  
E/Boot/App/app.h, [459](#)
  - ARMCM4\_STM32F4\_Nucleo\_F429ZI\_CubeID↔  
E/Prog/App/app.c, [418](#)
  - ARMCM4\_STM32F4\_Nucleo\_F429ZI\_CubeID↔  
E/Prog/App/app.h, [460](#)
  - ARMCM4\_STM32F4\_Olimex\_STM32P405↔

- CubeIDE/Boot/App/app.c, [419](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_↔
  - CubeIDE/Boot/App/app.h, [461](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_↔
  - CubeIDE/Prog/App/app.c, [420](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_↔
  - CubeIDE/Prog/App/app.h, [463](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_CubeID↔
  - E/Boot/App/app.c, [422](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_CubeID↔
  - E/Boot/App/app.h, [464](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_CubeID↔
  - E/Prog/App/app.c, [423](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_CubeID↔
  - E/Prog/App/app.h, [465](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_CubeID↔
  - E/Boot/App/app.c, [424](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_CubeID↔
  - E/Boot/App/app.h, [466](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_CubeID↔
  - E/Prog/App/app.c, [425](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_CubeID↔
  - E/Prog/App/app.h, [468](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_CubeID↔
  - E/Boot/App/app.c, [427](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_CubeID↔
  - E/Boot/App/app.h, [469](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_CubeID↔
  - E/Prog/App/app.c, [428](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_CubeID↔
  - E/Prog/App/app.h, [470](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_CubeID↔
  - E/Boot/App/app.c, [429](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_CubeID↔
  - E/Boot/App/app.h, [471](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_CubeID↔
  - E/Prog/App/app.c, [431](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_CubeID↔
  - E/Prog/App/app.h, [473](#)
- AssertFailure
  - asserts.c, [474](#)
  - asserts.h, [475](#)
- asserts.c, [474](#)
  - AssertFailure, [474](#)
- asserts.h, [475](#)
  - AssertFailure, [475](#)
- BDM\_DEBUGGING\_ENABLED
  - HCS12\_DevKit\_S12G128\_CodeWarrior/Prog/header.↔
    - h, [1618](#)
  - HCS12\_Evbplus\_Dragon12p\_CodeWarrior/↔
    - Prog/header.h, [1619](#)
- BackDoorCheck
  - backdoor.c, [476](#)
  - backdoor.h, [478](#)
- BackDoorEntryHook
  - \_template/Boot/hooks.c, [1621](#)
- ARMCM0\_STM32F0\_Discovery\_STM32F051\_↔
  - CubeIDE/Boot/App/hooks.c, [1626](#)
- ARMCM0\_STM32F0\_Discovery\_STM32F051\_↔
  - GCC/Boot/hooks.c, [1631](#)
- ARMCM0\_STM32F0\_Discovery\_STM32F051\_↔
  - AR/Boot/hooks.c, [1636](#)
- ARMCM0\_STM32F0\_Discovery\_STM32F051\_↔
  - Keil/Boot/hooks.c, [1641](#)
- ARMCM0\_STM32F0\_Nucleo\_F091RC\_CubeID↔
  - E/Boot/App/hooks.c, [1646](#)
- ARMCM0\_STM32F0\_Nucleo\_F091RC\_GCC/↔
  - Boot/hooks.c, [1651](#)
- ARMCM0\_STM32F0\_Nucleo\_F091RC\_IAR/↔
  - Boot/hooks.c, [1656](#)
- ARMCM0\_STM32F0\_Nucleo\_F091RC\_Keil/↔
  - Boot/hooks.c, [1661](#)
- ARMCM0\_STM32G0\_Nucleo\_G071RB\_CubeID↔
  - E/Boot/App/hooks.c, [1666](#)
- ARMCM0\_STM32G0\_Nucleo\_G071RB\_GCC/↔
  - Boot/hooks.c, [1671](#)
- ARMCM0\_STM32G0\_Nucleo\_G071RB\_IAR/↔
  - Boot/hooks.c, [1676](#)
- ARMCM0\_STM32G0\_Nucleo\_G071RB\_Keil/↔
  - Boot/hooks.c, [1681](#)
- ARMCM0\_XMC1\_XMC1400\_Boot\_Kit\_GCC/↔
  - Boot/hooks.c, [1686](#)
- ARMCM0\_XMC1\_XMC1400\_Boot\_Kit\_IAR/↔
  - Boot/hooks.c, [1691](#)
- ARMCM33\_STM32L5\_Nucleo\_L552ZE\_CubeID↔
  - E/Boot/App/hooks.c, [1696](#)
- ARMCM33\_STM32L5\_Nucleo\_L552ZE\_GCC/↔
  - Boot/hooks.c, [1702](#)
- ARMCM33\_STM32L5\_Nucleo\_L552ZE\_IAR/↔
  - Boot/hooks.c, [1708](#)
- ARMCM33\_STM32L5\_Nucleo\_L552ZE\_Keil/↔
  - Boot/hooks.c, [1714](#)
- ARMCM3\_EFM32\_Olimex\_EM32G880F128ST↔
  - K\_GCC/Boot/hooks.c, [1720](#)
- ARMCM3\_EFM32\_Olimex\_EM32G880F128ST↔
  - K\_IAR/Boot/hooks.c, [1725](#)
- ARMCM3\_LM3S\_EK\_LM3S6965\_GCC/Boot/hooks.↔
  - c, [1730](#)
- ARMCM3\_LM3S\_EK\_LM3S6965\_IAR/Boot/hooks.↔
  - c, [1738](#)
- ARMCM3\_LM3S\_EK\_LM3S8962\_GCC/Boot/hooks.↔
  - c, [1745](#)
- ARMCM3\_LM3S\_EK\_LM3S8962\_IAR/Boot/hooks.↔
  - c, [1751](#)
- ARMCM3\_STM32F1\_Nucleo\_F103RB\_CubeID↔
  - E/Boot/App/hooks.c, [1756](#)
- ARMCM3\_STM32F1\_Nucleo\_F103RB\_GCC/↔
  - Boot/hooks.c, [1761](#)
- ARMCM3\_STM32F1\_Nucleo\_F103RB\_IAR/↔
  - Boot/hooks.c, [1766](#)
- ARMCM3\_STM32F1\_Nucleo\_F103RB\_Keil/↔
  - Boot/hooks.c, [1771](#)
- ARMCM3\_STM32F1\_Olimex\_STM32H103\_↔
  - CubeIDE/Boot/App/hooks.c, [1776](#)
- ARMCM3\_STM32F1\_Olimex\_STM32H103\_GC↔
  - C/Boot/hooks.c, [1782](#)

- ARMCM3\_STM32F1\_Olimex\_STM32H103\_IA↔  
R/Boot/hooks.c, [1788](#)
- ARMCM3\_STM32F1\_Olimex\_STM32H103\_↔  
Keil/Boot/hooks.c, [1794](#)
- ARMCM3\_STM32F1\_Olimex\_STM32P103\_↔  
CubeIDE/Boot/App/hooks.c, [1800](#)
- ARMCM3\_STM32F1\_Olimex\_STM32P103\_GC↔  
C/Boot/hooks.c, [1808](#)
- ARMCM3\_STM32F1\_Olimex\_STM32P103\_IA↔  
R/Boot/hooks.c, [1815](#)
- ARMCM3\_STM32F1\_Olimex\_STM32P103\_Keil/↔  
Boot/hooks.c, [1823](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_CubeI↔  
DE/Boot/App/hooks.c, [1831](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_GCC/↔  
Boot/hooks.c, [1839](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_IAR/↔  
Boot/hooks.c, [1848](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_Keil/↔  
Boot/hooks.c, [1857](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_↔  
CubeIDE/Boot/App/hooks.c, [1866](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_GC↔  
C/Boot/hooks.c, [1874](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_IA↔  
R/Boot/hooks.c, [1881](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_Keil/↔  
Boot/hooks.c, [1889](#)
- ARMCM4\_S32K14\_S32K144EVB\_GCC/Boot/hooks.↔  
c, [1896](#)
- ARMCM4\_S32K14\_S32K144EVB\_IAR/Boot/hooks.↔  
c, [1902](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_Cube↔  
IDE/Boot/App/hooks.c, [1907](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_GCC/↔  
Boot/hooks.c, [1913](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_IAR/↔  
Boot/hooks.c, [1919](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_Keil/↔  
Boot/hooks.c, [1925](#)
- ARMCM4\_STM32F3\_Nucleo\_F303K8\_CubeID↔  
E/Boot/App/hooks.c, [1931](#)
- ARMCM4\_STM32F3\_Nucleo\_F303K8\_GCC/↔  
Boot/hooks.c, [1936](#)
- ARMCM4\_STM32F3\_Nucleo\_F303K8\_IAR/↔  
Boot/hooks.c, [1941](#)
- ARMCM4\_STM32F3\_Nucleo\_F303K8\_Keil/↔  
Boot/hooks.c, [1946](#)
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_CubeID↔  
E/Boot/App/hooks.c, [1951](#)
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_GCC/↔  
Boot/hooks.c, [1957](#)
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_IAR/↔  
Boot/hooks.c, [1963](#)
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_Keil/↔  
Boot/hooks.c, [1969](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_↔  
CubeIDE/Boot/App/hooks.c, [1975](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_GC↔  
C/Boot/hooks.c, [1984](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_IA↔  
R/Boot/hooks.c, [1992](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_Keil/↔  
Boot/hooks.c, [2001](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_CubeID↔  
E/Boot/App/hooks.c, [2010](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_GCC/↔  
Boot/hooks.c, [2015](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_IAR/↔  
Boot/hooks.c, [2020](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_Keil/↔  
Boot/hooks.c, [2025](#)
- ARMCM4\_TM4C\_DK\_TM4C123G\_IAR/Boot/hooks.↔  
c, [2031](#)
- ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_GCC/↔  
Boot/hooks.c, [2039](#)
- ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_IAR/↔  
Boot/hooks.c, [2047](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_CubeID↔  
E/Boot/App/hooks.c, [2054](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_GCC/↔  
Boot/hooks.c, [2061](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_IAR/↔  
Boot/hooks.c, [2067](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_Keil/↔  
Boot/hooks.c, [2073](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_CubeID↔  
E/Boot/App/hooks.c, [2079](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_GCC/↔  
Boot/hooks.c, [2084](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_IAR/↔  
Boot/hooks.c, [2089](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_Keil/↔  
Boot/hooks.c, [2094](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_CubeID↔  
E/Boot/App/hooks.c, [2099](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_GCC/↔  
Boot/hooks.c, [2105](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_IAR/↔  
Boot/hooks.c, [2111](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_Keil/↔  
Boot/hooks.c, [2117](#)
- backdoor.c, [477](#)
- HCS12\_DevKit\_S12G128\_CodeWarrior/Boot/hooks.↔  
c, [2123](#)
- HCS12\_Evbplus\_Dragon12p\_CodeWarrior/↔  
Boot/hooks.c, [2128](#)
- BackDoorInit  
backdoor.c, [477](#)  
backdoor.h, [479](#)
- BackDoorInitHook  
\_template/Boot/hooks.c, [1621](#)
- ARMCM0\_STM32F0\_Discovery\_STM32F051\_↔  
CubeIDE/Boot/App/hooks.c, [1626](#)
- ARMCM0\_STM32F0\_Discovery\_STM32F051\_↔  
GCC/Boot/hooks.c, [1631](#)

- ARMCM0\_STM32F0\_Discovery\_STM32F051\_I↔  
AR/Boot/hooks.c, [1636](#)
- ARMCM0\_STM32F0\_Discovery\_STM32F051↔  
Keil/Boot/hooks.c, [1641](#)
- ARMCM0\_STM32F0\_Nucleo\_F091RC\_CubeID↔  
E/Boot/App/hooks.c, [1646](#)
- ARMCM0\_STM32F0\_Nucleo\_F091RC\_GCC↔  
Boot/hooks.c, [1651](#)
- ARMCM0\_STM32F0\_Nucleo\_F091RC\_IAR↔  
Boot/hooks.c, [1656](#)
- ARMCM0\_STM32F0\_Nucleo\_F091RC\_Keil↔  
Boot/hooks.c, [1661](#)
- ARMCM0\_STM32G0\_Nucleo\_G071RB\_CubeID↔  
E/Boot/App/hooks.c, [1666](#)
- ARMCM0\_STM32G0\_Nucleo\_G071RB\_GCC↔  
Boot/hooks.c, [1671](#)
- ARMCM0\_STM32G0\_Nucleo\_G071RB\_IAR↔  
Boot/hooks.c, [1676](#)
- ARMCM0\_STM32G0\_Nucleo\_G071RB\_Keil↔  
Boot/hooks.c, [1681](#)
- ARMCM0\_XMC1\_XMC1400\_Boot\_Kit\_GCC↔  
Boot/hooks.c, [1686](#)
- ARMCM0\_XMC1\_XMC1400\_Boot\_Kit\_IAR↔  
Boot/hooks.c, [1691](#)
- ARMCM33\_STM32L5\_Nucleo\_L552ZE\_CubeID↔  
E/Boot/App/hooks.c, [1696](#)
- ARMCM33\_STM32L5\_Nucleo\_L552ZE\_GCC↔  
Boot/hooks.c, [1702](#)
- ARMCM33\_STM32L5\_Nucleo\_L552ZE\_IAR↔  
Boot/hooks.c, [1708](#)
- ARMCM33\_STM32L5\_Nucleo\_L552ZE\_Keil↔  
Boot/hooks.c, [1714](#)
- ARMCM3\_EFM32\_Olimex\_EM32G880F128ST↔  
K\_GCC/Boot/hooks.c, [1720](#)
- ARMCM3\_EFM32\_Olimex\_EM32G880F128ST↔  
K\_IAR/Boot/hooks.c, [1725](#)
- ARMCM3\_LM3S\_EK\_LM3S6965\_GCC/Boot/hooks.↔  
c, [1731](#)
- ARMCM3\_LM3S\_EK\_LM3S6965\_IAR/Boot/hooks.↔  
c, [1738](#)
- ARMCM3\_LM3S\_EK\_LM3S8962\_GCC/Boot/hooks.↔  
c, [1746](#)
- ARMCM3\_LM3S\_EK\_LM3S8962\_IAR/Boot/hooks.↔  
c, [1751](#)
- ARMCM3\_STM32F1\_Nucleo\_F103RB\_CubeID↔  
E/Boot/App/hooks.c, [1756](#)
- ARMCM3\_STM32F1\_Nucleo\_F103RB\_GCC↔  
Boot/hooks.c, [1761](#)
- ARMCM3\_STM32F1\_Nucleo\_F103RB\_IAR↔  
Boot/hooks.c, [1766](#)
- ARMCM3\_STM32F1\_Nucleo\_F103RB\_Keil↔  
Boot/hooks.c, [1771](#)
- ARMCM3\_STM32F1\_Olimex\_STM32H103↔  
CubeIDE/Boot/App/hooks.c, [1776](#)
- ARMCM3\_STM32F1\_Olimex\_STM32H103\_GC↔  
C/Boot/hooks.c, [1782](#)
- ARMCM3\_STM32F1\_Olimex\_STM32H103\_IA↔  
R/Boot/hooks.c, [1788](#)
- ARMCM3\_STM32F1\_Olimex\_STM32H103↔  
Keil/Boot/hooks.c, [1794](#)
- ARMCM3\_STM32F1\_Olimex\_STM32P103↔  
CubeIDE/Boot/App/hooks.c, [1800](#)
- ARMCM3\_STM32F1\_Olimex\_STM32P103\_GC↔  
C/Boot/hooks.c, [1808](#)
- ARMCM3\_STM32F1\_Olimex\_STM32P103\_IA↔  
R/Boot/hooks.c, [1816](#)
- ARMCM3\_STM32F1\_Olimex\_STM32P103\_Keil↔  
Boot/hooks.c, [1823](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_Cube↔  
DE/Boot/App/hooks.c, [1831](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_GCC↔  
Boot/hooks.c, [1840](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_IAR↔  
Boot/hooks.c, [1849](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_Keil↔  
Boot/hooks.c, [1858](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207↔  
CubeIDE/Boot/App/hooks.c, [1866](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_GC↔  
C/Boot/hooks.c, [1874](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_IA↔  
R/Boot/hooks.c, [1882](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_Keil↔  
Boot/hooks.c, [1889](#)
- ARMCM4\_S32K14\_S32K144EVB\_GCC/Boot/hooks.↔  
c, [1897](#)
- ARMCM4\_S32K14\_S32K144EVB\_IAR/Boot/hooks.↔  
c, [1902](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_Cube↔  
IDE/Boot/App/hooks.c, [1907](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_GCC↔  
Boot/hooks.c, [1913](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_IAR↔  
Boot/hooks.c, [1919](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_Keil↔  
Boot/hooks.c, [1925](#)
- ARMCM4\_STM32F3\_Nucleo\_F303K8\_CubeID↔  
E/Boot/App/hooks.c, [1931](#)
- ARMCM4\_STM32F3\_Nucleo\_F303K8\_GCC↔  
Boot/hooks.c, [1936](#)
- ARMCM4\_STM32F3\_Nucleo\_F303K8\_IAR↔  
Boot/hooks.c, [1941](#)
- ARMCM4\_STM32F3\_Nucleo\_F303K8\_Keil↔  
Boot/hooks.c, [1946](#)
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_CubeID↔  
E/Boot/App/hooks.c, [1951](#)
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_GCC↔  
Boot/hooks.c, [1957](#)
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_IAR↔  
Boot/hooks.c, [1963](#)
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_Keil↔  
Boot/hooks.c, [1969](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405↔  
CubeIDE/Boot/App/hooks.c, [1975](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_GC↔  
C/Boot/hooks.c, [1984](#)



- ARMCM4\_STM32F4\_Olimex\_STM32P405\_IAR/↔  
R/Boot/hooks.c, [1993](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_Keil/↔  
Boot/hooks.c, [2002](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_CubeID↔  
E/Boot/App/hooks.c, [2010](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_GCC/↔  
Boot/hooks.c, [2015](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_IAR/↔  
Boot/hooks.c, [2020](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_Keil/↔  
Boot/hooks.c, [2025](#)
- ARMCM4\_TM4C\_DK\_TM4C123G\_IAR/Boot/hooks.↔  
c, [2031](#)
- ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_GCC/↔  
Boot/hooks.c, [2039](#)
- ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_IAR/↔  
Boot/hooks.c, [2047](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_CubeID↔  
E/Boot/App/hooks.c, [2055](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_GCC/↔  
Boot/hooks.c, [2061](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_IAR/↔  
Boot/hooks.c, [2067](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_Keil/↔  
Boot/hooks.c, [2073](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_CubeID↔  
E/Boot/App/hooks.c, [2079](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_GCC/↔  
Boot/hooks.c, [2084](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_IAR/↔  
Boot/hooks.c, [2089](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_Keil/↔  
Boot/hooks.c, [2094](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_CubeID↔  
E/Boot/App/hooks.c, [2099](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_GCC/↔  
Boot/hooks.c, [2105](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_IAR/↔  
Boot/hooks.c, [2111](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_Keil/↔  
Boot/hooks.c, [2117](#)
- backdoor.c, [477](#)
- HCS12\_DevKit\_S12G128\_CodeWarrior/Boot/hooks.↔  
c, [2123](#)
- HCS12\_Evbplus\_Dragon12p\_CodeWarrior/↔  
Boot/hooks.c, [2128](#)
- backdoor.c, [476](#)
  - BackDoorCheck, [476](#)
  - BackDoorEntryHook, [477](#)
  - BackDoorInit, [477](#)
  - BackDoorInitHook, [477](#)
- backdoor.h, [478](#)
  - BackDoorCheck, [478](#)
  - BackDoorInit, [479](#)
- bank\_num
  - tFlashSector, [376](#)
- blockInfo
  - \_template/flash.c, [1275](#)
  - ARMCM0\_S32K11/flash.c, [1284](#)
  - ARMCM0\_STM32F0/flash.c, [1292](#)
  - ARMCM0\_STM32G0/flash.c, [1301](#)
  - ARMCM0\_XMC1/flash.c, [1310](#)
  - ARMCM33\_STM32L5/flash.c, [1319](#)
  - ARMCM3\_EFM32/flash.c, [1328](#)
  - ARMCM3\_LM3S/flash.c, [1337](#)
  - ARMCM3\_STM32F1/flash.c, [1344](#)
  - ARMCM3\_STM32F2/flash.c, [1352](#)
  - ARMCM4\_S32K14/flash.c, [1361](#)
  - ARMCM4\_STM32F3/flash.c, [1368](#)
  - ARMCM4\_STM32F4/flash.c, [1376](#)
  - ARMCM4\_STM32L4/flash.c, [1385](#)
  - ARMCM4\_TM4C/flash.c, [1394](#)
  - ARMCM4\_XMC4/flash.c, [1403](#)
  - ARMCM7\_STM32F7/flash.c, [1412](#)
  - ARMCM7\_STM32H7/flash.c, [1420](#)
  - flash\_ecc.c, [1511](#)
  - HCS12/flash.c, [1431](#)
- blt\_addr
  - \_template/types.h, [3530](#)
  - ARMCM0\_S32K11/types.h, [3532](#)
  - ARMCM0\_STM32F0/types.h, [3534](#)
  - ARMCM0\_STM32G0/types.h, [3536](#)
  - ARMCM0\_XMC1/types.h, [3538](#)
  - ARMCM33\_STM32L5/types.h, [3540](#)
  - ARMCM3\_EFM32/types.h, [3542](#)
  - ARMCM3\_LM3S/types.h, [3544](#)
  - ARMCM3\_STM32F1/types.h, [3546](#)
  - ARMCM3\_STM32F2/types.h, [3548](#)
  - ARMCM4\_S32K14/types.h, [3550](#)
  - ARMCM4\_STM32F3/types.h, [3552](#)
  - ARMCM4\_STM32F4/types.h, [3554](#)
  - ARMCM4\_STM32L4/types.h, [3556](#)
  - ARMCM4\_TM4C/types.h, [3558](#)
  - ARMCM4\_XMC4/types.h, [3560](#)
  - ARMCM7\_STM32F7/types.h, [3562](#)
  - ARMCM7\_STM32H7/types.h, [3564](#)
  - HCS12/types.h, [3566](#)
- blt\_bool
  - \_template/types.h, [3530](#)
  - ARMCM0\_S32K11/types.h, [3532](#)
  - ARMCM0\_STM32F0/types.h, [3534](#)
  - ARMCM0\_STM32G0/types.h, [3536](#)
  - ARMCM0\_XMC1/types.h, [3538](#)
  - ARMCM33\_STM32L5/types.h, [3540](#)
  - ARMCM3\_EFM32/types.h, [3542](#)
  - ARMCM3\_LM3S/types.h, [3544](#)
  - ARMCM3\_STM32F1/types.h, [3546](#)
  - ARMCM3\_STM32F2/types.h, [3548](#)
  - ARMCM4\_S32K14/types.h, [3550](#)
  - ARMCM4\_STM32F3/types.h, [3552](#)
  - ARMCM4\_STM32F4/types.h, [3554](#)
  - ARMCM4\_STM32L4/types.h, [3556](#)
  - ARMCM4\_TM4C/types.h, [3558](#)
  - ARMCM4\_XMC4/types.h, [3560](#)
  - ARMCM7\_STM32F7/types.h, [3562](#)

- ARMCM7\_STM32H7/types.h, [3564](#)
- HCS12/types.h, [3566](#)
- blt\_char
  - \_template/types.h, [3530](#)
  - ARMCM0\_S32K11/types.h, [3532](#)
  - ARMCM0\_STM32F0/types.h, [3534](#)
  - ARMCM0\_STM32G0/types.h, [3536](#)
  - ARMCM0\_XMC1/types.h, [3538](#)
  - ARMCM33\_STM32L5/types.h, [3540](#)
  - ARMCM3\_EFM32/types.h, [3542](#)
  - ARMCM3\_LM3S/types.h, [3544](#)
  - ARMCM3\_STM32F1/types.h, [3546](#)
  - ARMCM3\_STM32F2/types.h, [3548](#)
  - ARMCM4\_S32K14/types.h, [3550](#)
  - ARMCM4\_STM32F3/types.h, [3552](#)
  - ARMCM4\_STM32F4/types.h, [3554](#)
  - ARMCM4\_STM32L4/types.h, [3556](#)
  - ARMCM4\_TM4C/types.h, [3558](#)
  - ARMCM4\_XMC4/types.h, [3560](#)
  - ARMCM7\_STM32F7/types.h, [3562](#)
  - ARMCM7\_STM32H7/types.h, [3564](#)
  - HCS12/types.h, [3566](#)
- blt\_conf.h, [479](#), [480](#), [482–484](#), [486](#), [487](#), [489](#), [490](#), [492–496](#), [498](#), [500](#), [501](#), [503](#), [504](#), [506](#), [507](#), [509](#), [511](#), [513](#), [514](#), [516–518](#), [520–523](#), [525](#), [526](#), [528](#), [530](#), [531](#), [533](#), [535](#), [537](#), [539](#), [541](#), [543](#), [545](#), [546](#), [548](#), [550–555](#), [557](#), [558](#), [560](#), [561](#), [564](#), [566](#), [569](#), [571](#), [573](#), [575](#), [577](#), [578](#), [580](#), [582](#), [583](#), [585](#), [586](#), [589](#), [591](#), [593](#), [594](#), [596](#), [598](#), [600](#), [602](#), [604](#), [606](#), [607](#), [609](#), [611](#), [612](#), [614](#)
- blt\_int16s
  - \_template/types.h, [3531](#)
  - ARMCM0\_S32K11/types.h, [3532](#)
  - ARMCM0\_STM32F0/types.h, [3534](#)
  - ARMCM0\_STM32G0/types.h, [3536](#)
  - ARMCM0\_XMC1/types.h, [3539](#)
  - ARMCM33\_STM32L5/types.h, [3540](#)
  - ARMCM3\_EFM32/types.h, [3543](#)
  - ARMCM3\_LM3S/types.h, [3544](#)
  - ARMCM3\_STM32F1/types.h, [3546](#)
  - ARMCM3\_STM32F2/types.h, [3548](#)
  - ARMCM4\_S32K14/types.h, [3550](#)
  - ARMCM4\_STM32F3/types.h, [3552](#)
  - ARMCM4\_STM32F4/types.h, [3554](#)
  - ARMCM4\_STM32L4/types.h, [3556](#)
  - ARMCM4\_TM4C/types.h, [3558](#)
  - ARMCM4\_XMC4/types.h, [3560](#)
  - ARMCM7\_STM32F7/types.h, [3562](#)
  - ARMCM7\_STM32H7/types.h, [3564](#)
  - HCS12/types.h, [3566](#)
- blt\_int16u
  - \_template/types.h, [3531](#)
  - ARMCM0\_S32K11/types.h, [3533](#)
  - ARMCM0\_STM32F0/types.h, [3535](#)
  - ARMCM0\_STM32G0/types.h, [3537](#)
  - ARMCM0\_XMC1/types.h, [3539](#)
  - ARMCM33\_STM32L5/types.h, [3541](#)
  - ARMCM3\_EFM32/types.h, [3543](#)
  - ARMCM3\_LM3S/types.h, [3545](#)
  - ARMCM3\_STM32F1/types.h, [3547](#)
  - ARMCM3\_STM32F2/types.h, [3549](#)
  - ARMCM4\_S32K14/types.h, [3551](#)
  - ARMCM4\_STM32F3/types.h, [3553](#)
  - ARMCM4\_STM32F4/types.h, [3555](#)
  - ARMCM4\_STM32L4/types.h, [3557](#)
  - ARMCM4\_TM4C/types.h, [3559](#)
  - ARMCM4\_XMC4/types.h, [3561](#)
  - ARMCM7\_STM32F7/types.h, [3563](#)
  - ARMCM7\_STM32H7/types.h, [3565](#)
  - HCS12/types.h, [3567](#)
- blt\_int32s
  - \_template/types.h, [3531](#)
  - ARMCM0\_S32K11/types.h, [3533](#)
  - ARMCM0\_STM32F0/types.h, [3535](#)
  - ARMCM0\_STM32G0/types.h, [3537](#)
  - ARMCM0\_XMC1/types.h, [3539](#)
  - ARMCM33\_STM32L5/types.h, [3541](#)
  - ARMCM3\_EFM32/types.h, [3543](#)
  - ARMCM3\_LM3S/types.h, [3545](#)
  - ARMCM3\_STM32F1/types.h, [3547](#)
  - ARMCM3\_STM32F2/types.h, [3549](#)
  - ARMCM4\_S32K14/types.h, [3551](#)
  - ARMCM4\_STM32F3/types.h, [3553](#)
  - ARMCM4\_STM32F4/types.h, [3555](#)
  - ARMCM4\_STM32L4/types.h, [3557](#)
  - ARMCM4\_TM4C/types.h, [3559](#)
  - ARMCM4\_XMC4/types.h, [3561](#)
  - ARMCM7\_STM32F7/types.h, [3563](#)
  - ARMCM7\_STM32H7/types.h, [3565](#)
  - HCS12/types.h, [3567](#)
- blt\_int32u
  - \_template/types.h, [3531](#)
  - ARMCM0\_S32K11/types.h, [3533](#)
  - ARMCM0\_STM32F0/types.h, [3535](#)
  - ARMCM0\_STM32G0/types.h, [3537](#)
  - ARMCM0\_XMC1/types.h, [3539](#)
  - ARMCM33\_STM32L5/types.h, [3541](#)
  - ARMCM3\_EFM32/types.h, [3543](#)
  - ARMCM3\_LM3S/types.h, [3545](#)
  - ARMCM3\_STM32F1/types.h, [3547](#)
  - ARMCM3\_STM32F2/types.h, [3549](#)
  - ARMCM4\_S32K14/types.h, [3551](#)
  - ARMCM4\_STM32F3/types.h, [3553](#)
  - ARMCM4\_STM32F4/types.h, [3555](#)
  - ARMCM4\_STM32L4/types.h, [3557](#)
  - ARMCM4\_TM4C/types.h, [3559](#)
  - ARMCM4\_XMC4/types.h, [3561](#)
  - ARMCM7\_STM32F7/types.h, [3563](#)
  - ARMCM7\_STM32H7/types.h, [3565](#)
  - HCS12/types.h, [3567](#)
- blt\_int64s
  - ARMCM0\_STM32G0/types.h, [3537](#)
  - ARMCM33\_STM32L5/types.h, [3541](#)
- blt\_int64u
  - ARMCM0\_STM32G0/types.h, [3537](#)

- ARMCM33\_STM32L5/types.h, 3541
- blt\_int8s
  - \_template/types.h, 3531
  - ARMCM0\_S32K11/types.h, 3533
  - ARMCM0\_STM32F0/types.h, 3535
  - ARMCM0\_STM32G0/types.h, 3537
  - ARMCM0\_XMC1/types.h, 3539
  - ARMCM33\_STM32L5/types.h, 3541
  - ARMCM3\_EFM32/types.h, 3543
  - ARMCM3\_LM3S/types.h, 3545
  - ARMCM3\_STM32F1/types.h, 3547
  - ARMCM3\_STM32F2/types.h, 3549
  - ARMCM4\_S32K14/types.h, 3551
  - ARMCM4\_STM32F3/types.h, 3553
  - ARMCM4\_STM32F4/types.h, 3555
  - ARMCM4\_STM32L4/types.h, 3557
  - ARMCM4\_TM4C/types.h, 3559
  - ARMCM4\_XMC4/types.h, 3561
  - ARMCM7\_STM32F7/types.h, 3563
  - ARMCM7\_STM32H7/types.h, 3565
  - HCS12/types.h, 3567
- blt\_int8u
  - \_template/types.h, 3531
  - ARMCM0\_S32K11/types.h, 3533
  - ARMCM0\_STM32F0/types.h, 3535
  - ARMCM0\_STM32G0/types.h, 3537
  - ARMCM0\_XMC1/types.h, 3539
  - ARMCM33\_STM32L5/types.h, 3541
  - ARMCM3\_EFM32/types.h, 3543
  - ARMCM3\_LM3S/types.h, 3545
  - ARMCM3\_STM32F1/types.h, 3547
  - ARMCM3\_STM32F2/types.h, 3549
  - ARMCM4\_S32K14/types.h, 3551
  - ARMCM4\_STM32F3/types.h, 3553
  - ARMCM4\_STM32F4/types.h, 3555
  - ARMCM4\_STM32L4/types.h, 3557
  - ARMCM4\_TM4C/types.h, 3559
  - ARMCM4\_XMC4/types.h, 3561
  - ARMCM7\_STM32F7/types.h, 3563
  - ARMCM7\_STM32H7/types.h, 3565
  - HCS12/types.h, 3567
- boot.c, 615, 619, 622, 625, 628, 631, 636, 641, 646, 651, 654, 657, 660, 663, 666, 670, 675, 680, 685, 690, 693, 696, 698, 701, 705, 708, 712, 715, 718, 721, 726, 731, 736, 741, 744, 748, 752, 756, 761, 766, 771, 776, 783, 790, 795, 800, 805, 810, 815, 820, 825, 830, 835, 840, 845, 850, 855, 860, 865, 869, 872, 875, 879, 884, 889, 894, 899, 902, 905, 908, 911, 916, 921, 926, 931, 936
- boot.h, 937, 939, 941, 942, 944, 946, 947, 949, 951, 952, 954, 956, 957, 959, 961, 962, 964, 966, 967, 969, 970, 972, 973, 975, 976, 978, 980, 981, 983, 985, 986, 988, 990, 991, 993, 995, 996, 998, 1000, 1001, 1003, 1005, 1006, 1008, 1010, 1011, 1013, 1015, 1016, 1018, 1020, 1021, 1023, 1025, 1026, 1028, 1030, 1031, 1033, 1035, 1036, 1038, 1040, 1041, 1043, 1045, 1046, 1048, 1050, 1051, 1053, 1055, 1056, 1058, 1060, 1061
- BootActivate
  - Demo/\_template/Prog/boot.c, 616
  - Demo/\_template/Prog/boot.h, 938
  - Demo/ARMCM0\_STM32F0\_Discovery\_STM32←  
F051\_CubeIDE/Prog/App/boot.c, 620
  - Demo/ARMCM0\_STM32F0\_Discovery\_STM32←  
F051\_CubeIDE/Prog/App/boot.h, 940
  - Demo/ARMCM0\_STM32F0\_Discovery\_STM32←  
F051\_GCC/Prog/boot.c, 623
  - Demo/ARMCM0\_STM32F0\_Discovery\_STM32←  
F051\_GCC/Prog/boot.h, 941
  - Demo/ARMCM0\_STM32F0\_Discovery\_STM32←  
F051\_IAR/Prog/boot.c, 626
  - Demo/ARMCM0\_STM32F0\_Discovery\_STM32←  
F051\_IAR/Prog/boot.h, 943
  - Demo/ARMCM0\_STM32F0\_Discovery\_STM32←  
F051\_Keil/Prog/boot.c, 629
  - Demo/ARMCM0\_STM32F0\_Discovery\_STM32←  
F051\_Keil/Prog/boot.h, 945
  - Demo/ARMCM0\_STM32F0\_Nucleo\_F091RC←  
CubeIDE/Prog/App/boot.c, 632
  - Demo/ARMCM0\_STM32F0\_Nucleo\_F091RC←  
CubeIDE/Prog/App/boot.h, 946
  - Demo/ARMCM0\_STM32F0\_Nucleo\_F091RC\_G←  
CC/Prog/boot.c, 637
  - Demo/ARMCM0\_STM32F0\_Nucleo\_F091RC\_G←  
CC/Prog/boot.h, 948
  - Demo/ARMCM0\_STM32F0\_Nucleo\_F091RC\_I←  
AR/Prog/boot.c, 642
  - Demo/ARMCM0\_STM32F0\_Nucleo\_F091RC\_I←  
AR/Prog/boot.h, 950
  - Demo/ARMCM0\_STM32F0\_Nucleo\_F091RC←  
Keil/Prog/boot.c, 647
  - Demo/ARMCM0\_STM32F0\_Nucleo\_F091RC←  
Keil/Prog/boot.h, 951
  - Demo/ARMCM0\_STM32G0\_Nucleo\_G071RB←  
CubeIDE/Prog/App/boot.c, 652
  - Demo/ARMCM0\_STM32G0\_Nucleo\_G071RB←  
CubeIDE/Prog/App/boot.h, 953
  - Demo/ARMCM0\_STM32G0\_Nucleo\_G071RB←  
GCC/Prog/boot.c, 655
  - Demo/ARMCM0\_STM32G0\_Nucleo\_G071RB←  
GCC/Prog/boot.h, 955
  - Demo/ARMCM0\_STM32G0\_Nucleo\_G071RB\_I←  
AR/Prog/boot.c, 658
  - Demo/ARMCM0\_STM32G0\_Nucleo\_G071RB\_I←  
AR/Prog/boot.h, 956
  - Demo/ARMCM0\_STM32G0\_Nucleo\_G071RB←  
Keil/Prog/boot.c, 661
  - Demo/ARMCM0\_STM32G0\_Nucleo\_G071RB←  
Keil/Prog/boot.h, 958
  - Demo/ARMCM0\_XMC1\_XMC1400\_Boot\_Kit\_G←  
CC/Prog/boot.c, 664
  - Demo/ARMCM0\_XMC1\_XMC1400\_Boot\_Kit\_G←  
CC/Prog/boot.h, 960
  - Demo/ARMCM0\_XMC1\_XMC1400\_Boot\_Kit\_IA←



- R/Prog/boot.c, [667](#)
- Demo/ARMCM0\_XMC1\_XMC1400\_Boot\_Kit\_IAR/Prog/boot.h, [961](#)
- Demo/ARMCM33\_STM32L5\_Nucleo\_L552ZE\_CubeIDE/Prog/App/boot.c, [671](#)
- Demo/ARMCM33\_STM32L5\_Nucleo\_L552ZE\_CubeIDE/Prog/App/boot.h, [963](#)
- Demo/ARMCM33\_STM32L5\_Nucleo\_L552ZE\_GCC/Prog/boot.c, [676](#)
- Demo/ARMCM33\_STM32L5\_Nucleo\_L552ZE\_GCC/Prog/boot.h, [965](#)
- Demo/ARMCM33\_STM32L5\_Nucleo\_L552ZE\_IAR/Prog/boot.c, [681](#)
- Demo/ARMCM33\_STM32L5\_Nucleo\_L552ZE\_IAR/Prog/boot.h, [966](#)
- Demo/ARMCM33\_STM32L5\_Nucleo\_L552ZE\_Keil/Prog/boot.c, [686](#)
- Demo/ARMCM33\_STM32L5\_Nucleo\_L552ZE\_Keil/Prog/boot.h, [968](#)
- Demo/ARMCM3\_EFM32\_Olimex\_EM32G880\_F128STK\_GCC/Prog/boot.c, [691](#)
- Demo/ARMCM3\_EFM32\_Olimex\_EM32G880\_F128STK\_GCC/Prog/boot.h, [969](#)
- Demo/ARMCM3\_EFM32\_Olimex\_EM32G880\_F128STK\_IAR/Prog/boot.c, [694](#)
- Demo/ARMCM3\_EFM32\_Olimex\_EM32G880\_F128STK\_IAR/Prog/boot.h, [971](#)
- Demo/ARMCM3\_LM3S\_EK\_LM3S6965\_GCC/Prog/boot.c, [696](#)
- Demo/ARMCM3\_LM3S\_EK\_LM3S6965\_GCC/Prog/boot.h, [972](#)
- Demo/ARMCM3\_LM3S\_EK\_LM3S6965\_IAR/Prog/boot.c, [699](#)
- Demo/ARMCM3\_LM3S\_EK\_LM3S6965\_IAR/Prog/boot.h, [974](#)
- Demo/ARMCM3\_LM3S\_EK\_LM3S8962\_GCC/Prog/boot.c, [702](#)
- Demo/ARMCM3\_LM3S\_EK\_LM3S8962\_GCC/Prog/boot.h, [975](#)
- Demo/ARMCM3\_LM3S\_EK\_LM3S8962\_IAR/Prog/boot.c, [706](#)
- Demo/ARMCM3\_LM3S\_EK\_LM3S8962\_IAR/Prog/boot.h, [977](#)
- Demo/ARMCM3\_STM32F1\_Nucleo\_F103RB\_CubeIDE/Prog/App/boot.c, [710](#)
- Demo/ARMCM3\_STM32F1\_Nucleo\_F103RB\_CubeIDE/Prog/App/boot.h, [979](#)
- Demo/ARMCM3\_STM32F1\_Nucleo\_F103RB\_GCC/Prog/boot.c, [713](#)
- Demo/ARMCM3\_STM32F1\_Nucleo\_F103RB\_GCC/Prog/boot.h, [980](#)
- Demo/ARMCM3\_STM32F1\_Nucleo\_F103RB\_IAR/Prog/boot.c, [716](#)
- Demo/ARMCM3\_STM32F1\_Nucleo\_F103RB\_IAR/Prog/boot.h, [982](#)
- Demo/ARMCM3\_STM32F1\_Nucleo\_F103RB\_Keil/Prog/boot.c, [719](#)
- Demo/ARMCM3\_STM32F1\_Nucleo\_F103RB\_Keil/Prog/boot.h, [984](#)
- Demo/ARMCM3\_STM32F1\_Olimex\_STM32\_P103\_CubeIDE/Prog/App/boot.c, [722](#)
- Demo/ARMCM3\_STM32F1\_Olimex\_STM32\_P103\_CubeIDE/Prog/App/boot.h, [985](#)
- Demo/ARMCM3\_STM32F1\_Olimex\_STM32\_P103\_GCC/Prog/boot.c, [727](#)
- Demo/ARMCM3\_STM32F1\_Olimex\_STM32\_P103\_GCC/Prog/boot.h, [987](#)
- Demo/ARMCM3\_STM32F1\_Olimex\_STM32\_P103\_IAR/Prog/boot.c, [732](#)
- Demo/ARMCM3\_STM32F1\_Olimex\_STM32\_P103\_IAR/Prog/boot.h, [989](#)
- Demo/ARMCM3\_STM32F1\_Olimex\_STM32\_P103\_Keil/Prog/boot.c, [737](#)
- Demo/ARMCM3\_STM32F1\_Olimex\_STM32\_P103\_Keil/Prog/boot.h, [990](#)
- Demo/ARMCM3\_STM32F1\_Olimexino\_STM32\_CubeIDE/Prog/App/boot.c, [742](#)
- Demo/ARMCM3\_STM32F1\_Olimexino\_STM32\_CubeIDE/Prog/App/boot.h, [992](#)
- Demo/ARMCM3\_STM32F1\_Olimexino\_STM32\_GCC/Prog/boot.c, [745](#)
- Demo/ARMCM3\_STM32F1\_Olimexino\_STM32\_GCC/Prog/boot.h, [994](#)
- Demo/ARMCM3\_STM32F1\_Olimexino\_STM32\_IAR/Prog/boot.c, [749](#)
- Demo/ARMCM3\_STM32F1\_Olimexino\_STM32\_IAR/Prog/boot.h, [995](#)
- Demo/ARMCM3\_STM32F1\_Olimexino\_STM32\_Keil/Prog/boot.c, [753](#)
- Demo/ARMCM3\_STM32F1\_Olimexino\_STM32\_Keil/Prog/boot.h, [997](#)
- Demo/ARMCM3\_STM32F2\_Olimex\_STM32\_P207\_CubeIDE/Prog/App/boot.c, [757](#)
- Demo/ARMCM3\_STM32F2\_Olimex\_STM32\_P207\_CubeIDE/Prog/App/boot.h, [999](#)
- Demo/ARMCM3\_STM32F2\_Olimex\_STM32\_P207\_GCC/Prog/boot.c, [763](#)
- Demo/ARMCM3\_STM32F2\_Olimex\_STM32\_P207\_GCC/Prog/boot.h, [1000](#)
- Demo/ARMCM3\_STM32F2\_Olimex\_STM32\_P207\_IAR/Prog/boot.c, [768](#)
- Demo/ARMCM3\_STM32F2\_Olimex\_STM32\_P207\_IAR/Prog/boot.h, [1002](#)
- Demo/ARMCM3\_STM32F2\_Olimex\_STM32\_P207\_Keil/Prog/boot.c, [773](#)
- Demo/ARMCM3\_STM32F2\_Olimex\_STM32\_P207\_Keil/Prog/boot.h, [1004](#)
- Demo/ARMCM4\_S32K14\_S32K144EVB\_GCC/Prog/boot.c, [778](#)
- Demo/ARMCM4\_S32K14\_S32K144EVB\_GCC/Prog/boot.h, [1005](#)
- Demo/ARMCM4\_S32K14\_S32K144EVB\_IAR/Prog/boot.c, [784](#)
- Demo/ARMCM4\_S32K14\_S32K144EVB\_IAR/Prog/boot.h, [1007](#)
- Demo/ARMCM4\_STM32F3\_Nucleo\_F303K8

- CubeIDE/Prog/App/boot.c, [791](#)
- Demo/ARMCM4\_STM32F3\_Nucleo\_F303K8\_↔  
CubeIDE/Prog/App/boot.h, [1009](#)
- Demo/ARMCM4\_STM32F3\_Nucleo\_F303K8\_G↔  
CC/Prog/boot.c, [796](#)
- Demo/ARMCM4\_STM32F3\_Nucleo\_F303K8\_G↔  
CC/Prog/boot.h, [1010](#)
- Demo/ARMCM4\_STM32F3\_Nucleo\_F303K8\_IA↔  
R/Prog/boot.c, [801](#)
- Demo/ARMCM4\_STM32F3\_Nucleo\_F303K8\_IA↔  
R/Prog/boot.h, [1012](#)
- Demo/ARMCM4\_STM32F3\_Nucleo\_F303K8\_↔  
Keil/Prog/boot.c, [806](#)
- Demo/ARMCM4\_STM32F3\_Nucleo\_F303K8\_↔  
Keil/Prog/boot.h, [1014](#)
- Demo/ARMCM4\_STM32F4\_Nucleo\_F429ZI\_↔  
CubeIDE/Prog/App/boot.c, [811](#)
- Demo/ARMCM4\_STM32F4\_Nucleo\_F429ZI\_↔  
CubeIDE/Prog/App/boot.h, [1015](#)
- Demo/ARMCM4\_STM32F4\_Nucleo\_F429ZI\_G↔  
CC/Prog/boot.c, [816](#)
- Demo/ARMCM4\_STM32F4\_Nucleo\_F429ZI\_G↔  
CC/Prog/boot.h, [1017](#)
- Demo/ARMCM4\_STM32F4\_Nucleo\_F429ZI\_IA↔  
R/Prog/boot.c, [821](#)
- Demo/ARMCM4\_STM32F4\_Nucleo\_F429ZI\_IA↔  
R/Prog/boot.h, [1019](#)
- Demo/ARMCM4\_STM32F4\_Nucleo\_F429ZI\_↔  
Keil/Prog/boot.c, [826](#)
- Demo/ARMCM4\_STM32F4\_Nucleo\_F429ZI\_↔  
Keil/Prog/boot.h, [1020](#)
- Demo/ARMCM4\_STM32F4\_Olimex\_STM32↔  
P405\_CubeIDE/Prog/App/boot.c, [831](#)
- Demo/ARMCM4\_STM32F4\_Olimex\_STM32↔  
P405\_CubeIDE/Prog/App/boot.h, [1022](#)
- Demo/ARMCM4\_STM32F4\_Olimex\_STM32↔  
P405\_GCC/Prog/boot.c, [836](#)
- Demo/ARMCM4\_STM32F4\_Olimex\_STM32↔  
P405\_GCC/Prog/boot.h, [1024](#)
- Demo/ARMCM4\_STM32F4\_Olimex\_STM32↔  
P405\_IAR/Prog/boot.c, [841](#)
- Demo/ARMCM4\_STM32F4\_Olimex\_STM32↔  
P405\_IAR/Prog/boot.h, [1025](#)
- Demo/ARMCM4\_STM32F4\_Olimex\_STM32↔  
P405\_Keil/Prog/boot.c, [846](#)
- Demo/ARMCM4\_STM32F4\_Olimex\_STM32↔  
P405\_Keil/Prog/boot.h, [1027](#)
- Demo/ARMCM4\_STM32L4\_Nucleo\_L476RG\_↔  
CubeIDE/Prog/App/boot.c, [851](#)
- Demo/ARMCM4\_STM32L4\_Nucleo\_L476RG\_↔  
CubeIDE/Prog/App/boot.h, [1029](#)
- Demo/ARMCM4\_STM32L4\_Nucleo\_L476RG\_G↔  
CC/Prog/boot.c, [856](#)
- Demo/ARMCM4\_STM32L4\_Nucleo\_L476RG\_G↔  
CC/Prog/boot.h, [1030](#)
- Demo/ARMCM4\_STM32L4\_Nucleo\_L476RG\_I↔  
AR/Prog/boot.c, [861](#)
- Demo/ARMCM4\_STM32L4\_Nucleo\_L476RG\_I↔  
AR/Prog/boot.h, [1032](#)
- Demo/ARMCM4\_STM32L4\_Nucleo\_L476RG\_↔  
Keil/Prog/boot.c, [866](#)
- Demo/ARMCM4\_STM32L4\_Nucleo\_L476RG\_↔  
Keil/Prog/boot.h, [1034](#)
- Demo/ARMCM4\_TM4C\_DK\_TM4C123G\_IAR/↔  
Prog/boot.c, [870](#)
- Demo/ARMCM4\_TM4C\_DK\_TM4C123G\_IAR/↔  
Prog/boot.h, [1035](#)
- Demo/ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_↔  
GCC/Prog/boot.c, [873](#)
- Demo/ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_↔  
GCC/Prog/boot.h, [1037](#)
- Demo/ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_I↔  
AR/Prog/boot.c, [877](#)
- Demo/ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_I↔  
AR/Prog/boot.h, [1039](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F746ZG\_↔  
CubeIDE/Prog/App/boot.c, [880](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F746ZG\_↔  
CubeIDE/Prog/App/boot.h, [1040](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F746ZG\_G↔  
CC/Prog/boot.c, [885](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F746ZG\_G↔  
CC/Prog/boot.h, [1042](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F746ZG\_I↔  
AR/Prog/boot.c, [890](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F746ZG\_I↔  
AR/Prog/boot.h, [1044](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F746ZG\_↔  
Keil/Prog/boot.c, [895](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F746ZG\_↔  
Keil/Prog/boot.h, [1045](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F767ZI\_↔  
CubeIDE/Prog/App/boot.c, [900](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F767ZI\_↔  
CubeIDE/Prog/App/boot.h, [1047](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F767ZI\_G↔  
CC/Prog/boot.c, [903](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F767ZI\_G↔  
CC/Prog/boot.h, [1049](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F767ZI\_IA↔  
R/Prog/boot.c, [906](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F767ZI\_IA↔  
R/Prog/boot.h, [1050](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F767ZI\_↔  
Keil/Prog/boot.c, [909](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F767ZI\_↔  
Keil/Prog/boot.h, [1052](#)
- Demo/ARMCM7\_STM32H7\_Nucleo\_H743ZI\_↔  
CubeIDE/Prog/App/boot.c, [912](#)
- Demo/ARMCM7\_STM32H7\_Nucleo\_H743ZI\_↔  
CubeIDE/Prog/App/boot.h, [1054](#)
- Demo/ARMCM7\_STM32H7\_Nucleo\_H743ZI\_G↔  
CC/Prog/boot.c, [917](#)
- Demo/ARMCM7\_STM32H7\_Nucleo\_H743ZI\_G↔  
CC/Prog/boot.h, [1055](#)
- Demo/ARMCM7\_STM32H7\_Nucleo\_H743ZI\_IA↔

- R/Prog/boot.c, [922](#)
- Demo/ARMCM7\_STM32H7\_Nucleo\_H743ZI\_IA↔  
R/Prog/boot.h, [1057](#)
- Demo/ARMCM7\_STM32H7\_Nucleo\_H743ZI↔  
Keil/Prog/boot.c, [927](#)
- Demo/ARMCM7\_STM32H7\_Nucleo\_H743ZI↔  
Keil/Prog/boot.h, [1059](#)
- Demo/HCS12\_Evbplus\_Dragon12p\_Code↔  
Warrior/Prog/boot.c, [932](#)
- Demo/HCS12\_Evbplus\_Dragon12p\_Code↔  
Warrior/Prog/boot.h, [1060](#)
- bootBlockInfo
  - \_template/flash.c, [1275](#)
  - ARMCM0\_S32K11/flash.c, [1284](#)
  - ARMCM0\_STM32F0/flash.c, [1292](#)
  - ARMCM0\_STM32G0/flash.c, [1301](#)
  - ARMCM0\_XMC1/flash.c, [1310](#)
  - ARMCM33\_STM32L5/flash.c, [1319](#)
  - ARMCM3\_EFM32/flash.c, [1328](#)
  - ARMCM3\_LM3S/flash.c, [1337](#)
  - ARMCM3\_STM32F1/flash.c, [1344](#)
  - ARMCM3\_STM32F2/flash.c, [1352](#)
  - ARMCM4\_S32K14/flash.c, [1361](#)
  - ARMCM4\_STM32F3/flash.c, [1368](#)
  - ARMCM4\_STM32F4/flash.c, [1376](#)
  - ARMCM4\_STM32L4/flash.c, [1385](#)
  - ARMCM4\_TM4C/flash.c, [1394](#)
  - ARMCM4\_XMC4/flash.c, [1404](#)
  - ARMCM7\_STM32F7/flash.c, [1412](#)
  - ARMCM7\_STM32H7/flash.c, [1421](#)
  - flash\_ecc.c, [1512](#)
  - HCS12/flash.c, [1431](#)
- BootComCanCheckActivationRequest
  - Demo/ARMCM0\_STM32F0\_Nucleo\_F091RC↔  
CubeIDE/Prog/App/boot.c, [632](#)
  - Demo/ARMCM0\_STM32F0\_Nucleo\_F091RC\_G↔  
CC/Prog/boot.c, [637](#)
  - Demo/ARMCM0\_STM32F0\_Nucleo\_F091RC\_I↔  
AR/Prog/boot.c, [642](#)
  - Demo/ARMCM0\_STM32F0\_Nucleo\_F091RC↔  
Keil/Prog/boot.c, [647](#)
  - Demo/ARMCM0\_XMC1\_XMC1400\_Boot\_Kit\_G↔  
CC/Prog/boot.c, [664](#)
  - Demo/ARMCM0\_XMC1\_XMC1400\_Boot\_Kit\_IA↔  
R/Prog/boot.c, [667](#)
  - Demo/ARMCM33\_STM32L5\_Nucleo\_L552ZE↔  
CubeIDE/Prog/App/boot.c, [671](#)
  - Demo/ARMCM33\_STM32L5\_Nucleo\_L552ZE↔  
GCC/Prog/boot.c, [676](#)
  - Demo/ARMCM33\_STM32L5\_Nucleo\_L552ZE\_I↔  
AR/Prog/boot.c, [681](#)
  - Demo/ARMCM33\_STM32L5\_Nucleo\_L552ZE↔  
Keil/Prog/boot.c, [686](#)
  - Demo/ARMCM3\_LM3S\_EK\_LM3S8962\_GCC↔  
Prog/boot.c, [702](#)
  - Demo/ARMCM3\_LM3S\_EK\_LM3S8962\_IAR↔  
Prog/boot.c, [706](#)
  - Demo/ARMCM3\_STM32F1\_Olimex\_STM32↔  
P103\_CubeIDE/Prog/App/boot.c, [722](#)
  - Demo/ARMCM3\_STM32F1\_Olimex\_STM32↔  
P103\_GCC/Prog/boot.c, [727](#)
  - Demo/ARMCM3\_STM32F1\_Olimex\_STM32↔  
P103\_IAR/Prog/boot.c, [732](#)
  - Demo/ARMCM3\_STM32F1\_Olimex\_STM32↔  
P103\_Keil/Prog/boot.c, [737](#)
  - Demo/ARMCM3\_STM32F1\_Olimexino\_STM32↔  
\_CubeIDE/Prog/App/boot.c, [742](#)
  - Demo/ARMCM3\_STM32F1\_Olimexino\_STM32↔  
\_GCC/Prog/boot.c, [745](#)
  - Demo/ARMCM3\_STM32F1\_Olimexino\_STM32↔  
\_IAR/Prog/boot.c, [749](#)
  - Demo/ARMCM3\_STM32F1\_Olimexino\_STM32↔  
\_Keil/Prog/boot.c, [753](#)
  - Demo/ARMCM3\_STM32F2\_Olimex\_STM32↔  
P207\_CubeIDE/Prog/App/boot.c, [758](#)
  - Demo/ARMCM3\_STM32F2\_Olimex\_STM32↔  
P207\_GCC/Prog/boot.c, [763](#)
  - Demo/ARMCM3\_STM32F2\_Olimex\_STM32↔  
P207\_IAR/Prog/boot.c, [768](#)
  - Demo/ARMCM3\_STM32F2\_Olimex\_STM32↔  
P207\_Keil/Prog/boot.c, [773](#)
  - Demo/ARMCM4\_S32K14\_S32K144EVB\_GCC↔  
Prog/boot.c, [778](#)
  - Demo/ARMCM4\_S32K14\_S32K144EVB\_IAR↔  
Prog/boot.c, [785](#)
  - Demo/ARMCM4\_STM32F3\_Nucleo\_F303K8↔  
CubeIDE/Prog/App/boot.c, [791](#)
  - Demo/ARMCM4\_STM32F3\_Nucleo\_F303K8\_G↔  
CC/Prog/boot.c, [796](#)
  - Demo/ARMCM4\_STM32F3\_Nucleo\_F303K8\_IA↔  
R/Prog/boot.c, [801](#)
  - Demo/ARMCM4\_STM32F3\_Nucleo\_F303K8↔  
Keil/Prog/boot.c, [806](#)
  - Demo/ARMCM4\_STM32F4\_Nucleo\_F429ZI↔  
CubeIDE/Prog/App/boot.c, [811](#)
  - Demo/ARMCM4\_STM32F4\_Nucleo\_F429ZI\_G↔  
CC/Prog/boot.c, [816](#)
  - Demo/ARMCM4\_STM32F4\_Nucleo\_F429ZI\_IA↔  
R/Prog/boot.c, [821](#)
  - Demo/ARMCM4\_STM32F4\_Nucleo\_F429ZI↔  
Keil/Prog/boot.c, [826](#)
  - Demo/ARMCM4\_STM32F4\_Olimex\_STM32↔  
P405\_CubeIDE/Prog/App/boot.c, [831](#)
  - Demo/ARMCM4\_STM32F4\_Olimex\_STM32↔  
P405\_GCC/Prog/boot.c, [836](#)
  - Demo/ARMCM4\_STM32F4\_Olimex\_STM32↔  
P405\_IAR/Prog/boot.c, [841](#)
  - Demo/ARMCM4\_STM32F4\_Olimex\_STM32↔  
P405\_Keil/Prog/boot.c, [846](#)
  - Demo/ARMCM4\_STM32L4\_Nucleo\_L476RG↔  
CubeIDE/Prog/App/boot.c, [851](#)
  - Demo/ARMCM4\_STM32L4\_Nucleo\_L476RG\_G↔  
CC/Prog/boot.c, [856](#)
  - Demo/ARMCM4\_STM32L4\_Nucleo\_L476RG\_I↔  
AR/Prog/boot.c, [861](#)
  - Demo/ARMCM4\_STM32L4\_Nucleo\_L476RG↔

- Keil/Prog/boot.c, [866](#)
- Demo/ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_↔  
GCC/Prog/boot.c, [873](#)
- Demo/ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_I↔  
AR/Prog/boot.c, [877](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F746ZG\_↔  
CubeIDE/Prog/App/boot.c, [880](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F746ZG\_G↔  
CC/Prog/boot.c, [885](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F746ZG\_I↔  
AR/Prog/boot.c, [890](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F746ZG\_↔  
Keil/Prog/boot.c, [895](#)
- Demo/ARMCM7\_STM32H7\_Nucleo\_H743ZI\_↔  
CubeIDE/Prog/App/boot.c, [912](#)
- Demo/ARMCM7\_STM32H7\_Nucleo\_H743ZI\_G↔  
CC/Prog/boot.c, [917](#)
- Demo/ARMCM7\_STM32H7\_Nucleo\_H743ZI\_IA↔  
R/Prog/boot.c, [922](#)
- Demo/ARMCM7\_STM32H7\_Nucleo\_H743ZI\_↔  
Keil/Prog/boot.c, [927](#)
- Demo/HCS12\_Evbplus\_Dragon12p\_Code↔  
Warrior/Prog/boot.c, [932](#)
- BootComCanInit
  - Demo/ARMCM0\_STM32F0\_Nucleo\_F091RC\_↔  
CubeIDE/Prog/App/boot.c, [632](#)
  - Demo/ARMCM0\_STM32F0\_Nucleo\_F091RC\_G↔  
CC/Prog/boot.c, [637](#)
  - Demo/ARMCM0\_STM32F0\_Nucleo\_F091RC\_I↔  
AR/Prog/boot.c, [642](#)
  - Demo/ARMCM0\_STM32F0\_Nucleo\_F091RC\_↔  
Keil/Prog/boot.c, [647](#)
  - Demo/ARMCM0\_XMC1\_XMC1400\_Boot\_Kit\_G↔  
CC/Prog/boot.c, [664](#)
  - Demo/ARMCM0\_XMC1\_XMC1400\_Boot\_Kit\_IA↔  
R/Prog/boot.c, [668](#)
  - Demo/ARMCM33\_STM32L5\_Nucleo\_L552ZE\_↔  
CubeIDE/Prog/App/boot.c, [672](#)
  - Demo/ARMCM33\_STM32L5\_Nucleo\_L552ZE\_↔  
GCC/Prog/boot.c, [677](#)
  - Demo/ARMCM33\_STM32L5\_Nucleo\_L552ZE\_I↔  
AR/Prog/boot.c, [682](#)
  - Demo/ARMCM33\_STM32L5\_Nucleo\_L552ZE\_↔  
Keil/Prog/boot.c, [687](#)
  - Demo/ARMCM3\_LM3S\_EK\_LM3S8962\_GCC/↔  
Prog/boot.c, [702](#)
  - Demo/ARMCM3\_LM3S\_EK\_LM3S8962\_IAR/↔  
Prog/boot.c, [706](#)
  - Demo/ARMCM3\_STM32F1\_Olimex\_STM32↔  
P103\_CubeIDE/Prog/App/boot.c, [722](#)
  - Demo/ARMCM3\_STM32F1\_Olimex\_STM32↔  
P103\_GCC/Prog/boot.c, [727](#)
  - Demo/ARMCM3\_STM32F1\_Olimex\_STM32↔  
P103\_IAR/Prog/boot.c, [732](#)
  - Demo/ARMCM3\_STM32F1\_Olimex\_STM32↔  
P103\_Keil/Prog/boot.c, [737](#)
  - Demo/ARMCM3\_STM32F1\_Olimexino\_STM32↔  
\_CubeIDE/Prog/App/boot.c, [742](#)
  - Demo/ARMCM3\_STM32F1\_Olimexino\_STM32↔  
\_GCC/Prog/boot.c, [746](#)
  - Demo/ARMCM3\_STM32F1\_Olimexino\_STM32↔  
\_IAR/Prog/boot.c, [750](#)
  - Demo/ARMCM3\_STM32F1\_Olimexino\_STM32↔  
\_Keil/Prog/boot.c, [754](#)
  - Demo/ARMCM3\_STM32F2\_Olimex\_STM32↔  
P207\_CubeIDE/Prog/App/boot.c, [758](#)
  - Demo/ARMCM3\_STM32F2\_Olimex\_STM32↔  
P207\_GCC/Prog/boot.c, [763](#)
  - Demo/ARMCM3\_STM32F2\_Olimex\_STM32↔  
P207\_IAR/Prog/boot.c, [768](#)
  - Demo/ARMCM3\_STM32F2\_Olimex\_STM32↔  
P207\_Keil/Prog/boot.c, [773](#)
  - Demo/ARMCM4\_S32K14\_S32K144EVB\_GCC/↔  
Prog/boot.c, [778](#)
  - Demo/ARMCM4\_S32K14\_S32K144EVB\_IAR/↔  
Prog/boot.c, [785](#)
  - Demo/ARMCM4\_STM32F3\_Nucleo\_F303K8\_↔  
CubeIDE/Prog/App/boot.c, [791](#)
  - Demo/ARMCM4\_STM32F3\_Nucleo\_F303K8\_G↔  
CC/Prog/boot.c, [796](#)
  - Demo/ARMCM4\_STM32F3\_Nucleo\_F303K8\_IA↔  
R/Prog/boot.c, [801](#)
  - Demo/ARMCM4\_STM32F3\_Nucleo\_F303K8\_↔  
Keil/Prog/boot.c, [806](#)
  - Demo/ARMCM4\_STM32F4\_Nucleo\_F429ZI\_↔  
CubeIDE/Prog/App/boot.c, [811](#)
  - Demo/ARMCM4\_STM32F4\_Nucleo\_F429ZI\_G↔  
CC/Prog/boot.c, [816](#)
  - Demo/ARMCM4\_STM32F4\_Nucleo\_F429ZI\_IA↔  
R/Prog/boot.c, [821](#)
  - Demo/ARMCM4\_STM32F4\_Nucleo\_F429ZI\_↔  
Keil/Prog/boot.c, [826](#)
  - Demo/ARMCM4\_STM32F4\_Olimex\_STM32↔  
P405\_CubeIDE/Prog/App/boot.c, [831](#)
  - Demo/ARMCM4\_STM32F4\_Olimex\_STM32↔  
P405\_GCC/Prog/boot.c, [836](#)
  - Demo/ARMCM4\_STM32F4\_Olimex\_STM32↔  
P405\_IAR/Prog/boot.c, [841](#)
  - Demo/ARMCM4\_STM32F4\_Olimex\_STM32↔  
P405\_Keil/Prog/boot.c, [846](#)
  - Demo/ARMCM4\_STM32L4\_Nucleo\_L476RG\_↔  
CubeIDE/Prog/App/boot.c, [851](#)
  - Demo/ARMCM4\_STM32L4\_Nucleo\_L476RG\_G↔  
CC/Prog/boot.c, [856](#)
  - Demo/ARMCM4\_STM32L4\_Nucleo\_L476RG\_I↔  
AR/Prog/boot.c, [861](#)
  - Demo/ARMCM4\_STM32L4\_Nucleo\_L476RG\_↔  
Keil/Prog/boot.c, [866](#)
  - Demo/ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_↔  
GCC/Prog/boot.c, [873](#)
  - Demo/ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_I↔  
AR/Prog/boot.c, [877](#)
  - Demo/ARMCM7\_STM32F7\_Nucleo\_F746ZG\_↔  
CubeIDE/Prog/App/boot.c, [881](#)
  - Demo/ARMCM7\_STM32F7\_Nucleo\_F746ZG\_G↔  
CC/Prog/boot.c, [886](#)



- Demo/ARMCM7\_STM32F7\_Nucleo\_F746ZG\_I↔  
AR/Prog/boot.c, [891](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F746ZG↔  
Keil/Prog/boot.c, [896](#)
- Demo/ARMCM7\_STM32H7\_Nucleo\_H743ZI↔  
CubeIDE/Prog/App/boot.c, [913](#)
- Demo/ARMCM7\_STM32H7\_Nucleo\_H743ZI\_G↔  
CC/Prog/boot.c, [918](#)
- Demo/ARMCM7\_STM32H7\_Nucleo\_H743ZI\_IA↔  
R/Prog/boot.c, [923](#)
- Demo/ARMCM7\_STM32H7\_Nucleo\_H743ZI↔  
Keil/Prog/boot.c, [928](#)
- Demo/HCS12\_Evbplus\_Dragon12p\_Code↔  
Warrior/Prog/boot.c, [933](#)
- BootComCheckActivationRequest
  - Demo/\_template/Prog/boot.c, [617](#)
  - Demo/\_template/Prog/boot.h, [938](#)
  - Demo/ARMCM0\_STM32F0\_Discovery\_STM32↔  
F051\_CubeIDE/Prog/App/boot.c, [620](#)
  - Demo/ARMCM0\_STM32F0\_Discovery\_STM32↔  
F051\_CubeIDE/Prog/App/boot.h, [940](#)
  - Demo/ARMCM0\_STM32F0\_Discovery\_STM32↔  
F051\_GCC/Prog/boot.c, [623](#)
  - Demo/ARMCM0\_STM32F0\_Discovery\_STM32↔  
F051\_GCC/Prog/boot.h, [941](#)
  - Demo/ARMCM0\_STM32F0\_Discovery\_STM32↔  
F051\_IAR/Prog/boot.c, [626](#)
  - Demo/ARMCM0\_STM32F0\_Discovery\_STM32↔  
F051\_IAR/Prog/boot.h, [943](#)
  - Demo/ARMCM0\_STM32F0\_Discovery\_STM32↔  
F051\_Keil/Prog/boot.c, [629](#)
  - Demo/ARMCM0\_STM32F0\_Discovery\_STM32↔  
F051\_Keil/Prog/boot.h, [945](#)
  - Demo/ARMCM0\_STM32F0\_Nucleo\_F091RC↔  
CubeIDE/Prog/App/boot.c, [633](#)
  - Demo/ARMCM0\_STM32F0\_Nucleo\_F091RC↔  
CubeIDE/Prog/App/boot.h, [946](#)
  - Demo/ARMCM0\_STM32F0\_Nucleo\_F091RC\_G↔  
CC/Prog/boot.c, [638](#)
  - Demo/ARMCM0\_STM32F0\_Nucleo\_F091RC\_G↔  
CC/Prog/boot.h, [948](#)
  - Demo/ARMCM0\_STM32F0\_Nucleo\_F091RC\_I↔  
AR/Prog/boot.c, [643](#)
  - Demo/ARMCM0\_STM32F0\_Nucleo\_F091RC\_I↔  
AR/Prog/boot.h, [950](#)
  - Demo/ARMCM0\_STM32F0\_Nucleo\_F091RC↔  
Keil/Prog/boot.c, [648](#)
  - Demo/ARMCM0\_STM32F0\_Nucleo\_F091RC↔  
Keil/Prog/boot.h, [951](#)
  - Demo/ARMCM0\_STM32G0\_Nucleo\_G071RB↔  
CubeIDE/Prog/App/boot.c, [652](#)
  - Demo/ARMCM0\_STM32G0\_Nucleo\_G071RB↔  
CubeIDE/Prog/App/boot.h, [953](#)
  - Demo/ARMCM0\_STM32G0\_Nucleo\_G071RB↔  
GCC/Prog/boot.c, [655](#)
  - Demo/ARMCM0\_STM32G0\_Nucleo\_G071RB↔  
GCC/Prog/boot.h, [955](#)
  - Demo/ARMCM0\_STM32G0\_Nucleo\_G071RB\_I↔  
AR/Prog/boot.c, [658](#)
  - Demo/ARMCM0\_STM32G0\_Nucleo\_G071RB\_I↔  
AR/Prog/boot.h, [956](#)
  - Demo/ARMCM0\_STM32G0\_Nucleo\_G071RB↔  
Keil/Prog/boot.c, [661](#)
  - Demo/ARMCM0\_STM32G0\_Nucleo\_G071RB↔  
Keil/Prog/boot.h, [958](#)
  - Demo/ARMCM0\_XMC1\_XMC1400\_Boot\_Kit\_G↔  
CC/Prog/boot.c, [665](#)
  - Demo/ARMCM0\_XMC1\_XMC1400\_Boot\_Kit\_G↔  
CC/Prog/boot.h, [960](#)
  - Demo/ARMCM0\_XMC1\_XMC1400\_Boot\_Kit\_IA↔  
R/Prog/boot.c, [668](#)
  - Demo/ARMCM0\_XMC1\_XMC1400\_Boot\_Kit\_IA↔  
R/Prog/boot.h, [961](#)
  - Demo/ARMCM33\_STM32L5\_Nucleo\_L552ZE↔  
CubeIDE/Prog/App/boot.c, [672](#)
  - Demo/ARMCM33\_STM32L5\_Nucleo\_L552ZE↔  
CubeIDE/Prog/App/boot.h, [963](#)
  - Demo/ARMCM33\_STM32L5\_Nucleo\_L552ZE↔  
GCC/Prog/boot.c, [677](#)
  - Demo/ARMCM33\_STM32L5\_Nucleo\_L552ZE↔  
GCC/Prog/boot.h, [965](#)
  - Demo/ARMCM33\_STM32L5\_Nucleo\_L552ZE\_I↔  
AR/Prog/boot.c, [682](#)
  - Demo/ARMCM33\_STM32L5\_Nucleo\_L552ZE\_I↔  
AR/Prog/boot.h, [966](#)
  - Demo/ARMCM33\_STM32L5\_Nucleo\_L552ZE↔  
Keil/Prog/boot.c, [687](#)
  - Demo/ARMCM33\_STM32L5\_Nucleo\_L552ZE↔  
Keil/Prog/boot.h, [968](#)
  - Demo/ARMCM3\_EFM32\_Olimex\_EM32G880↔  
F128STK\_GCC/Prog/boot.c, [691](#)
  - Demo/ARMCM3\_EFM32\_Olimex\_EM32G880↔  
F128STK\_GCC/Prog/boot.h, [970](#)
  - Demo/ARMCM3\_EFM32\_Olimex\_EM32G880↔  
F128STK\_IAR/Prog/boot.c, [694](#)
  - Demo/ARMCM3\_EFM32\_Olimex\_EM32G880↔  
F128STK\_IAR/Prog/boot.h, [971](#)
  - Demo/ARMCM3\_LM3S\_EK\_LM3S6965\_GCC↔  
Prog/boot.c, [697](#)
  - Demo/ARMCM3\_LM3S\_EK\_LM3S6965\_GCC↔  
Prog/boot.h, [973](#)
  - Demo/ARMCM3\_LM3S\_EK\_LM3S6965\_IAR↔  
Prog/boot.c, [699](#)
  - Demo/ARMCM3\_LM3S\_EK\_LM3S6965\_IAR↔  
Prog/boot.h, [974](#)
  - Demo/ARMCM3\_LM3S\_EK\_LM3S8962\_GCC↔  
Prog/boot.c, [703](#)
  - Demo/ARMCM3\_LM3S\_EK\_LM3S8962\_GCC↔  
Prog/boot.h, [976](#)
  - Demo/ARMCM3\_LM3S\_EK\_LM3S8962\_IAR↔  
Prog/boot.c, [706](#)
  - Demo/ARMCM3\_LM3S\_EK\_LM3S8962\_IAR↔  
Prog/boot.h, [977](#)
  - Demo/ARMCM3\_STM32F1\_Nucleo\_F103RB↔  
CubeIDE/Prog/App/boot.c, [710](#)
  - Demo/ARMCM3\_STM32F1\_Nucleo\_F103RB↔

- CubeIDE/Prog/App/boot.h, [979](#)
- Demo/ARMCM3\_STM32F1\_Nucleo\_F103RB\_G↔  
CC/Prog/boot.c, [713](#)
- Demo/ARMCM3\_STM32F1\_Nucleo\_F103RB\_G↔  
CC/Prog/boot.h, [980](#)
- Demo/ARMCM3\_STM32F1\_Nucleo\_F103RB\_I↔  
AR/Prog/boot.c, [716](#)
- Demo/ARMCM3\_STM32F1\_Nucleo\_F103RB\_I↔  
AR/Prog/boot.h, [982](#)
- Demo/ARMCM3\_STM32F1\_Nucleo\_F103RB\_↔  
Keil/Prog/boot.c, [719](#)
- Demo/ARMCM3\_STM32F1\_Nucleo\_F103RB\_↔  
Keil/Prog/boot.h, [984](#)
- Demo/ARMCM3\_STM32F1\_Olimex\_STM32↔  
P103\_CubeIDE/Prog/App/boot.c, [723](#)
- Demo/ARMCM3\_STM32F1\_Olimex\_STM32↔  
P103\_CubeIDE/Prog/App/boot.h, [985](#)
- Demo/ARMCM3\_STM32F1\_Olimex\_STM32↔  
P103\_GCC/Prog/boot.c, [728](#)
- Demo/ARMCM3\_STM32F1\_Olimex\_STM32↔  
P103\_GCC/Prog/boot.h, [987](#)
- Demo/ARMCM3\_STM32F1\_Olimex\_STM32↔  
P103\_IAR/Prog/boot.c, [733](#)
- Demo/ARMCM3\_STM32F1\_Olimex\_STM32↔  
P103\_IAR/Prog/boot.h, [989](#)
- Demo/ARMCM3\_STM32F1\_Olimex\_STM32↔  
P103\_Keil/Prog/boot.c, [738](#)
- Demo/ARMCM3\_STM32F1\_Olimex\_STM32↔  
P103\_Keil/Prog/boot.h, [990](#)
- Demo/ARMCM3\_STM32F1\_Olimexino\_STM32↔  
\_CubeIDE/Prog/App/boot.c, [742](#)
- Demo/ARMCM3\_STM32F1\_Olimexino\_STM32↔  
\_CubeIDE/Prog/App/boot.h, [992](#)
- Demo/ARMCM3\_STM32F1\_Olimexino\_STM32↔  
\_GCC/Prog/boot.c, [746](#)
- Demo/ARMCM3\_STM32F1\_Olimexino\_STM32↔  
\_GCC/Prog/boot.h, [994](#)
- Demo/ARMCM3\_STM32F1\_Olimexino\_STM32↔  
\_IAR/Prog/boot.c, [750](#)
- Demo/ARMCM3\_STM32F1\_Olimexino\_STM32↔  
\_IAR/Prog/boot.h, [995](#)
- Demo/ARMCM3\_STM32F1\_Olimexino\_STM32↔  
\_Keil/Prog/boot.c, [754](#)
- Demo/ARMCM3\_STM32F1\_Olimexino\_STM32↔  
\_Keil/Prog/boot.h, [997](#)
- Demo/ARMCM3\_STM32F2\_Olimex\_STM32↔  
P207\_CubeIDE/Prog/App/boot.c, [758](#)
- Demo/ARMCM3\_STM32F2\_Olimex\_STM32↔  
P207\_CubeIDE/Prog/App/boot.h, [999](#)
- Demo/ARMCM3\_STM32F2\_Olimex\_STM32↔  
P207\_GCC/Prog/boot.c, [763](#)
- Demo/ARMCM3\_STM32F2\_Olimex\_STM32↔  
P207\_GCC/Prog/boot.h, [1000](#)
- Demo/ARMCM3\_STM32F2\_Olimex\_STM32↔  
P207\_IAR/Prog/boot.c, [768](#)
- Demo/ARMCM3\_STM32F2\_Olimex\_STM32↔  
P207\_IAR/Prog/boot.h, [1002](#)
- Demo/ARMCM3\_STM32F2\_Olimex\_STM32↔
- P207\_Keil/Prog/boot.c, [773](#)
- Demo/ARMCM3\_STM32F2\_Olimex\_STM32↔  
P207\_Keil/Prog/boot.h, [1004](#)
- Demo/ARMCM4\_S32K14\_S32K144EVB\_GCC/↔  
Prog/boot.c, [779](#)
- Demo/ARMCM4\_S32K14\_S32K144EVB\_GCC/↔  
Prog/boot.h, [1005](#)
- Demo/ARMCM4\_S32K14\_S32K144EVB\_IAR/↔  
Prog/boot.c, [785](#)
- Demo/ARMCM4\_S32K14\_S32K144EVB\_IAR/↔  
Prog/boot.h, [1007](#)
- Demo/ARMCM4\_STM32F3\_Nucleo\_F303K8\_↔  
CubeIDE/Prog/App/boot.c, [792](#)
- Demo/ARMCM4\_STM32F3\_Nucleo\_F303K8\_↔  
CubeIDE/Prog/App/boot.h, [1009](#)
- Demo/ARMCM4\_STM32F3\_Nucleo\_F303K8\_G↔  
CC/Prog/boot.c, [797](#)
- Demo/ARMCM4\_STM32F3\_Nucleo\_F303K8\_G↔  
CC/Prog/boot.h, [1010](#)
- Demo/ARMCM4\_STM32F3\_Nucleo\_F303K8\_IA↔  
R/Prog/boot.c, [802](#)
- Demo/ARMCM4\_STM32F3\_Nucleo\_F303K8\_IA↔  
R/Prog/boot.h, [1012](#)
- Demo/ARMCM4\_STM32F3\_Nucleo\_F303K8\_↔  
Keil/Prog/boot.c, [807](#)
- Demo/ARMCM4\_STM32F3\_Nucleo\_F303K8\_↔  
Keil/Prog/boot.h, [1014](#)
- Demo/ARMCM4\_STM32F4\_Nucleo\_F429ZI\_↔  
CubeIDE/Prog/App/boot.c, [812](#)
- Demo/ARMCM4\_STM32F4\_Nucleo\_F429ZI\_↔  
CubeIDE/Prog/App/boot.h, [1015](#)
- Demo/ARMCM4\_STM32F4\_Nucleo\_F429ZI\_G↔  
CC/Prog/boot.c, [817](#)
- Demo/ARMCM4\_STM32F4\_Nucleo\_F429ZI\_G↔  
CC/Prog/boot.h, [1017](#)
- Demo/ARMCM4\_STM32F4\_Nucleo\_F429ZI\_IA↔  
R/Prog/boot.c, [822](#)
- Demo/ARMCM4\_STM32F4\_Nucleo\_F429ZI\_IA↔  
R/Prog/boot.h, [1019](#)
- Demo/ARMCM4\_STM32F4\_Nucleo\_F429ZI\_↔  
Keil/Prog/boot.c, [827](#)
- Demo/ARMCM4\_STM32F4\_Nucleo\_F429ZI\_↔  
Keil/Prog/boot.h, [1020](#)
- Demo/ARMCM4\_STM32F4\_Olimex\_STM32↔  
P405\_CubeIDE/Prog/App/boot.c, [832](#)
- Demo/ARMCM4\_STM32F4\_Olimex\_STM32↔  
P405\_CubeIDE/Prog/App/boot.h, [1022](#)
- Demo/ARMCM4\_STM32F4\_Olimex\_STM32↔  
P405\_GCC/Prog/boot.c, [837](#)
- Demo/ARMCM4\_STM32F4\_Olimex\_STM32↔  
P405\_GCC/Prog/boot.h, [1024](#)
- Demo/ARMCM4\_STM32F4\_Olimex\_STM32↔  
P405\_IAR/Prog/boot.c, [842](#)
- Demo/ARMCM4\_STM32F4\_Olimex\_STM32↔  
P405\_IAR/Prog/boot.h, [1025](#)
- Demo/ARMCM4\_STM32F4\_Olimex\_STM32↔  
P405\_Keil/Prog/boot.c, [847](#)
- Demo/ARMCM4\_STM32F4\_Olimex\_STM32↔

- P405\_Keil/Prog/boot.h, [1027](#)
- Demo/ARMCM4\_STM32L4\_Nucleo\_L476RG\_↔  
CubeIDE/Prog/App/boot.c, [852](#)
- Demo/ARMCM4\_STM32L4\_Nucleo\_L476RG\_↔  
CubeIDE/Prog/App/boot.h, [1029](#)
- Demo/ARMCM4\_STM32L4\_Nucleo\_L476RG\_G↔  
CC/Prog/boot.c, [857](#)
- Demo/ARMCM4\_STM32L4\_Nucleo\_L476RG\_G↔  
CC/Prog/boot.h, [1030](#)
- Demo/ARMCM4\_STM32L4\_Nucleo\_L476RG\_I↔  
AR/Prog/boot.c, [862](#)
- Demo/ARMCM4\_STM32L4\_Nucleo\_L476RG\_I↔  
AR/Prog/boot.h, [1032](#)
- Demo/ARMCM4\_STM32L4\_Nucleo\_L476RG\_↔  
Keil/Prog/boot.c, [867](#)
- Demo/ARMCM4\_STM32L4\_Nucleo\_L476RG\_↔  
Keil/Prog/boot.h, [1034](#)
- Demo/ARMCM4\_TM4C\_DK\_TM4C123G\_IAR↔  
Prog/boot.c, [870](#)
- Demo/ARMCM4\_TM4C\_DK\_TM4C123G\_IAR↔  
Prog/boot.h, [1035](#)
- Demo/ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_↔  
GCC/Prog/boot.c, [874](#)
- Demo/ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_↔  
GCC/Prog/boot.h, [1037](#)
- Demo/ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_I↔  
AR/Prog/boot.c, [877](#)
- Demo/ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_I↔  
AR/Prog/boot.h, [1039](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F746ZG\_↔  
CubeIDE/Prog/App/boot.c, [881](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F746ZG\_↔  
CubeIDE/Prog/App/boot.h, [1040](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F746ZG\_G↔  
CC/Prog/boot.c, [886](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F746ZG\_G↔  
CC/Prog/boot.h, [1042](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F746ZG\_I↔  
AR/Prog/boot.c, [891](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F746ZG\_I↔  
AR/Prog/boot.h, [1044](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F746ZG\_↔  
Keil/Prog/boot.c, [896](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F746ZG\_↔  
Keil/Prog/boot.h, [1045](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F767ZI\_↔  
CubeIDE/Prog/App/boot.c, [900](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F767ZI\_↔  
CubeIDE/Prog/App/boot.h, [1047](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F767ZI\_G↔  
CC/Prog/boot.c, [903](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F767ZI\_G↔  
CC/Prog/boot.h, [1049](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F767ZI\_IA↔  
R/Prog/boot.c, [906](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F767ZI\_IA↔  
R/Prog/boot.h, [1050](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F767ZI\_↔  
Keil/Prog/boot.c, [909](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F767ZI\_↔  
Keil/Prog/boot.h, [1052](#)
- Demo/ARMCM7\_STM32H7\_Nucleo\_H743ZI\_↔  
CubeIDE/Prog/App/boot.c, [913](#)
- Demo/ARMCM7\_STM32H7\_Nucleo\_H743ZI\_↔  
CubeIDE/Prog/App/boot.h, [1054](#)
- Demo/ARMCM7\_STM32H7\_Nucleo\_H743ZI\_G↔  
CC/Prog/boot.c, [918](#)
- Demo/ARMCM7\_STM32H7\_Nucleo\_H743ZI\_G↔  
CC/Prog/boot.h, [1055](#)
- Demo/ARMCM7\_STM32H7\_Nucleo\_H743ZI\_IA↔  
R/Prog/boot.c, [923](#)
- Demo/ARMCM7\_STM32H7\_Nucleo\_H743ZI\_IA↔  
R/Prog/boot.h, [1057](#)
- Demo/ARMCM7\_STM32H7\_Nucleo\_H743ZI\_↔  
Keil/Prog/boot.c, [928](#)
- Demo/ARMCM7\_STM32H7\_Nucleo\_H743ZI\_↔  
Keil/Prog/boot.h, [1059](#)
- Demo/HCS12\_Evbplus\_Dragon12p\_Code↔  
Warrior/Prog/boot.c, [933](#)
- Demo/HCS12\_Evbplus\_Dragon12p\_Code↔  
Warrior/Prog/boot.h, [1060](#)
- BootComInit
  - Demo/\_template/Prog/boot.c, [617](#)
  - Demo/\_template/Prog/boot.h, [938](#)
  - Demo/ARMCM0\_STM32F0\_Discovery\_STM32↔  
F051\_CubeIDE/Prog/App/boot.c, [620](#)
  - Demo/ARMCM0\_STM32F0\_Discovery\_STM32↔  
F051\_CubeIDE/Prog/App/boot.h, [940](#)
  - Demo/ARMCM0\_STM32F0\_Discovery\_STM32↔  
F051\_GCC/Prog/boot.c, [623](#)
  - Demo/ARMCM0\_STM32F0\_Discovery\_STM32↔  
F051\_GCC/Prog/boot.h, [942](#)
  - Demo/ARMCM0\_STM32F0\_Discovery\_STM32↔  
F051\_IAR/Prog/boot.c, [626](#)
  - Demo/ARMCM0\_STM32F0\_Discovery\_STM32↔  
F051\_IAR/Prog/boot.h, [943](#)
  - Demo/ARMCM0\_STM32F0\_Discovery\_STM32↔  
F051\_Keil/Prog/boot.c, [629](#)
  - Demo/ARMCM0\_STM32F0\_Discovery\_STM32↔  
F051\_Keil/Prog/boot.h, [945](#)
  - Demo/ARMCM0\_STM32F0\_Nucleo\_F091RC\_↔  
CubeIDE/Prog/App/boot.c, [633](#)
  - Demo/ARMCM0\_STM32F0\_Nucleo\_F091RC\_↔  
CubeIDE/Prog/App/boot.h, [947](#)
  - Demo/ARMCM0\_STM32F0\_Nucleo\_F091RC\_G↔  
CC/Prog/boot.c, [638](#)
  - Demo/ARMCM0\_STM32F0\_Nucleo\_F091RC\_G↔  
CC/Prog/boot.h, [948](#)
  - Demo/ARMCM0\_STM32F0\_Nucleo\_F091RC\_I↔  
AR/Prog/boot.c, [643](#)
  - Demo/ARMCM0\_STM32F0\_Nucleo\_F091RC\_I↔  
AR/Prog/boot.h, [950](#)
  - Demo/ARMCM0\_STM32F0\_Nucleo\_F091RC\_↔  
Keil/Prog/boot.c, [648](#)
  - Demo/ARMCM0\_STM32F0\_Nucleo\_F091RC\_↔  
Keil/Prog/boot.h, [952](#)

- Demo/ARMCM0\_STM32G0\_Nucleo\_G071RB\_↔  
CubeIDE/Prog/App/boot.c, [652](#)
- Demo/ARMCM0\_STM32G0\_Nucleo\_G071RB\_↔  
CubeIDE/Prog/App/boot.h, [953](#)
- Demo/ARMCM0\_STM32G0\_Nucleo\_G071RB\_↔  
GCC/Prog/boot.c, [655](#)
- Demo/ARMCM0\_STM32G0\_Nucleo\_G071RB\_↔  
GCC/Prog/boot.h, [955](#)
- Demo/ARMCM0\_STM32G0\_Nucleo\_G071RB\_I↔  
AR/Prog/boot.c, [658](#)
- Demo/ARMCM0\_STM32G0\_Nucleo\_G071RB\_I↔  
AR/Prog/boot.h, [957](#)
- Demo/ARMCM0\_STM32G0\_Nucleo\_G071RB\_↔  
Keil/Prog/boot.c, [661](#)
- Demo/ARMCM0\_STM32G0\_Nucleo\_G071RB\_↔  
Keil/Prog/boot.h, [958](#)
- Demo/ARMCM0\_XMC1\_XMC1400\_Boot\_Kit\_G↔  
CC/Prog/boot.c, [665](#)
- Demo/ARMCM0\_XMC1\_XMC1400\_Boot\_Kit\_G↔  
CC/Prog/boot.h, [960](#)
- Demo/ARMCM0\_XMC1\_XMC1400\_Boot\_Kit\_IA↔  
R/Prog/boot.c, [668](#)
- Demo/ARMCM0\_XMC1\_XMC1400\_Boot\_Kit\_IA↔  
R/Prog/boot.h, [962](#)
- Demo/ARMCM33\_STM32L5\_Nucleo\_L552ZE\_↔  
CubeIDE/Prog/App/boot.c, [672](#)
- Demo/ARMCM33\_STM32L5\_Nucleo\_L552ZE\_↔  
CubeIDE/Prog/App/boot.h, [963](#)
- Demo/ARMCM33\_STM32L5\_Nucleo\_L552ZE\_↔  
GCC/Prog/boot.c, [677](#)
- Demo/ARMCM33\_STM32L5\_Nucleo\_L552ZE\_↔  
GCC/Prog/boot.h, [965](#)
- Demo/ARMCM33\_STM32L5\_Nucleo\_L552ZE\_I↔  
AR/Prog/boot.c, [682](#)
- Demo/ARMCM33\_STM32L5\_Nucleo\_L552ZE\_I↔  
AR/Prog/boot.h, [967](#)
- Demo/ARMCM33\_STM32L5\_Nucleo\_L552ZE\_↔  
Keil/Prog/boot.c, [687](#)
- Demo/ARMCM33\_STM32L5\_Nucleo\_L552ZE\_↔  
Keil/Prog/boot.h, [968](#)
- Demo/ARMCM3\_EFM32\_Olimex\_EM32G880↔  
F128STK\_GCC/Prog/boot.c, [691](#)
- Demo/ARMCM3\_EFM32\_Olimex\_EM32G880↔  
F128STK\_GCC/Prog/boot.h, [970](#)
- Demo/ARMCM3\_EFM32\_Olimex\_EM32G880↔  
F128STK\_IAR/Prog/boot.c, [694](#)
- Demo/ARMCM3\_EFM32\_Olimex\_EM32G880↔  
F128STK\_IAR/Prog/boot.h, [971](#)
- Demo/ARMCM3\_LM3S\_EK\_LM3S6965\_GCC/↔  
Prog/boot.c, [697](#)
- Demo/ARMCM3\_LM3S\_EK\_LM3S6965\_GCC/↔  
Prog/boot.h, [973](#)
- Demo/ARMCM3\_LM3S\_EK\_LM3S6965\_IAR/↔  
Prog/boot.c, [699](#)
- Demo/ARMCM3\_LM3S\_EK\_LM3S6965\_IAR/↔  
Prog/boot.h, [974](#)
- Demo/ARMCM3\_LM3S\_EK\_LM3S8962\_GCC/↔  
Prog/boot.c, [703](#)
- Demo/ARMCM3\_LM3S\_EK\_LM3S8962\_GCC/↔  
Prog/boot.h, [976](#)
- Demo/ARMCM3\_LM3S\_EK\_LM3S8962\_IAR/↔  
Prog/boot.c, [707](#)
- Demo/ARMCM3\_LM3S\_EK\_LM3S8962\_IAR/↔  
Prog/boot.h, [977](#)
- Demo/ARMCM3\_STM32F1\_Nucleo\_F103RB\_↔  
CubeIDE/Prog/App/boot.c, [710](#)
- Demo/ARMCM3\_STM32F1\_Nucleo\_F103RB\_↔  
CubeIDE/Prog/App/boot.h, [979](#)
- Demo/ARMCM3\_STM32F1\_Nucleo\_F103RB\_G↔  
CC/Prog/boot.c, [713](#)
- Demo/ARMCM3\_STM32F1\_Nucleo\_F103RB\_G↔  
CC/Prog/boot.h, [981](#)
- Demo/ARMCM3\_STM32F1\_Nucleo\_F103RB\_I↔  
AR/Prog/boot.c, [716](#)
- Demo/ARMCM3\_STM32F1\_Nucleo\_F103RB\_I↔  
AR/Prog/boot.h, [982](#)
- Demo/ARMCM3\_STM32F1\_Nucleo\_F103RB\_↔  
Keil/Prog/boot.c, [719](#)
- Demo/ARMCM3\_STM32F1\_Nucleo\_F103RB\_↔  
Keil/Prog/boot.h, [984](#)
- Demo/ARMCM3\_STM32F1\_Olimex\_STM32↔  
P103\_CubeIDE/Prog/App/boot.c, [723](#)
- Demo/ARMCM3\_STM32F1\_Olimex\_STM32↔  
P103\_CubeIDE/Prog/App/boot.h, [986](#)
- Demo/ARMCM3\_STM32F1\_Olimex\_STM32↔  
P103\_GCC/Prog/boot.c, [728](#)
- Demo/ARMCM3\_STM32F1\_Olimex\_STM32↔  
P103\_GCC/Prog/boot.h, [987](#)
- Demo/ARMCM3\_STM32F1\_Olimex\_STM32↔  
P103\_IAR/Prog/boot.c, [733](#)
- Demo/ARMCM3\_STM32F1\_Olimex\_STM32↔  
P103\_IAR/Prog/boot.h, [989](#)
- Demo/ARMCM3\_STM32F1\_Olimex\_STM32↔  
P103\_Keil/Prog/boot.c, [738](#)
- Demo/ARMCM3\_STM32F1\_Olimex\_STM32↔  
P103\_Keil/Prog/boot.h, [991](#)
- Demo/ARMCM3\_STM32F1\_Olimexino\_STM32↔  
\_CubeIDE/Prog/App/boot.c, [743](#)
- Demo/ARMCM3\_STM32F1\_Olimexino\_STM32↔  
\_CubeIDE/Prog/App/boot.h, [992](#)
- Demo/ARMCM3\_STM32F1\_Olimexino\_STM32↔  
\_GCC/Prog/boot.c, [746](#)
- Demo/ARMCM3\_STM32F1\_Olimexino\_STM32↔  
\_GCC/Prog/boot.h, [994](#)
- Demo/ARMCM3\_STM32F1\_Olimexino\_STM32↔  
\_IAR/Prog/boot.c, [750](#)
- Demo/ARMCM3\_STM32F1\_Olimexino\_STM32↔  
\_IAR/Prog/boot.h, [996](#)
- Demo/ARMCM3\_STM32F1\_Olimexino\_STM32↔  
\_Keil/Prog/boot.c, [754](#)
- Demo/ARMCM3\_STM32F1\_Olimexino\_STM32↔  
\_Keil/Prog/boot.h, [997](#)
- Demo/ARMCM3\_STM32F2\_Olimex\_STM32↔  
P207\_CubeIDE/Prog/App/boot.c, [759](#)
- Demo/ARMCM3\_STM32F2\_Olimex\_STM32↔  
P207\_CubeIDE/Prog/App/boot.h, [999](#)



- Demo/ARMCM3\_STM32F2\_Olimex\_STM32↔  
P207\_GCC/Prog/boot.c, [764](#)
- Demo/ARMCM3\_STM32F2\_Olimex\_STM32↔  
P207\_GCC/Prog/boot.h, [1001](#)
- Demo/ARMCM3\_STM32F2\_Olimex\_STM32↔  
P207\_IAR/Prog/boot.c, [769](#)
- Demo/ARMCM3\_STM32F2\_Olimex\_STM32↔  
P207\_IAR/Prog/boot.h, [1002](#)
- Demo/ARMCM3\_STM32F2\_Olimex\_STM32↔  
P207\_Keil/Prog/boot.c, [774](#)
- Demo/ARMCM3\_STM32F2\_Olimex\_STM32↔  
P207\_Keil/Prog/boot.h, [1004](#)
- Demo/ARMCM4\_S32K14\_S32K144EVB\_GCC↔  
Prog/boot.c, [779](#)
- Demo/ARMCM4\_S32K14\_S32K144EVB\_GCC↔  
Prog/boot.h, [1006](#)
- Demo/ARMCM4\_S32K14\_S32K144EVB\_IAR↔  
Prog/boot.c, [786](#)
- Demo/ARMCM4\_S32K14\_S32K144EVB\_IAR↔  
Prog/boot.h, [1007](#)
- Demo/ARMCM4\_STM32F3\_Nucleo\_F303K8↔  
CubeIDE/Prog/App/boot.c, [792](#)
- Demo/ARMCM4\_STM32F3\_Nucleo\_F303K8↔  
CubeIDE/Prog/App/boot.h, [1009](#)
- Demo/ARMCM4\_STM32F3\_Nucleo\_F303K8↔  
GCC/Prog/boot.c, [797](#)
- Demo/ARMCM4\_STM32F3\_Nucleo\_F303K8↔  
GCC/Prog/boot.h, [1011](#)
- Demo/ARMCM4\_STM32F3\_Nucleo\_F303K8↔  
IAR/Prog/boot.c, [802](#)
- Demo/ARMCM4\_STM32F3\_Nucleo\_F303K8↔  
IAR/Prog/boot.h, [1012](#)
- Demo/ARMCM4\_STM32F3\_Nucleo\_F303K8↔  
Keil/Prog/boot.c, [807](#)
- Demo/ARMCM4\_STM32F3\_Nucleo\_F303K8↔  
Keil/Prog/boot.h, [1014](#)
- Demo/ARMCM4\_STM32F4\_Nucleo\_F429ZI↔  
CubeIDE/Prog/App/boot.c, [812](#)
- Demo/ARMCM4\_STM32F4\_Nucleo\_F429ZI↔  
CubeIDE/Prog/App/boot.h, [1016](#)
- Demo/ARMCM4\_STM32F4\_Nucleo\_F429ZI↔  
GCC/Prog/boot.c, [817](#)
- Demo/ARMCM4\_STM32F4\_Nucleo\_F429ZI↔  
GCC/Prog/boot.h, [1017](#)
- Demo/ARMCM4\_STM32F4\_Nucleo\_F429ZI↔  
IAR/Prog/boot.c, [822](#)
- Demo/ARMCM4\_STM32F4\_Nucleo\_F429ZI↔  
IAR/Prog/boot.h, [1019](#)
- Demo/ARMCM4\_STM32F4\_Nucleo\_F429ZI↔  
Keil/Prog/boot.c, [827](#)
- Demo/ARMCM4\_STM32F4\_Nucleo\_F429ZI↔  
Keil/Prog/boot.h, [1021](#)
- Demo/ARMCM4\_STM32F4\_Olimex\_STM32↔  
P405\_CubeIDE/Prog/App/boot.c, [832](#)
- Demo/ARMCM4\_STM32F4\_Olimex\_STM32↔  
P405\_CubeIDE/Prog/App/boot.h, [1022](#)
- Demo/ARMCM4\_STM32F4\_Olimex\_STM32↔  
P405\_GCC/Prog/boot.c, [837](#)
- Demo/ARMCM4\_STM32F4\_Olimex\_STM32↔  
P405\_GCC/Prog/boot.h, [1024](#)
- Demo/ARMCM4\_STM32F4\_Olimex\_STM32↔  
P405\_IAR/Prog/boot.c, [842](#)
- Demo/ARMCM4\_STM32F4\_Olimex\_STM32↔  
P405\_IAR/Prog/boot.h, [1026](#)
- Demo/ARMCM4\_STM32F4\_Olimex\_STM32↔  
P405\_Keil/Prog/boot.c, [847](#)
- Demo/ARMCM4\_STM32F4\_Olimex\_STM32↔  
P405\_Keil/Prog/boot.h, [1027](#)
- Demo/ARMCM4\_STM32L4\_Nucleo\_L476RG↔  
CubeIDE/Prog/App/boot.c, [852](#)
- Demo/ARMCM4\_STM32L4\_Nucleo\_L476RG↔  
CubeIDE/Prog/App/boot.h, [1029](#)
- Demo/ARMCM4\_STM32L4\_Nucleo\_L476RG↔  
GCC/Prog/boot.c, [857](#)
- Demo/ARMCM4\_STM32L4\_Nucleo\_L476RG↔  
GCC/Prog/boot.h, [1031](#)
- Demo/ARMCM4\_STM32L4\_Nucleo\_L476RG↔  
IAR/Prog/boot.c, [862](#)
- Demo/ARMCM4\_STM32L4\_Nucleo\_L476RG↔  
IAR/Prog/boot.h, [1032](#)
- Demo/ARMCM4\_STM32L4\_Nucleo\_L476RG↔  
Keil/Prog/boot.c, [867](#)
- Demo/ARMCM4\_STM32L4\_Nucleo\_L476RG↔  
Keil/Prog/boot.h, [1034](#)
- Demo/ARMCM4\_TM4C\_DK\_TM4C123G\_IAR↔  
Prog/boot.c, [871](#)
- Demo/ARMCM4\_TM4C\_DK\_TM4C123G\_IAR↔  
Prog/boot.h, [1036](#)
- Demo/ARMCM4\_XMC4\_XMC4700\_Relax\_Kit↔  
GCC/Prog/boot.c, [874](#)
- Demo/ARMCM4\_XMC4\_XMC4700\_Relax\_Kit↔  
GCC/Prog/boot.h, [1037](#)
- Demo/ARMCM4\_XMC4\_XMC4700\_Relax\_Kit↔  
IAR/Prog/boot.c, [878](#)
- Demo/ARMCM4\_XMC4\_XMC4700\_Relax\_Kit↔  
IAR/Prog/boot.h, [1039](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F746ZG↔  
CubeIDE/Prog/App/boot.c, [881](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F746ZG↔  
CubeIDE/Prog/App/boot.h, [1041](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F746ZG↔  
GCC/Prog/boot.c, [886](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F746ZG↔  
GCC/Prog/boot.h, [1042](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F746ZG↔  
IAR/Prog/boot.c, [891](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F746ZG↔  
IAR/Prog/boot.h, [1044](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F746ZG↔  
Keil/Prog/boot.c, [896](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F746ZG↔  
Keil/Prog/boot.h, [1046](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F767ZI↔  
CubeIDE/Prog/App/boot.c, [901](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F767ZI↔  
CubeIDE/Prog/App/boot.h, [1047](#)

- Demo/ARMCM7\_STM32F7\_Nucleo\_F767ZI\_G↔  
CC/Prog/boot.c, [903](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F767ZI\_G↔  
CC/Prog/boot.h, [1049](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F767ZI\_IA↔  
R/Prog/boot.c, [906](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F767ZI\_IA↔  
R/Prog/boot.h, [1051](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F767ZI\_↔  
Keil/Prog/boot.c, [909](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F767ZI\_↔  
Keil/Prog/boot.h, [1052](#)
- Demo/ARMCM7\_STM32H7\_Nucleo\_H743ZI\_↔  
CubeIDE/Prog/App/boot.c, [913](#)
- Demo/ARMCM7\_STM32H7\_Nucleo\_H743ZI\_↔  
CubeIDE/Prog/App/boot.h, [1054](#)
- Demo/ARMCM7\_STM32H7\_Nucleo\_H743ZI\_G↔  
CC/Prog/boot.c, [918](#)
- Demo/ARMCM7\_STM32H7\_Nucleo\_H743ZI\_G↔  
CC/Prog/boot.h, [1056](#)
- Demo/ARMCM7\_STM32H7\_Nucleo\_H743ZI\_IA↔  
R/Prog/boot.c, [923](#)
- Demo/ARMCM7\_STM32H7\_Nucleo\_H743ZI\_IA↔  
R/Prog/boot.h, [1057](#)
- Demo/ARMCM7\_STM32H7\_Nucleo\_H743ZI\_↔  
Keil/Prog/boot.c, [928](#)
- Demo/ARMCM7\_STM32H7\_Nucleo\_H743ZI\_↔  
Keil/Prog/boot.h, [1059](#)
- Demo/HCS12\_Evbplus\_Dragon12p\_Code↔  
Warrior/Prog/boot.c, [933](#)
- Demo/HCS12\_Evbplus\_Dragon12p\_Code↔  
Warrior/Prog/boot.h, [1061](#)
- BootComRs232CheckActivationRequest
  - Demo/\_template/Prog/boot.c, [617](#)
  - Demo/ARMCM0\_STM32F0\_Discovery\_STM32↔  
F051\_CubeIDE/Prog/App/boot.c, [620](#)
  - Demo/ARMCM0\_STM32F0\_Discovery\_STM32↔  
F051\_GCC/Prog/boot.c, [623](#)
  - Demo/ARMCM0\_STM32F0\_Discovery\_STM32↔  
F051\_IAR/Prog/boot.c, [626](#)
  - Demo/ARMCM0\_STM32F0\_Discovery\_STM32↔  
F051\_Keil/Prog/boot.c, [629](#)
  - Demo/ARMCM0\_STM32F0\_Nucleo\_F091RC\_↔  
CubeIDE/Prog/App/boot.c, [633](#)
  - Demo/ARMCM0\_STM32F0\_Nucleo\_F091RC\_G↔  
CC/Prog/boot.c, [638](#)
  - Demo/ARMCM0\_STM32F0\_Nucleo\_F091RC\_I↔  
AR/Prog/boot.c, [643](#)
  - Demo/ARMCM0\_STM32F0\_Nucleo\_F091RC\_↔  
Keil/Prog/boot.c, [648](#)
  - Demo/ARMCM0\_STM32G0\_Nucleo\_G071RB\_↔  
CubeIDE/Prog/App/boot.c, [652](#)
  - Demo/ARMCM0\_STM32G0\_Nucleo\_G071RB\_↔  
GCC/Prog/boot.c, [655](#)
  - Demo/ARMCM0\_STM32G0\_Nucleo\_G071RB\_I↔  
AR/Prog/boot.c, [658](#)
  - Demo/ARMCM0\_STM32G0\_Nucleo\_G071RB\_↔  
Keil/Prog/boot.c, [661](#)
- Demo/ARMCM0\_XMC1\_XMC1400\_Boot\_Kit\_G↔  
CC/Prog/boot.c, [665](#)
- Demo/ARMCM0\_XMC1\_XMC1400\_Boot\_Kit\_IA↔  
R/Prog/boot.c, [669](#)
- Demo/ARMCM33\_STM32L5\_Nucleo\_L552ZE\_↔  
CubeIDE/Prog/App/boot.c, [673](#)
- Demo/ARMCM33\_STM32L5\_Nucleo\_L552ZE\_↔  
GCC/Prog/boot.c, [678](#)
- Demo/ARMCM33\_STM32L5\_Nucleo\_L552ZE\_I↔  
AR/Prog/boot.c, [683](#)
- Demo/ARMCM33\_STM32L5\_Nucleo\_L552ZE\_↔  
Keil/Prog/boot.c, [688](#)
- Demo/ARMCM3\_EFM32\_Olimex\_EM32G880↔  
F128STK\_GCC/Prog/boot.c, [692](#)
- Demo/ARMCM3\_EFM32\_Olimex\_EM32G880↔  
F128STK\_IAR/Prog/boot.c, [694](#)
- Demo/ARMCM3\_LM3S\_EK\_LM3S6965\_GCC↔  
Prog/boot.c, [697](#)
- Demo/ARMCM3\_LM3S\_EK\_LM3S6965\_IAR↔  
Prog/boot.c, [700](#)
- Demo/ARMCM3\_LM3S\_EK\_LM3S8962\_GCC↔  
Prog/boot.c, [703](#)
- Demo/ARMCM3\_LM3S\_EK\_LM3S8962\_IAR↔  
Prog/boot.c, [707](#)
- Demo/ARMCM3\_STM32F1\_Nucleo\_F103RB\_↔  
CubeIDE/Prog/App/boot.c, [710](#)
- Demo/ARMCM3\_STM32F1\_Nucleo\_F103RB\_G↔  
CC/Prog/boot.c, [713](#)
- Demo/ARMCM3\_STM32F1\_Nucleo\_F103RB\_I↔  
AR/Prog/boot.c, [716](#)
- Demo/ARMCM3\_STM32F1\_Nucleo\_F103RB\_↔  
Keil/Prog/boot.c, [719](#)
- Demo/ARMCM3\_STM32F1\_Olimex\_STM32↔  
P103\_CubeIDE/Prog/App/boot.c, [723](#)
- Demo/ARMCM3\_STM32F1\_Olimex\_STM32↔  
P103\_GCC/Prog/boot.c, [728](#)
- Demo/ARMCM3\_STM32F1\_Olimex\_STM32↔  
P103\_IAR/Prog/boot.c, [733](#)
- Demo/ARMCM3\_STM32F1\_Olimex\_STM32↔  
P103\_Keil/Prog/boot.c, [738](#)
- Demo/ARMCM3\_STM32F2\_Olimex\_STM32↔  
P207\_CubeIDE/Prog/App/boot.c, [759](#)
- Demo/ARMCM3\_STM32F2\_Olimex\_STM32↔  
P207\_GCC/Prog/boot.c, [764](#)
- Demo/ARMCM3\_STM32F2\_Olimex\_STM32↔  
P207\_IAR/Prog/boot.c, [769](#)
- Demo/ARMCM3\_STM32F2\_Olimex\_STM32↔  
P207\_Keil/Prog/boot.c, [774](#)
- Demo/ARMCM4\_S32K14\_S32K144EVB\_GCC↔  
Prog/boot.c, [779](#)
- Demo/ARMCM4\_S32K14\_S32K144EVB\_IAR↔  
Prog/boot.c, [786](#)
- Demo/ARMCM4\_STM32F3\_Nucleo\_F303K8\_↔  
CubeIDE/Prog/App/boot.c, [792](#)
- Demo/ARMCM4\_STM32F3\_Nucleo\_F303K8\_G↔  
CC/Prog/boot.c, [797](#)
- Demo/ARMCM4\_STM32F3\_Nucleo\_F303K8\_IA↔  
R/Prog/boot.c, [802](#)

- Demo/ARMCM4\_STM32F3\_Nucleo\_F303K8\_↔  
Keil/Prog/boot.c, [807](#)
- Demo/ARMCM4\_STM32F4\_Nucleo\_F429ZI\_↔  
CubeIDE/Prog/App/boot.c, [812](#)
- Demo/ARMCM4\_STM32F4\_Nucleo\_F429ZI\_G↔  
CC/Prog/boot.c, [817](#)
- Demo/ARMCM4\_STM32F4\_Nucleo\_F429ZI\_IA↔  
R/Prog/boot.c, [822](#)
- Demo/ARMCM4\_STM32F4\_Nucleo\_F429ZI\_↔  
Keil/Prog/boot.c, [827](#)
- Demo/ARMCM4\_STM32F4\_Olimex\_STM32↔  
P405\_CubeIDE/Prog/App/boot.c, [832](#)
- Demo/ARMCM4\_STM32F4\_Olimex\_STM32↔  
P405\_GCC/Prog/boot.c, [837](#)
- Demo/ARMCM4\_STM32F4\_Olimex\_STM32↔  
P405\_IAR/Prog/boot.c, [842](#)
- Demo/ARMCM4\_STM32F4\_Olimex\_STM32↔  
P405\_Keil/Prog/boot.c, [847](#)
- Demo/ARMCM4\_STM32L4\_Nucleo\_L476RG\_↔  
CubeIDE/Prog/App/boot.c, [852](#)
- Demo/ARMCM4\_STM32L4\_Nucleo\_L476RG\_G↔  
CC/Prog/boot.c, [857](#)
- Demo/ARMCM4\_STM32L4\_Nucleo\_L476RG\_I↔  
AR/Prog/boot.c, [862](#)
- Demo/ARMCM4\_STM32L4\_Nucleo\_L476RG\_↔  
Keil/Prog/boot.c, [867](#)
- Demo/ARMCM4\_TM4C\_DK\_TM4C123G\_IAR↔  
Prog/boot.c, [871](#)
- Demo/ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_↔  
GCC/Prog/boot.c, [874](#)
- Demo/ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_I↔  
AR/Prog/boot.c, [878](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F746ZG\_↔  
CubeIDE/Prog/App/boot.c, [882](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F746ZG\_G↔  
CC/Prog/boot.c, [887](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F746ZG\_I↔  
AR/Prog/boot.c, [892](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F746ZG\_↔  
Keil/Prog/boot.c, [897](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F767ZI\_↔  
CubeIDE/Prog/App/boot.c, [901](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F767ZI\_G↔  
CC/Prog/boot.c, [904](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F767ZI\_IA↔  
R/Prog/boot.c, [907](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F767ZI\_↔  
Keil/Prog/boot.c, [910](#)
- Demo/ARMCM7\_STM32H7\_Nucleo\_H743ZI\_↔  
CubeIDE/Prog/App/boot.c, [914](#)
- Demo/ARMCM7\_STM32H7\_Nucleo\_H743ZI\_G↔  
CC/Prog/boot.c, [919](#)
- Demo/ARMCM7\_STM32H7\_Nucleo\_H743ZI\_IA↔  
R/Prog/boot.c, [924](#)
- Demo/ARMCM7\_STM32H7\_Nucleo\_H743ZI\_↔  
Keil/Prog/boot.c, [929](#)
- Demo/HCS12\_Evbplus\_Dragon12p\_Code↔  
Warrior/Prog/boot.c, [934](#)
- BootComRs232Init  
Demo/\_template/Prog/boot.c, [618](#)
- Demo/ARMCM0\_STM32F0\_Discovery\_STM32↔  
F051\_CubeIDE/Prog/App/boot.c, [621](#)
- Demo/ARMCM0\_STM32F0\_Discovery\_STM32↔  
F051\_GCC/Prog/boot.c, [624](#)
- Demo/ARMCM0\_STM32F0\_Discovery\_STM32↔  
F051\_IAR/Prog/boot.c, [627](#)
- Demo/ARMCM0\_STM32F0\_Discovery\_STM32↔  
F051\_Keil/Prog/boot.c, [630](#)
- Demo/ARMCM0\_STM32F0\_Nucleo\_F091RC\_↔  
CubeIDE/Prog/App/boot.c, [633](#)
- Demo/ARMCM0\_STM32F0\_Nucleo\_F091RC\_G↔  
CC/Prog/boot.c, [638](#)
- Demo/ARMCM0\_STM32F0\_Nucleo\_F091RC\_I↔  
AR/Prog/boot.c, [643](#)
- Demo/ARMCM0\_STM32F0\_Nucleo\_F091RC\_↔  
Keil/Prog/boot.c, [648](#)
- Demo/ARMCM0\_STM32G0\_Nucleo\_G071RB\_↔  
CubeIDE/Prog/App/boot.c, [653](#)
- Demo/ARMCM0\_STM32G0\_Nucleo\_G071RB\_↔  
GCC/Prog/boot.c, [656](#)
- Demo/ARMCM0\_STM32G0\_Nucleo\_G071RB\_I↔  
AR/Prog/boot.c, [659](#)
- Demo/ARMCM0\_STM32G0\_Nucleo\_G071RB\_↔  
Keil/Prog/boot.c, [662](#)
- Demo/ARMCM0\_XMC1\_XMC1400\_Boot\_Kit\_G↔  
CC/Prog/boot.c, [665](#)
- Demo/ARMCM0\_XMC1\_XMC1400\_Boot\_Kit\_IA↔  
R/Prog/boot.c, [669](#)
- Demo/ARMCM33\_STM32L5\_Nucleo\_L552ZE\_↔  
CubeIDE/Prog/App/boot.c, [673](#)
- Demo/ARMCM33\_STM32L5\_Nucleo\_L552ZE\_↔  
GCC/Prog/boot.c, [678](#)
- Demo/ARMCM33\_STM32L5\_Nucleo\_L552ZE\_I↔  
AR/Prog/boot.c, [683](#)
- Demo/ARMCM33\_STM32L5\_Nucleo\_L552ZE\_↔  
Keil/Prog/boot.c, [688](#)
- Demo/ARMCM3\_EFM32\_Olimex\_EM32G880↔  
F128STK\_GCC/Prog/boot.c, [692](#)
- Demo/ARMCM3\_EFM32\_Olimex\_EM32G880↔  
F128STK\_IAR/Prog/boot.c, [695](#)
- Demo/ARMCM3\_LM3S\_EK\_LM3S6965\_GCC↔  
Prog/boot.c, [697](#)
- Demo/ARMCM3\_LM3S\_EK\_LM3S6965\_IAR↔  
Prog/boot.c, [700](#)
- Demo/ARMCM3\_LM3S\_EK\_LM3S8962\_GCC↔  
Prog/boot.c, [703](#)
- Demo/ARMCM3\_LM3S\_EK\_LM3S8962\_IAR↔  
Prog/boot.c, [707](#)
- Demo/ARMCM3\_STM32F1\_Nucleo\_F103RB\_↔  
CubeIDE/Prog/App/boot.c, [711](#)
- Demo/ARMCM3\_STM32F1\_Nucleo\_F103RB\_G↔  
CC/Prog/boot.c, [714](#)
- Demo/ARMCM3\_STM32F1\_Nucleo\_F103RB\_I↔  
AR/Prog/boot.c, [717](#)
- Demo/ARMCM3\_STM32F1\_Nucleo\_F103RB\_↔  
Keil/Prog/boot.c, [720](#)

- Demo/ARMCM3\_STM32F1\_Olimex\_STM32↔  
P103\_CubeIDE/Prog/App/boot.c, [723](#)
- Demo/ARMCM3\_STM32F1\_Olimex\_STM32↔  
P103\_GCC/Prog/boot.c, [728](#)
- Demo/ARMCM3\_STM32F1\_Olimex\_STM32↔  
P103\_IAR/Prog/boot.c, [733](#)
- Demo/ARMCM3\_STM32F1\_Olimex\_STM32↔  
P103\_Keil/Prog/boot.c, [738](#)
- Demo/ARMCM3\_STM32F2\_Olimex\_STM32↔  
P207\_CubeIDE/Prog/App/boot.c, [759](#)
- Demo/ARMCM3\_STM32F2\_Olimex\_STM32↔  
P207\_GCC/Prog/boot.c, [764](#)
- Demo/ARMCM3\_STM32F2\_Olimex\_STM32↔  
P207\_IAR/Prog/boot.c, [769](#)
- Demo/ARMCM3\_STM32F2\_Olimex\_STM32↔  
P207\_Keil/Prog/boot.c, [774](#)
- Demo/ARMCM4\_S32K14\_S32K144EVB\_GCC↔  
Prog/boot.c, [779](#)
- Demo/ARMCM4\_S32K14\_S32K144EVB\_IAR↔  
Prog/boot.c, [786](#)
- Demo/ARMCM4\_STM32F3\_Nucleo\_F303K8↔  
CubeIDE/Prog/App/boot.c, [792](#)
- Demo/ARMCM4\_STM32F3\_Nucleo\_F303K8\_G↔  
CC/Prog/boot.c, [797](#)
- Demo/ARMCM4\_STM32F3\_Nucleo\_F303K8\_IA↔  
R/Prog/boot.c, [802](#)
- Demo/ARMCM4\_STM32F3\_Nucleo\_F303K8↔  
Keil/Prog/boot.c, [807](#)
- Demo/ARMCM4\_STM32F4\_Nucleo\_F429ZI↔  
CubeIDE/Prog/App/boot.c, [812](#)
- Demo/ARMCM4\_STM32F4\_Nucleo\_F429ZI\_G↔  
CC/Prog/boot.c, [817](#)
- Demo/ARMCM4\_STM32F4\_Nucleo\_F429ZI\_IA↔  
R/Prog/boot.c, [822](#)
- Demo/ARMCM4\_STM32F4\_Nucleo\_F429ZI↔  
Keil/Prog/boot.c, [827](#)
- Demo/ARMCM4\_STM32F4\_Olimex\_STM32↔  
P405\_CubeIDE/Prog/App/boot.c, [832](#)
- Demo/ARMCM4\_STM32F4\_Olimex\_STM32↔  
P405\_GCC/Prog/boot.c, [837](#)
- Demo/ARMCM4\_STM32F4\_Olimex\_STM32↔  
P405\_IAR/Prog/boot.c, [842](#)
- Demo/ARMCM4\_STM32F4\_Olimex\_STM32↔  
P405\_Keil/Prog/boot.c, [847](#)
- Demo/ARMCM4\_STM32L4\_Nucleo\_L476RG↔  
CubeIDE/Prog/App/boot.c, [852](#)
- Demo/ARMCM4\_STM32L4\_Nucleo\_L476RG\_G↔  
CC/Prog/boot.c, [857](#)
- Demo/ARMCM4\_STM32L4\_Nucleo\_L476RG\_I↔  
AR/Prog/boot.c, [862](#)
- Demo/ARMCM4\_STM32L4\_Nucleo\_L476RG↔  
Keil/Prog/boot.c, [867](#)
- Demo/ARMCM4\_TM4C\_DK\_TM4C123G\_IAR↔  
Prog/boot.c, [871](#)
- Demo/ARMCM4\_XMC4\_XMC4700\_Relax\_Kit↔  
GCC/Prog/boot.c, [874](#)
- Demo/ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_I↔  
AR/Prog/boot.c, [878](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F746ZG↔  
CubeIDE/Prog/App/boot.c, [882](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F746ZG\_G↔  
CC/Prog/boot.c, [887](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F746ZG\_I↔  
AR/Prog/boot.c, [892](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F746ZG↔  
Keil/Prog/boot.c, [897](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F767ZI↔  
CubeIDE/Prog/App/boot.c, [901](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F767ZI\_G↔  
CC/Prog/boot.c, [904](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F767ZI\_IA↔  
R/Prog/boot.c, [907](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F767ZI↔  
Keil/Prog/boot.c, [910](#)
- Demo/ARMCM7\_STM32H7\_Nucleo\_H743ZI↔  
CubeIDE/Prog/App/boot.c, [914](#)
- Demo/ARMCM7\_STM32H7\_Nucleo\_H743ZI\_G↔  
CC/Prog/boot.c, [919](#)
- Demo/ARMCM7\_STM32H7\_Nucleo\_H743ZI\_IA↔  
R/Prog/boot.c, [924](#)
- Demo/ARMCM7\_STM32H7\_Nucleo\_H743ZI↔  
Keil/Prog/boot.c, [929](#)
- Demo/HCS12\_Evbplus\_Dragon12p\_Code↔  
Warrior/Prog/boot.c, [934](#)
- BootInit
  - Source/boot.c, [937](#)
  - Source/boot.h, [1062](#)
- BootTask
  - Source/boot.c, [937](#)
  - Source/boot.h, [1062](#)
- Bootloader, [61](#), [64](#), [67](#), [70](#), [73](#), [76](#), [79](#), [82](#), [85](#), [88](#), [91](#), [94](#),  
[97](#), [100](#), [103](#), [106](#), [109](#), [112](#), [115](#), [118](#), [121](#),  
[124](#), [127](#), [130](#), [133](#), [136](#), [139](#), [142](#), [145](#), [148](#),  
[151](#), [154](#), [157](#), [160](#), [163](#), [166](#), [169](#), [172](#), [175](#),  
[178](#), [181](#), [184](#), [187](#), [190](#), [193](#), [196](#), [199](#), [202](#),  
[205](#), [208](#), [211](#), [214](#), [217](#), [220](#), [223](#), [226](#), [229](#),  
[232](#), [235](#), [238](#), [241](#), [244](#), [247](#), [250](#), [253](#), [256](#),  
[259](#), [262](#), [265](#), [268](#), [271](#), [274](#), [277](#), [280](#), [283](#),  
[286](#), [289](#), [292](#), [295](#), [298](#), [301](#), [304](#), [307](#), [310](#),  
[317](#), [320](#)
- Bootloader Core, [350](#)
- Bootloader Demos, [313](#)
- Bootloader Ports, [353](#)
- CAN driver of a port, [323](#)
- CPU driver of a port, [324](#)
- CPU\_USER\_PROGRAM\_STARTADDR\_PTR  
HCS12/cpu.c, [1185](#)
- CTRL
  - tSysTickRegs, [380](#)
- can.c, [1063](#), [1066](#), [1071](#), [1075](#), [1077](#), [1081](#), [1083](#), [1086](#),  
[1090](#), [1095](#), [1099](#), [1102](#), [1105](#), [1108](#), [1111](#),  
[1115](#)
- can.h, [1119](#)
  - CanInit, [1120](#)
  - CanReceivePacket, [1120](#)
  - CanTransmitPacket, [1120](#)



- CanDisabledModeEnter
  - ARMCM0\_S32K11/can.c, [1068](#)
  - ARMCM4\_S32K14/can.c, [1091](#)
  - Demo/ARMCM4\_S32K14\_S32K144EVB\_GCC/↔  
Prog/boot.c, [780](#)
  - Demo/ARMCM4\_S32K14\_S32K144EVB\_IAR/↔  
Prog/boot.c, [786](#)
- CanDisabledModeExit
  - ARMCM0\_S32K11/can.c, [1068](#)
  - ARMCM4\_S32K14/can.c, [1092](#)
  - Demo/ARMCM4\_S32K14\_S32K144EVB\_GCC/↔  
Prog/boot.c, [780](#)
  - Demo/ARMCM4\_S32K14\_S32K144EVB\_IAR/↔  
Prog/boot.c, [787](#)
- CanFreezeModeEnter
  - ARMCM0\_S32K11/can.c, [1068](#)
  - ARMCM4\_S32K14/can.c, [1092](#)
  - Demo/ARMCM4\_S32K14\_S32K144EVB\_GCC/↔  
Prog/boot.c, [780](#)
  - Demo/ARMCM4\_S32K14\_S32K144EVB\_IAR/↔  
Prog/boot.c, [787](#)
- CanFreezeModeExit
  - ARMCM0\_S32K11/can.c, [1068](#)
  - ARMCM4\_S32K14/can.c, [1092](#)
  - Demo/ARMCM4\_S32K14\_S32K144EVB\_GCC/↔  
Prog/boot.c, [781](#)
  - Demo/ARMCM4\_S32K14\_S32K144EVB\_IAR/↔  
Prog/boot.c, [787](#)
- CanGetSpeedConfig
  - \_template/can.c, [1064](#)
  - ARMCM0\_S32K11/can.c, [1069](#)
  - ARMCM0\_STM32F0/can.c, [1072](#)
  - ARMCM33\_STM32L5/can.c, [1078](#)
  - ARMCM3\_STM32F1/can.c, [1084](#)
  - ARMCM3\_STM32F2/can.c, [1088](#)
  - ARMCM4\_S32K14/can.c, [1093](#)
  - ARMCM4\_STM32F3/can.c, [1096](#)
  - ARMCM4\_STM32F4/can.c, [1100](#)
  - ARMCM4\_STM32L4/can.c, [1103](#)
  - ARMCM7\_STM32F7/can.c, [1109](#)
  - ARMCM7\_STM32H7/can.c, [1113](#)
  - Demo/ARMCM0\_STM32F0\_Nucleo\_F091RC\_↔  
CubeIDE/Prog/App/boot.c, [634](#)
  - Demo/ARMCM0\_STM32F0\_Nucleo\_F091RC\_G↔  
CC/Prog/boot.c, [639](#)
  - Demo/ARMCM0\_STM32F0\_Nucleo\_F091RC\_I↔  
AR/Prog/boot.c, [644](#)
  - Demo/ARMCM0\_STM32F0\_Nucleo\_F091RC\_↔  
Keil/Prog/boot.c, [649](#)
  - Demo/ARMCM33\_STM32L5\_Nucleo\_L552ZE\_↔  
CubeIDE/Prog/App/boot.c, [673](#)
  - Demo/ARMCM33\_STM32L5\_Nucleo\_L552ZE\_↔  
GCC/Prog/boot.c, [678](#)
  - Demo/ARMCM33\_STM32L5\_Nucleo\_L552ZE\_I↔  
AR/Prog/boot.c, [683](#)
  - Demo/ARMCM33\_STM32L5\_Nucleo\_L552ZE\_↔  
Keil/Prog/boot.c, [688](#)
  - Demo/ARMCM3\_STM32F1\_Olimex\_STM32↔  
P103\_CubeIDE/Prog/App/boot.c, [724](#)
  - Demo/ARMCM3\_STM32F1\_Olimex\_STM32↔  
P103\_GCC/Prog/boot.c, [729](#)
  - Demo/ARMCM3\_STM32F1\_Olimex\_STM32↔  
P103\_IAR/Prog/boot.c, [734](#)
  - Demo/ARMCM3\_STM32F1\_Olimex\_STM32↔  
P103\_Keil/Prog/boot.c, [739](#)
  - Demo/ARMCM3\_STM32F1\_Olimexino\_STM32↔  
\_CubeIDE/Prog/App/boot.c, [743](#)
  - Demo/ARMCM3\_STM32F1\_Olimexino\_STM32↔  
\_GCC/Prog/boot.c, [747](#)
  - Demo/ARMCM3\_STM32F1\_Olimexino\_STM32↔  
\_IAR/Prog/boot.c, [751](#)
  - Demo/ARMCM3\_STM32F1\_Olimexino\_STM32↔  
\_Keil/Prog/boot.c, [755](#)
  - Demo/ARMCM3\_STM32F2\_Olimex\_STM32↔  
P207\_CubeIDE/Prog/App/boot.c, [759](#)
  - Demo/ARMCM3\_STM32F2\_Olimex\_STM32↔  
P207\_GCC/Prog/boot.c, [764](#)
  - Demo/ARMCM3\_STM32F2\_Olimex\_STM32↔  
P207\_IAR/Prog/boot.c, [769](#)
  - Demo/ARMCM3\_STM32F2\_Olimex\_STM32↔  
P207\_Keil/Prog/boot.c, [774](#)
  - Demo/ARMCM4\_S32K14\_S32K144EVB\_GCC/↔  
Prog/boot.c, [781](#)
  - Demo/ARMCM4\_S32K14\_S32K144EVB\_IAR/↔  
Prog/boot.c, [788](#)
  - Demo/ARMCM4\_STM32F3\_Nucleo\_F303K8\_↔  
CubeIDE/Prog/App/boot.c, [793](#)
  - Demo/ARMCM4\_STM32F3\_Nucleo\_F303K8\_G↔  
CC/Prog/boot.c, [798](#)
  - Demo/ARMCM4\_STM32F3\_Nucleo\_F303K8\_I↔  
R/Prog/boot.c, [803](#)
  - Demo/ARMCM4\_STM32F3\_Nucleo\_F303K8\_↔  
Keil/Prog/boot.c, [808](#)
  - Demo/ARMCM4\_STM32F4\_Nucleo\_F429ZI\_↔  
CubeIDE/Prog/App/boot.c, [813](#)
  - Demo/ARMCM4\_STM32F4\_Nucleo\_F429ZI\_G↔  
CC/Prog/boot.c, [818](#)
  - Demo/ARMCM4\_STM32F4\_Nucleo\_F429ZI\_I↔  
R/Prog/boot.c, [823](#)
  - Demo/ARMCM4\_STM32F4\_Nucleo\_F429ZI\_↔  
Keil/Prog/boot.c, [828](#)
  - Demo/ARMCM4\_STM32F4\_Olimex\_STM32↔  
P405\_CubeIDE/Prog/App/boot.c, [833](#)
  - Demo/ARMCM4\_STM32F4\_Olimex\_STM32↔  
P405\_GCC/Prog/boot.c, [838](#)
  - Demo/ARMCM4\_STM32F4\_Olimex\_STM32↔  
P405\_IAR/Prog/boot.c, [843](#)
  - Demo/ARMCM4\_STM32F4\_Olimex\_STM32↔  
P405\_Keil/Prog/boot.c, [848](#)
  - Demo/ARMCM4\_STM32L4\_Nucleo\_L476RG\_↔  
CubeIDE/Prog/App/boot.c, [853](#)
  - Demo/ARMCM4\_STM32L4\_Nucleo\_L476RG\_G↔  
CC/Prog/boot.c, [858](#)
  - Demo/ARMCM4\_STM32L4\_Nucleo\_L476RG\_I↔  
AR/Prog/boot.c, [863](#)
  - Demo/ARMCM4\_STM32L4\_Nucleo\_L476RG\_↔

- Keil/Prog/boot.c, [868](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F746ZG\_↔
  - CubeIDE/Prog/App/boot.c, [882](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F746ZG\_G↔
  - CC/Prog/boot.c, [887](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F746ZG\_I↔
  - AR/Prog/boot.c, [892](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F746ZG\_↔
  - Keil/Prog/boot.c, [897](#)
- Demo/ARMCM7\_STM32H7\_Nucleo\_H743ZI\_↔
  - CubeIDE/Prog/App/boot.c, [914](#)
- Demo/ARMCM7\_STM32H7\_Nucleo\_H743ZI\_G↔
  - CC/Prog/boot.c, [919](#)
- Demo/ARMCM7\_STM32H7\_Nucleo\_H743ZI\_IA↔
  - R/Prog/boot.c, [924](#)
- Demo/ARMCM7\_STM32H7\_Nucleo\_H743ZI\_↔
  - Keil/Prog/boot.c, [929](#)
- Demo/HCS12\_Evbplus\_Dragon12p\_Code↔
  - Warrior/Prog/boot.c, [934](#)
- HCS12/can.c, [1117](#)
- CanInit
  - \_template/can.c, [1064](#)
  - ARMCM0\_S32K11/can.c, [1069](#)
  - ARMCM0\_STM32F0/can.c, [1073](#)
  - ARMCM0\_XMC1/can.c, [1076](#)
  - ARMCM33\_STM32L5/can.c, [1079](#)
  - ARMCM3\_LM3S/can.c, [1082](#)
  - ARMCM3\_STM32F1/can.c, [1085](#)
  - ARMCM3\_STM32F2/can.c, [1088](#)
  - ARMCM4\_S32K14/can.c, [1093](#)
  - ARMCM4\_STM32F3/can.c, [1097](#)
  - ARMCM4\_STM32F4/can.c, [1100](#)
  - ARMCM4\_STM32L4/can.c, [1104](#)
  - ARMCM4\_XMC4/can.c, [1107](#)
  - ARMCM7\_STM32F7/can.c, [1110](#)
  - ARMCM7\_STM32H7/can.c, [1113](#)
  - can.h, [1120](#)
  - HCS12/can.c, [1117](#)
- CanReceivePacket
  - \_template/can.c, [1064](#)
  - ARMCM0\_S32K11/can.c, [1069](#)
  - ARMCM0\_STM32F0/can.c, [1073](#)
  - ARMCM0\_XMC1/can.c, [1076](#)
  - ARMCM33\_STM32L5/can.c, [1079](#)
  - ARMCM3\_LM3S/can.c, [1082](#)
  - ARMCM3\_STM32F1/can.c, [1085](#)
  - ARMCM3\_STM32F2/can.c, [1088](#)
  - ARMCM4\_S32K14/can.c, [1093](#)
  - ARMCM4\_STM32F3/can.c, [1097](#)
  - ARMCM4\_STM32F4/can.c, [1100](#)
  - ARMCM4\_STM32L4/can.c, [1104](#)
  - ARMCM4\_XMC4/can.c, [1107](#)
  - ARMCM7\_STM32F7/can.c, [1110](#)
  - ARMCM7\_STM32H7/can.c, [1113](#)
  - can.h, [1120](#)
  - HCS12/can.c, [1117](#)
- CanSetBittiming
  - ARMCM3\_LM3S/can.c, [1082](#)
- Demo/ARMCM3\_LM3S\_EK\_LM3S8962\_GCC/↔
  - Prog/boot.c, [704](#)
- Demo/ARMCM3\_LM3S\_EK\_LM3S8962\_IAR/↔
  - Prog/boot.c, [707](#)
- canTiming
  - \_template/can.c, [1065](#)
  - ARMCM0\_S32K11/can.c, [1070](#)
  - ARMCM0\_STM32F0/can.c, [1074](#)
  - ARMCM33\_STM32L5/can.c, [1080](#)
  - ARMCM3\_STM32F1/can.c, [1086](#)
  - ARMCM3\_STM32F2/can.c, [1089](#)
  - ARMCM4\_S32K14/can.c, [1094](#)
  - ARMCM4\_STM32F3/can.c, [1098](#)
  - ARMCM4\_STM32F4/can.c, [1101](#)
  - ARMCM4\_STM32L4/can.c, [1105](#)
  - ARMCM7\_STM32F7/can.c, [1111](#)
  - ARMCM7\_STM32H7/can.c, [1114](#)
- Demo/ARMCM0\_STM32F0\_Nucleo\_F091RC\_↔
  - CubeIDE/Prog/App/boot.c, [635](#)
- Demo/ARMCM0\_STM32F0\_Nucleo\_F091RC\_G↔
  - CC/Prog/boot.c, [640](#)
- Demo/ARMCM0\_STM32F0\_Nucleo\_F091RC\_I↔
  - AR/Prog/boot.c, [645](#)
- Demo/ARMCM0\_STM32F0\_Nucleo\_F091RC\_↔
  - Keil/Prog/boot.c, [650](#)
- Demo/ARMCM33\_STM32L5\_Nucleo\_L552ZE\_↔
  - CubeIDE/Prog/App/boot.c, [674](#)
- Demo/ARMCM33\_STM32L5\_Nucleo\_L552ZE\_↔
  - GCC/Prog/boot.c, [679](#)
- Demo/ARMCM33\_STM32L5\_Nucleo\_L552ZE\_I↔
  - AR/Prog/boot.c, [684](#)
- Demo/ARMCM33\_STM32L5\_Nucleo\_L552ZE\_↔
  - Keil/Prog/boot.c, [689](#)
- Demo/ARMCM3\_STM32F1\_Olimex\_STM32↔
  - P103\_CubeIDE/Prog/App/boot.c, [725](#)
- Demo/ARMCM3\_STM32F1\_Olimex\_STM32↔
  - P103\_GCC/Prog/boot.c, [730](#)
- Demo/ARMCM3\_STM32F1\_Olimex\_STM32↔
  - P103\_IAR/Prog/boot.c, [735](#)
- Demo/ARMCM3\_STM32F1\_Olimex\_STM32↔
  - P103\_Keil/Prog/boot.c, [740](#)
- Demo/ARMCM3\_STM32F1\_Olimexino\_STM32↔
  - \_CubeIDE/Prog/App/boot.c, [744](#)
- Demo/ARMCM3\_STM32F1\_Olimexino\_STM32↔
  - \_GCC/Prog/boot.c, [747](#)
- Demo/ARMCM3\_STM32F1\_Olimexino\_STM32↔
  - \_IAR/Prog/boot.c, [751](#)
- Demo/ARMCM3\_STM32F1\_Olimexino\_STM32↔
  - \_Keil/Prog/boot.c, [755](#)
- Demo/ARMCM3\_STM32F2\_Olimex\_STM32↔
  - P207\_CubeIDE/Prog/App/boot.c, [760](#)
- Demo/ARMCM3\_STM32F2\_Olimex\_STM32↔
  - P207\_GCC/Prog/boot.c, [765](#)
- Demo/ARMCM3\_STM32F2\_Olimex\_STM32↔
  - P207\_IAR/Prog/boot.c, [770](#)
- Demo/ARMCM3\_STM32F2\_Olimex\_STM32↔
  - P207\_Keil/Prog/boot.c, [775](#)
- Demo/ARMCM4\_S32K14\_S32K144EVB\_GCC/↔

- Prog/boot.c, [782](#)
- Demo/ARMCM4\_S32K14\_S32K144EVB\_IAR/↔
  - Prog/boot.c, [789](#)
- Demo/ARMCM4\_STM32F3\_Nucleo\_F303K8\_↔
  - CubeIDE/Prog/App/boot.c, [794](#)
- Demo/ARMCM4\_STM32F3\_Nucleo\_F303K8\_G↔
  - CC/Prog/boot.c, [799](#)
- Demo/ARMCM4\_STM32F3\_Nucleo\_F303K8\_IA↔
  - R/Prog/boot.c, [804](#)
- Demo/ARMCM4\_STM32F3\_Nucleo\_F303K8\_↔
  - Keil/Prog/boot.c, [809](#)
- Demo/ARMCM4\_STM32F4\_Nucleo\_F429ZI\_↔
  - CubeIDE/Prog/App/boot.c, [814](#)
- Demo/ARMCM4\_STM32F4\_Nucleo\_F429ZI\_G↔
  - CC/Prog/boot.c, [819](#)
- Demo/ARMCM4\_STM32F4\_Nucleo\_F429ZI\_IA↔
  - R/Prog/boot.c, [824](#)
- Demo/ARMCM4\_STM32F4\_Nucleo\_F429ZI\_↔
  - Keil/Prog/boot.c, [829](#)
- Demo/ARMCM4\_STM32F4\_Olimex\_STM32↔
  - P405\_CubeIDE/Prog/App/boot.c, [834](#)
- Demo/ARMCM4\_STM32F4\_Olimex\_STM32↔
  - P405\_GCC/Prog/boot.c, [839](#)
- Demo/ARMCM4\_STM32F4\_Olimex\_STM32↔
  - P405\_IAR/Prog/boot.c, [844](#)
- Demo/ARMCM4\_STM32F4\_Olimex\_STM32↔
  - P405\_Keil/Prog/boot.c, [849](#)
- Demo/ARMCM4\_STM32L4\_Nucleo\_L476RG\_↔
  - CubeIDE/Prog/App/boot.c, [854](#)
- Demo/ARMCM4\_STM32L4\_Nucleo\_L476RG\_G↔
  - CC/Prog/boot.c, [859](#)
- Demo/ARMCM4\_STM32L4\_Nucleo\_L476RG\_I↔
  - AR/Prog/boot.c, [864](#)
- Demo/ARMCM4\_STM32L4\_Nucleo\_L476RG\_↔
  - Keil/Prog/boot.c, [869](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F746ZG\_↔
  - CubeIDE/Prog/App/boot.c, [883](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F746ZG\_G↔
  - CC/Prog/boot.c, [888](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F746ZG\_I↔
  - AR/Prog/boot.c, [893](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F746ZG\_↔
  - Keil/Prog/boot.c, [898](#)
- Demo/ARMCM7\_STM32H7\_Nucleo\_H743ZI\_↔
  - CubeIDE/Prog/App/boot.c, [915](#)
- Demo/ARMCM7\_STM32H7\_Nucleo\_H743ZI\_G↔
  - CC/Prog/boot.c, [920](#)
- Demo/ARMCM7\_STM32H7\_Nucleo\_H743ZI\_IA↔
  - R/Prog/boot.c, [925](#)
- Demo/ARMCM7\_STM32H7\_Nucleo\_H743ZI\_↔
  - Keil/Prog/boot.c, [930](#)
- Demo/HCS12\_Evbplus\_Dragon12p\_Code↔
  - Warrior/Prog/boot.c, [935](#)
- HCS12/can.c, [1118](#)
- CanTransmitPacket
  - \_template/can.c, [1065](#)
  - ARMCM0\_S32K11/can.c, [1070](#)
  - ARMCM0\_STM32F0/can.c, [1073](#)
  - ARMCM0\_XMC1/can.c, [1077](#)
  - ARMCM33\_STM32L5/can.c, [1079](#)
  - ARMCM3\_LM3S/can.c, [1082](#)
  - ARMCM3\_STM32F1/can.c, [1085](#)
  - ARMCM3\_STM32F2/can.c, [1089](#)
  - ARMCM4\_S32K14/can.c, [1094](#)
  - ARMCM4\_STM32F3/can.c, [1097](#)
  - ARMCM4\_STM32F4/can.c, [1101](#)
  - ARMCM4\_STM32L4/can.c, [1104](#)
  - ARMCM4\_XMC4/can.c, [1107](#)
  - ARMCM7\_STM32F7/can.c, [1110](#)
  - ARMCM7\_STM32H7/can.c, [1114](#)
  - can.h, [1120](#)
  - HCS12/can.c, [1118](#)
- canUse
  - ARMCM3\_LM3S\_EK\_LM3S6965\_GCC/Boot/hooks.↔
    - c, [1736](#)
  - ARMCM3\_LM3S\_EK\_LM3S6965\_IAR/Boot/hooks.↔
    - c, [1744](#)
  - ARMCM3\_STM32F1\_Olimex\_STM32P103\_↔
    - CubeIDE/Boot/App/hooks.c, [1806](#)
  - ARMCM3\_STM32F1\_Olimex\_STM32P103\_GC↔
    - C/Boot/hooks.c, [1813](#)
  - ARMCM3\_STM32F1\_Olimex\_STM32P103\_IA↔
    - R/Boot/hooks.c, [1821](#)
  - ARMCM3\_STM32F1\_Olimex\_STM32P103\_Keil/↔
    - Boot/hooks.c, [1829](#)
  - ARMCM3\_STM32F1\_Olimexino\_STM32\_Cube↔
    - IDE/Boot/App/hooks.c, [1837](#)
  - ARMCM3\_STM32F1\_Olimexino\_STM32\_GCC/↔
    - Boot/hooks.c, [1846](#)
  - ARMCM3\_STM32F1\_Olimexino\_STM32\_IAR/↔
    - Boot/hooks.c, [1855](#)
  - ARMCM3\_STM32F1\_Olimexino\_STM32\_Keil/↔
    - Boot/hooks.c, [1864](#)
  - ARMCM3\_STM32F2\_Olimex\_STM32P207\_↔
    - CubeIDE/Boot/App/hooks.c, [1872](#)
  - ARMCM3\_STM32F2\_Olimex\_STM32P207\_GC↔
    - C/Boot/hooks.c, [1879](#)
  - ARMCM3\_STM32F2\_Olimex\_STM32P207\_IA↔
    - R/Boot/hooks.c, [1887](#)
  - ARMCM3\_STM32F2\_Olimex\_STM32P207\_Keil/↔
    - Boot/hooks.c, [1895](#)
  - ARMCM4\_STM32F4\_Olimex\_STM32P405\_↔
    - CubeIDE/Boot/App/hooks.c, [1981](#)
  - ARMCM4\_STM32F4\_Olimex\_STM32P405\_GC↔
    - C/Boot/hooks.c, [1990](#)
  - ARMCM4\_STM32F4\_Olimex\_STM32P405\_IA↔
    - R/Boot/hooks.c, [1999](#)
  - ARMCM4\_STM32F4\_Olimex\_STM32P405\_Keil/↔
    - Boot/hooks.c, [2008](#)
  - ARMCM4\_TM4C\_DK\_TM4C123G\_IAR/Boot/hooks.↔
    - c, [2037](#)
  - ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_GCC/↔
    - Boot/hooks.c, [2045](#)
  - ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_IAR/↔
    - Boot/hooks.c, [2053](#)
- cbtr0

- tCanRegs, [359](#)
- cbtr1
  - tCanRegs, [359](#)
- cctl0
  - tCanRegs, [359](#)
- cctl1
  - tCanRegs, [359](#)
- cforc
  - tTimerRegs, [381](#)
- cidac
  - tCanRegs, [359](#)
- cidar0
  - tCanRegs, [359](#)
- cidar1
  - tCanRegs, [359](#)
- cidar2
  - tCanRegs, [360](#)
- cidar3
  - tCanRegs, [360](#)
- cidar4
  - tCanRegs, [360](#)
- cidar5
  - tCanRegs, [360](#)
- cidar6
  - tCanRegs, [360](#)
- cidar7
  - tCanRegs, [360](#)
- cidmr0
  - tCanRegs, [360](#)
- cidmr1
  - tCanRegs, [360](#)
- cidmr2
  - tCanRegs, [361](#)
- cidmr3
  - tCanRegs, [361](#)
- cidmr4
  - tCanRegs, [361](#)
- cidmr5
  - tCanRegs, [361](#)
- cidmr6
  - tCanRegs, [361](#)
- cidmr7
  - tCanRegs, [361](#)
- com.c, [1121](#)
  - ComFree, [1122](#)
  - ComGetActiveInterfaceMaxRxLen, [1122](#)
  - ComGetActiveInterfaceMaxTxLen, [1122](#)
  - ComInit, [1122](#)
  - ComIsConnected, [1123](#)
  - ComTask, [1123](#)
  - ComTransmitPacket, [1123](#)
- com.h, [1124](#)
  - ComFree, [1126](#)
  - ComGetActiveInterfaceMaxRxLen, [1126](#)
  - ComGetActiveInterfaceMaxTxLen, [1126](#)
  - ComInit, [1126](#)
  - ComIsConnected, [1127](#)
  - ComTask, [1127](#)
  - ComTransmitPacket, [1127](#)
  - tComInterfaceld, [1125](#)
- ComFree
  - com.c, [1122](#)
  - com.h, [1126](#)
- ComGetActiveInterfaceMaxRxLen
  - com.c, [1122](#)
  - com.h, [1126](#)
- ComGetActiveInterfaceMaxTxLen
  - com.c, [1122](#)
  - com.h, [1126](#)
- ComInit
  - com.c, [1122](#)
  - com.h, [1126](#)
- ComIsConnected
  - com.c, [1123](#)
  - com.h, [1127](#)
- ComTask
  - com.c, [1123](#)
  - com.h, [1127](#)
- ComTransmitPacket
  - com.c, [1123](#)
  - com.h, [1127](#)
- Compiler specifics of a port, [326](#)
- connected
  - tXcplInfo, [384](#)
- cop.c, [1128](#)
  - CopInit, [1129](#)
  - CopInitHook, [1129](#)
  - CopService, [1129](#)
  - CopServiceHook, [1129](#)
- cop.h, [1130](#)
  - CopInit, [1130](#)
  - CopService, [1130](#)
- CopInit
  - cop.c, [1129](#)
  - cop.h, [1130](#)
- CopInitHook
  - \_template/Boot/hooks.c, [1621](#)
  - ARMCM0\_STM32F0\_Discovery\_STM32F051\_↔  
CubeIDE/Boot/App/hooks.c, [1626](#)
  - ARMCM0\_STM32F0\_Discovery\_STM32F051\_↔  
GCC/Boot/hooks.c, [1631](#)
  - ARMCM0\_STM32F0\_Discovery\_STM32F051\_I↔  
AR/Boot/hooks.c, [1636](#)
  - ARMCM0\_STM32F0\_Discovery\_STM32F051\_↔  
Keil/Boot/hooks.c, [1641](#)
  - ARMCM0\_STM32F0\_Nucleo\_F091RC\_CubeID↔  
E/Boot/App/hooks.c, [1646](#)
  - ARMCM0\_STM32F0\_Nucleo\_F091RC\_GCC/↔  
Boot/hooks.c, [1651](#)
  - ARMCM0\_STM32F0\_Nucleo\_F091RC\_IAR/↔  
Boot/hooks.c, [1656](#)
  - ARMCM0\_STM32F0\_Nucleo\_F091RC\_Keil/↔  
Boot/hooks.c, [1661](#)
  - ARMCM0\_STM32G0\_Nucleo\_G071RB\_CubeID↔  
E/Boot/App/hooks.c, [1666](#)



- ARMCM0\_STM32G0\_Nucleo\_G071RB\_GCC/↔  
Boot/hooks.c, [1671](#)
- ARMCM0\_STM32G0\_Nucleo\_G071RB\_IAR/↔  
Boot/hooks.c, [1676](#)
- ARMCM0\_STM32G0\_Nucleo\_G071RB\_Keil/↔  
Boot/hooks.c, [1681](#)
- ARMCM0\_XMC1\_XMC1400\_Boot\_Kit\_GCC/↔  
Boot/hooks.c, [1686](#)
- ARMCM0\_XMC1\_XMC1400\_Boot\_Kit\_IAR/↔  
Boot/hooks.c, [1691](#)
- ARMCM33\_STM32L5\_Nucleo\_L552ZE\_CubeID↔  
E/Boot/App/hooks.c, [1696](#)
- ARMCM33\_STM32L5\_Nucleo\_L552ZE\_GCC/↔  
Boot/hooks.c, [1702](#)
- ARMCM33\_STM32L5\_Nucleo\_L552ZE\_IAR/↔  
Boot/hooks.c, [1708](#)
- ARMCM33\_STM32L5\_Nucleo\_L552ZE\_Keil/↔  
Boot/hooks.c, [1714](#)
- ARMCM3\_EFM32\_Olimex\_EM32G880F128ST↔  
K\_GCC/Boot/hooks.c, [1720](#)
- ARMCM3\_EFM32\_Olimex\_EM32G880F128ST↔  
K\_IAR/Boot/hooks.c, [1725](#)
- ARMCM3\_LM3S\_EK\_LM3S6965\_GCC/Boot/hooks.↔  
c, [1731](#)
- ARMCM3\_LM3S\_EK\_LM3S6965\_IAR/Boot/hooks.↔  
c, [1739](#)
- ARMCM3\_LM3S\_EK\_LM3S8962\_GCC/Boot/hooks.↔  
c, [1746](#)
- ARMCM3\_LM3S\_EK\_LM3S8962\_IAR/Boot/hooks.↔  
c, [1751](#)
- ARMCM3\_STM32F1\_Nucleo\_F103RB\_CubeID↔  
E/Boot/App/hooks.c, [1756](#)
- ARMCM3\_STM32F1\_Nucleo\_F103RB\_GCC/↔  
Boot/hooks.c, [1761](#)
- ARMCM3\_STM32F1\_Nucleo\_F103RB\_IAR/↔  
Boot/hooks.c, [1766](#)
- ARMCM3\_STM32F1\_Nucleo\_F103RB\_Keil/↔  
Boot/hooks.c, [1771](#)
- ARMCM3\_STM32F1\_Olimex\_STM32H103\_↔  
CubeIDE/Boot/App/hooks.c, [1776](#)
- ARMCM3\_STM32F1\_Olimex\_STM32H103\_GC↔  
C/Boot/hooks.c, [1782](#)
- ARMCM3\_STM32F1\_Olimex\_STM32H103\_IA↔  
R/Boot/hooks.c, [1788](#)
- ARMCM3\_STM32F1\_Olimex\_STM32H103\_↔  
Keil/Boot/hooks.c, [1794](#)
- ARMCM3\_STM32F1\_Olimex\_STM32P103\_↔  
CubeIDE/Boot/App/hooks.c, [1801](#)
- ARMCM3\_STM32F1\_Olimex\_STM32P103\_GC↔  
C/Boot/hooks.c, [1808](#)
- ARMCM3\_STM32F1\_Olimex\_STM32P103\_IA↔  
R/Boot/hooks.c, [1816](#)
- ARMCM3\_STM32F1\_Olimex\_STM32P103\_Keil/↔  
Boot/hooks.c, [1824](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_Cube↔  
IDE/Boot/App/hooks.c, [1831](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_GCC/↔  
Boot/hooks.c, [1840](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_IAR/↔  
Boot/hooks.c, [1849](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_Keil/↔  
Boot/hooks.c, [1858](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_↔  
CubeIDE/Boot/App/hooks.c, [1867](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_GC↔  
C/Boot/hooks.c, [1874](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_IA↔  
R/Boot/hooks.c, [1882](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_Keil/↔  
Boot/hooks.c, [1890](#)
- ARMCM4\_S32K14\_S32K144EVB\_GCC/Boot/hooks.↔  
c, [1897](#)
- ARMCM4\_S32K14\_S32K144EVB\_IAR/Boot/hooks.↔  
c, [1902](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_Cube↔  
IDE/Boot/App/hooks.c, [1907](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_GCC/↔  
Boot/hooks.c, [1913](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_IAR/↔  
Boot/hooks.c, [1919](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_Keil/↔  
Boot/hooks.c, [1925](#)
- ARMCM4\_STM32F3\_Nucleo\_F303K8\_CubeID↔  
E/Boot/App/hooks.c, [1931](#)
- ARMCM4\_STM32F3\_Nucleo\_F303K8\_GCC/↔  
Boot/hooks.c, [1936](#)
- ARMCM4\_STM32F3\_Nucleo\_F303K8\_IAR/↔  
Boot/hooks.c, [1941](#)
- ARMCM4\_STM32F3\_Nucleo\_F303K8\_Keil/↔  
Boot/hooks.c, [1946](#)
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_CubeID↔  
E/Boot/App/hooks.c, [1951](#)
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_GCC/↔  
Boot/hooks.c, [1957](#)
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_IAR/↔  
Boot/hooks.c, [1963](#)
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_Keil/↔  
Boot/hooks.c, [1969](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_↔  
CubeIDE/Boot/App/hooks.c, [1976](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_GC↔  
C/Boot/hooks.c, [1984](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_IA↔  
R/Boot/hooks.c, [1993](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_Keil/↔  
Boot/hooks.c, [2002](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_CubeID↔  
E/Boot/App/hooks.c, [2010](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_GCC/↔  
Boot/hooks.c, [2015](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_IAR/↔  
Boot/hooks.c, [2020](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_Keil/↔  
Boot/hooks.c, [2025](#)
- ARMCM4\_TM4C\_DK\_TM4C123G\_IAR/Boot/hooks.↔  
c, [2031](#)

- ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_GCC/↔  
Boot/hooks.c, [2040](#)
- ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_IAR/↔  
Boot/hooks.c, [2047](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_CubeID↔  
E/Boot/App/hooks.c, [2055](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_GCC/↔  
Boot/hooks.c, [2061](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_IAR/↔  
Boot/hooks.c, [2067](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_Keil/↔  
Boot/hooks.c, [2073](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_CubeID↔  
E/Boot/App/hooks.c, [2079](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_GCC/↔  
Boot/hooks.c, [2084](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_IAR/↔  
Boot/hooks.c, [2089](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_Keil/↔  
Boot/hooks.c, [2094](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_CubeID↔  
E/Boot/App/hooks.c, [2099](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_GCC/↔  
Boot/hooks.c, [2105](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_IAR/↔  
Boot/hooks.c, [2111](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_Keil/↔  
Boot/hooks.c, [2117](#)
- cop.c, [1129](#)
- HCS12\_DevKit\_S12G128\_CodeWarrior/Boot/hooks.↔  
c, [2123](#)
- HCS12\_Evbplus\_Dragon12p\_CodeWarrior/↔  
Boot/hooks.c, [2128](#)
- CopService
  - cop.c, [1129](#)
  - cop.h, [1130](#)
- CopServiceHook
  - \_template/Boot/hooks.c, [1622](#)
  - ARMCM0\_STM32F0\_Discovery\_STM32F051\_↔  
CubeIDE/Boot/App/hooks.c, [1627](#)
  - ARMCM0\_STM32F0\_Discovery\_STM32F051\_↔  
GCC/Boot/hooks.c, [1632](#)
  - ARMCM0\_STM32F0\_Discovery\_STM32F051\_I↔  
AR/Boot/hooks.c, [1637](#)
  - ARMCM0\_STM32F0\_Discovery\_STM32F051\_↔  
Keil/Boot/hooks.c, [1642](#)
  - ARMCM0\_STM32F0\_Nucleo\_F091RC\_CubeID↔  
E/Boot/App/hooks.c, [1647](#)
  - ARMCM0\_STM32F0\_Nucleo\_F091RC\_GCC/↔  
Boot/hooks.c, [1652](#)
  - ARMCM0\_STM32F0\_Nucleo\_F091RC\_IAR/↔  
Boot/hooks.c, [1657](#)
  - ARMCM0\_STM32F0\_Nucleo\_F091RC\_Keil/↔  
Boot/hooks.c, [1662](#)
  - ARMCM0\_STM32G0\_Nucleo\_G071RB\_CubeID↔  
E/Boot/App/hooks.c, [1667](#)
  - ARMCM0\_STM32G0\_Nucleo\_G071RB\_GCC/↔  
Boot/hooks.c, [1672](#)
  - ARMCM0\_STM32G0\_Nucleo\_G071RB\_IAR/↔  
Boot/hooks.c, [1677](#)
  - ARMCM0\_STM32G0\_Nucleo\_G071RB\_Keil/↔  
Boot/hooks.c, [1682](#)
  - ARMCM0\_XMC1\_XMC1400\_Boot\_Kit\_GCC/↔  
Boot/hooks.c, [1687](#)
  - ARMCM0\_XMC1\_XMC1400\_Boot\_Kit\_IAR/↔  
Boot/hooks.c, [1692](#)
  - ARMCM33\_STM32L5\_Nucleo\_L552ZE\_CubeID↔  
E/Boot/App/hooks.c, [1697](#)
  - ARMCM33\_STM32L5\_Nucleo\_L552ZE\_GCC/↔  
Boot/hooks.c, [1703](#)
  - ARMCM33\_STM32L5\_Nucleo\_L552ZE\_IAR/↔  
Boot/hooks.c, [1709](#)
  - ARMCM33\_STM32L5\_Nucleo\_L552ZE\_Keil/↔  
Boot/hooks.c, [1715](#)
  - ARMCM3\_EFM32\_Olimex\_EM32G880F128ST↔  
K\_GCC/Boot/hooks.c, [1720](#)
  - ARMCM3\_EFM32\_Olimex\_EM32G880F128ST↔  
K\_IAR/Boot/hooks.c, [1726](#)
  - ARMCM3\_LM3S\_EK\_LM3S6965\_GCC/Boot/hooks.↔  
c, [1731](#)
  - ARMCM3\_LM3S\_EK\_LM3S6965\_IAR/Boot/hooks.↔  
c, [1739](#)
  - ARMCM3\_LM3S\_EK\_LM3S8962\_GCC/Boot/hooks.↔  
c, [1746](#)
  - ARMCM3\_LM3S\_EK\_LM3S8962\_IAR/Boot/hooks.↔  
c, [1751](#)
  - ARMCM3\_STM32F1\_Nucleo\_F103RB\_CubeID↔  
E/Boot/App/hooks.c, [1757](#)
  - ARMCM3\_STM32F1\_Nucleo\_F103RB\_GCC/↔  
Boot/hooks.c, [1762](#)
  - ARMCM3\_STM32F1\_Nucleo\_F103RB\_IAR/↔  
Boot/hooks.c, [1767](#)
  - ARMCM3\_STM32F1\_Nucleo\_F103RB\_Keil/↔  
Boot/hooks.c, [1772](#)
  - ARMCM3\_STM32F1\_Olimex\_STM32H103\_↔  
CubeIDE/Boot/App/hooks.c, [1777](#)
  - ARMCM3\_STM32F1\_Olimex\_STM32H103\_GC↔  
C/Boot/hooks.c, [1782](#)
  - ARMCM3\_STM32F1\_Olimex\_STM32H103\_IA↔  
R/Boot/hooks.c, [1788](#)
  - ARMCM3\_STM32F1\_Olimex\_STM32H103\_↔  
Keil/Boot/hooks.c, [1794](#)
  - ARMCM3\_STM32F1\_Olimex\_STM32P103\_↔  
CubeIDE/Boot/App/hooks.c, [1801](#)
  - ARMCM3\_STM32F1\_Olimex\_STM32P103\_GC↔  
C/Boot/hooks.c, [1809](#)
  - ARMCM3\_STM32F1\_Olimex\_STM32P103\_IA↔  
R/Boot/hooks.c, [1816](#)
  - ARMCM3\_STM32F1\_Olimex\_STM32P103\_Keil/↔  
Boot/hooks.c, [1824](#)
  - ARMCM3\_STM32F1\_Olimexino\_STM32\_Cube↔  
DE/Boot/App/hooks.c, [1832](#)
  - ARMCM3\_STM32F1\_Olimexino\_STM32\_GCC/↔  
Boot/hooks.c, [1840](#)
  - ARMCM3\_STM32F1\_Olimexino\_STM32\_IAR/↔  
Boot/hooks.c, [1849](#)

- ARMCM3\_STM32F1\_Olimexino\_STM32\_Keil/↔  
Boot/hooks.c, [1858](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_↔  
CubeIDE/Boot/App/hooks.c, [1867](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_GC↔  
C/Boot/hooks.c, [1875](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_IA↔  
R/Boot/hooks.c, [1882](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_Keil/↔  
Boot/hooks.c, [1890](#)
- ARMCM4\_S32K14\_S32K144EVB\_GCC/Boot/hooks.↔  
c, [1897](#)
- ARMCM4\_S32K14\_S32K144EVB\_IAR/Boot/hooks.↔  
c, [1902](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_Cube↔  
IDE/Boot/App/hooks.c, [1908](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_GCC/↔  
Boot/hooks.c, [1913](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_IAR/↔  
Boot/hooks.c, [1919](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_Keil/↔  
Boot/hooks.c, [1925](#)
- ARMCM4\_STM32F3\_Nucleo\_F303K8\_CubeID↔  
E/Boot/App/hooks.c, [1931](#)
- ARMCM4\_STM32F3\_Nucleo\_F303K8\_GCC/↔  
Boot/hooks.c, [1937](#)
- ARMCM4\_STM32F3\_Nucleo\_F303K8\_IAR/↔  
Boot/hooks.c, [1942](#)
- ARMCM4\_STM32F3\_Nucleo\_F303K8\_Keil/↔  
Boot/hooks.c, [1947](#)
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_CubeID↔  
E/Boot/App/hooks.c, [1952](#)
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_GCC/↔  
Boot/hooks.c, [1958](#)
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_IAR/↔  
Boot/hooks.c, [1964](#)
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_Keil/↔  
Boot/hooks.c, [1970](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_↔  
CubeIDE/Boot/App/hooks.c, [1976](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_GC↔  
C/Boot/hooks.c, [1984](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_IA↔  
R/Boot/hooks.c, [1993](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_Keil/↔  
Boot/hooks.c, [2002](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_CubeID↔  
E/Boot/App/hooks.c, [2011](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_GCC/↔  
Boot/hooks.c, [2016](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_IAR/↔  
Boot/hooks.c, [2021](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_Keil/↔  
Boot/hooks.c, [2026](#)
- ARMCM4\_TM4C\_DK\_TM4C123G\_IAR/Boot/hooks.↔  
c, [2031](#)
- ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_GCC/↔  
Boot/hooks.c, [2040](#)
- ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_IAR/↔  
Boot/hooks.c, [2048](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_CubeID↔  
E/Boot/App/hooks.c, [2055](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_GCC/↔  
Boot/hooks.c, [2061](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_IAR/↔  
Boot/hooks.c, [2067](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_Keil/↔  
Boot/hooks.c, [2073](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_CubeID↔  
E/Boot/App/hooks.c, [2079](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_GCC/↔  
Boot/hooks.c, [2085](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_IAR/↔  
Boot/hooks.c, [2090](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_Keil/↔  
Boot/hooks.c, [2095](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_CubeID↔  
E/Boot/App/hooks.c, [2100](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_GCC/↔  
Boot/hooks.c, [2105](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_IAR/↔  
Boot/hooks.c, [2111](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_Keil/↔  
Boot/hooks.c, [2117](#)
- cop.c, [1129](#)
- HCS12\_DevKit\_S12G128\_CodeWarrior/Boot/hooks.↔  
c, [2123](#)
- HCS12\_Evbplus\_Dragon12p\_CodeWarrior/↔  
Boot/hooks.c, [2129](#)
- cpu.c, [1131](#), [1134](#), [1137](#), [1140](#), [1143](#), [1146](#), [1149](#), [1152](#),  
[1155](#), [1158](#), [1161](#), [1164](#), [1167](#), [1170](#), [1173](#),  
[1176](#), [1179](#), [1182](#), [1185](#)
- cpu.h, [1188](#)
- CpuInit, [1188](#)
- CpuIrqDisable, [1188](#)
- CpuIrqEnable, [1189](#)
- CpuMemCopy, [1189](#)
- CpuMemSet, [1190](#)
- CpuStartUserProgram, [1190](#)
- cpu\_comp.c, [1190](#), [1192–1194](#), [1196–1199](#), [1201–](#)  
[1204](#), [1206–1209](#), [1211–1214](#), [1216–1219](#),  
[1221–1224](#), [1226–1229](#), [1231–1234](#), [1236–](#)  
[1239](#), [1241–1244](#), [1246](#)
- CpuInit  
\_template/cpu.c, [1132](#)
- ARMCM0\_S32K11/cpu.c, [1135](#)
- ARMCM0\_STM32F0/cpu.c, [1138](#)
- ARMCM0\_STM32G0/cpu.c, [1141](#)
- ARMCM0\_XMC1/cpu.c, [1144](#)
- ARMCM33\_STM32L5/cpu.c, [1147](#)
- ARMCM3\_EFM32/cpu.c, [1150](#)
- ARMCM3\_LM3S/cpu.c, [1153](#)
- ARMCM3\_STM32F1/cpu.c, [1156](#)
- ARMCM3\_STM32F2/cpu.c, [1159](#)
- ARMCM4\_S32K14/cpu.c, [1162](#)
- ARMCM4\_STM32F3/cpu.c, [1165](#)

- ARMCM4\_STM32F4/cpu.c, 1168
- ARMCM4\_STM32L4/cpu.c, 1171
- ARMCM4\_TM4C/cpu.c, 1174
- ARMCM4\_XMC4/cpu.c, 1177
- ARMCM7\_STM32F7/cpu.c, 1180
- ARMCM7\_STM32H7/cpu.c, 1183
- cpu.h, 1188
- HCS12/cpu.c, 1186
- CpuIrqDisable
  - \_template/GCC/cpu\_comp.c, 1191
  - ARMCM0\_S32K11/GCC/cpu\_comp.c, 1192
  - ARMCM0\_S32K11/IAR/cpu\_comp.c, 1194
  - ARMCM0\_STM32F0/GCC/cpu\_comp.c, 1195
  - ARMCM0\_STM32F0/IAR/cpu\_comp.c, 1196
  - ARMCM0\_STM32F0/Keil/cpu\_comp.c, 1197
  - ARMCM0\_STM32G0/GCC/cpu\_comp.c, 1199
  - ARMCM0\_STM32G0/IAR/cpu\_comp.c, 1200
  - ARMCM0\_STM32G0/Keil/cpu\_comp.c, 1201
  - ARMCM0\_XMC1/GCC/cpu\_comp.c, 1202
  - ARMCM0\_XMC1/IAR/cpu\_comp.c, 1204
  - ARMCM33\_STM32L5/GCC/cpu\_comp.c, 1205
  - ARMCM33\_STM32L5/IAR/cpu\_comp.c, 1206
  - ARMCM33\_STM32L5/Keil/cpu\_comp.c, 1207
  - ARMCM3\_EFM32/GCC/cpu\_comp.c, 1209
  - ARMCM3\_EFM32/IAR/cpu\_comp.c, 1210
  - ARMCM3\_LM3S/GCC/cpu\_comp.c, 1211
  - ARMCM3\_LM3S/IAR/cpu\_comp.c, 1212
  - ARMCM3\_STM32F1/GCC/cpu\_comp.c, 1214
  - ARMCM3\_STM32F1/IAR/cpu\_comp.c, 1215
  - ARMCM3\_STM32F1/Keil/cpu\_comp.c, 1216
  - ARMCM3\_STM32F2/GCC/cpu\_comp.c, 1217
  - ARMCM3\_STM32F2/IAR/cpu\_comp.c, 1219
  - ARMCM3\_STM32F2/Keil/cpu\_comp.c, 1220
  - ARMCM4\_S32K14/GCC/cpu\_comp.c, 1221
  - ARMCM4\_S32K14/IAR/cpu\_comp.c, 1222
  - ARMCM4\_STM32F3/GCC/cpu\_comp.c, 1224
  - ARMCM4\_STM32F3/IAR/cpu\_comp.c, 1225
  - ARMCM4\_STM32F3/Keil/cpu\_comp.c, 1226
  - ARMCM4\_STM32F4/GCC/cpu\_comp.c, 1227
  - ARMCM4\_STM32F4/IAR/cpu\_comp.c, 1229
  - ARMCM4\_STM32F4/Keil/cpu\_comp.c, 1230
  - ARMCM4\_STM32L4/GCC/cpu\_comp.c, 1231
  - ARMCM4\_STM32L4/IAR/cpu\_comp.c, 1232
  - ARMCM4\_STM32L4/Keil/cpu\_comp.c, 1234
  - ARMCM4\_TM4C/IAR/cpu\_comp.c, 1235
  - ARMCM4\_XMC4/GCC/cpu\_comp.c, 1236
  - ARMCM4\_XMC4/IAR/cpu\_comp.c, 1237
  - ARMCM7\_STM32F7/GCC/cpu\_comp.c, 1239
  - ARMCM7\_STM32F7/IAR/cpu\_comp.c, 1240
  - ARMCM7\_STM32F7/Keil/cpu\_comp.c, 1241
  - ARMCM7\_STM32H7/GCC/cpu\_comp.c, 1242
  - ARMCM7\_STM32H7/IAR/cpu\_comp.c, 1244
  - ARMCM7\_STM32H7/Keil/cpu\_comp.c, 1245
  - cpu.h, 1188
  - HCS12/CodeWarrior/cpu\_comp.c, 1246
- CpuIrqEnable
  - \_template/GCC/cpu\_comp.c, 1191
  - ARMCM0\_S32K11/GCC/cpu\_comp.c, 1193
- ARMCM0\_S32K11/IAR/cpu\_comp.c, 1194
- ARMCM0\_STM32F0/GCC/cpu\_comp.c, 1195
- ARMCM0\_STM32F0/IAR/cpu\_comp.c, 1196
- ARMCM0\_STM32F0/Keil/cpu\_comp.c, 1198
- ARMCM0\_STM32G0/GCC/cpu\_comp.c, 1199
- ARMCM0\_STM32G0/IAR/cpu\_comp.c, 1200
- ARMCM0\_STM32G0/Keil/cpu\_comp.c, 1201
- ARMCM0\_XMC1/GCC/cpu\_comp.c, 1203
- ARMCM0\_XMC1/IAR/cpu\_comp.c, 1204
- ARMCM33\_STM32L5/GCC/cpu\_comp.c, 1205
- ARMCM33\_STM32L5/IAR/cpu\_comp.c, 1206
- ARMCM33\_STM32L5/Keil/cpu\_comp.c, 1208
- ARMCM3\_EFM32/GCC/cpu\_comp.c, 1209
- ARMCM3\_EFM32/IAR/cpu\_comp.c, 1210
- ARMCM3\_LM3S/GCC/cpu\_comp.c, 1211
- ARMCM3\_LM3S/IAR/cpu\_comp.c, 1213
- ARMCM3\_STM32F1/GCC/cpu\_comp.c, 1214
- ARMCM3\_STM32F1/IAR/cpu\_comp.c, 1215
- ARMCM3\_STM32F1/Keil/cpu\_comp.c, 1216
- ARMCM3\_STM32F2/GCC/cpu\_comp.c, 1218
- ARMCM3\_STM32F2/IAR/cpu\_comp.c, 1219
- ARMCM3\_STM32F2/Keil/cpu\_comp.c, 1220
- ARMCM4\_S32K14/GCC/cpu\_comp.c, 1221
- ARMCM4\_S32K14/IAR/cpu\_comp.c, 1223
- ARMCM4\_STM32F3/GCC/cpu\_comp.c, 1224
- ARMCM4\_STM32F3/IAR/cpu\_comp.c, 1225
- ARMCM4\_STM32F3/Keil/cpu\_comp.c, 1226
- ARMCM4\_STM32F4/GCC/cpu\_comp.c, 1228
- ARMCM4\_STM32F4/IAR/cpu\_comp.c, 1229
- ARMCM4\_STM32F4/Keil/cpu\_comp.c, 1230
- ARMCM4\_STM32L4/GCC/cpu\_comp.c, 1231
- ARMCM4\_STM32L4/IAR/cpu\_comp.c, 1233
- ARMCM4\_STM32L4/Keil/cpu\_comp.c, 1234
- ARMCM4\_TM4C/IAR/cpu\_comp.c, 1235
- ARMCM4\_XMC4/GCC/cpu\_comp.c, 1236
- ARMCM4\_XMC4/IAR/cpu\_comp.c, 1238
- ARMCM7\_STM32F7/GCC/cpu\_comp.c, 1239
- ARMCM7\_STM32F7/IAR/cpu\_comp.c, 1240
- ARMCM7\_STM32F7/Keil/cpu\_comp.c, 1241
- ARMCM7\_STM32H7/GCC/cpu\_comp.c, 1243
- ARMCM7\_STM32H7/IAR/cpu\_comp.c, 1244
- ARMCM7\_STM32H7/Keil/cpu\_comp.c, 1245
- cpu.h, 1189
- HCS12/CodeWarrior/cpu\_comp.c, 1246
- CpuMemCopy
  - \_template/cpu.c, 1132
  - ARMCM0\_S32K11/cpu.c, 1135
  - ARMCM0\_STM32F0/cpu.c, 1138
  - ARMCM0\_STM32G0/cpu.c, 1141
  - ARMCM0\_XMC1/cpu.c, 1144
  - ARMCM33\_STM32L5/cpu.c, 1147
  - ARMCM3\_EFM32/cpu.c, 1150
  - ARMCM3\_LM3S/cpu.c, 1153
  - ARMCM3\_STM32F1/cpu.c, 1156
  - ARMCM3\_STM32F2/cpu.c, 1159
  - ARMCM4\_S32K14/cpu.c, 1162
  - ARMCM4\_STM32F3/cpu.c, 1165
  - ARMCM4\_STM32F4/cpu.c, 1168

- ARMCM4\_STM32L4/cpu.c, [1171](#)
- ARMCM4\_TM4C/cpu.c, [1174](#)
- ARMCM4\_XMC4/cpu.c, [1177](#)
- ARMCM7\_STM32F7/cpu.c, [1180](#)
- ARMCM7\_STM32H7/cpu.c, [1183](#)
- cpu.h, [1189](#)
- HCS12/cpu.c, [1186](#)
- CpuMemSet
  - \_template/cpu.c, [1132](#)
  - ARMCM0\_S32K11/cpu.c, [1135](#)
  - ARMCM0\_STM32F0/cpu.c, [1138](#)
  - ARMCM0\_STM32G0/cpu.c, [1141](#)
  - ARMCM0\_XMC1/cpu.c, [1144](#)
  - ARMCM33\_STM32L5/cpu.c, [1147](#)
  - ARMCM3\_EFM32/cpu.c, [1150](#)
  - ARMCM3\_LM3S/cpu.c, [1153](#)
  - ARMCM3\_STM32F1/cpu.c, [1156](#)
  - ARMCM3\_STM32F2/cpu.c, [1159](#)
  - ARMCM4\_S32K14/cpu.c, [1162](#)
  - ARMCM4\_STM32F3/cpu.c, [1165](#)
  - ARMCM4\_STM32F4/cpu.c, [1168](#)
  - ARMCM4\_STM32L4/cpu.c, [1171](#)
  - ARMCM4\_TM4C/cpu.c, [1174](#)
  - ARMCM4\_XMC4/cpu.c, [1177](#)
  - ARMCM7\_STM32F7/cpu.c, [1180](#)
  - ARMCM7\_STM32H7/cpu.c, [1183](#)
  - cpu.h, [1190](#)
  - HCS12/cpu.c, [1186](#)
- CpuStartUserProgram
  - \_template/cpu.c, [1133](#)
  - ARMCM0\_S32K11/cpu.c, [1136](#)
  - ARMCM0\_STM32F0/cpu.c, [1139](#)
  - ARMCM0\_STM32G0/cpu.c, [1142](#)
  - ARMCM0\_XMC1/cpu.c, [1145](#)
  - ARMCM33\_STM32L5/cpu.c, [1148](#)
  - ARMCM3\_EFM32/cpu.c, [1151](#)
  - ARMCM3\_LM3S/cpu.c, [1154](#)
  - ARMCM3\_STM32F1/cpu.c, [1157](#)
  - ARMCM3\_STM32F2/cpu.c, [1160](#)
  - ARMCM4\_S32K14/cpu.c, [1163](#)
  - ARMCM4\_STM32F3/cpu.c, [1166](#)
  - ARMCM4\_STM32F4/cpu.c, [1169](#)
  - ARMCM4\_STM32L4/cpu.c, [1172](#)
  - ARMCM4\_TM4C/cpu.c, [1175](#)
  - ARMCM4\_XMC4/cpu.c, [1178](#)
  - ARMCM7\_STM32F7/cpu.c, [1181](#)
  - ARMCM7\_STM32H7/cpu.c, [1184](#)
  - cpu.h, [1190](#)
  - HCS12/cpu.c, [1187](#)
- CpuUserProgramStartHook
  - \_template/Boot/hooks.c, [1622](#)
  - \_template/cpu.c, [1133](#)
  - ARMCM0\_S32K11/cpu.c, [1136](#)
  - ARMCM0\_STM32F0/cpu.c, [1139](#)
  - ARMCM0\_STM32F0\_Discovery\_STM32F051\_↔  
CubeIDE/Boot/App/hooks.c, [1627](#)
  - ARMCM0\_STM32F0\_Discovery\_STM32F051\_↔  
GCC/Boot/hooks.c, [1632](#)
  - ARMCM0\_STM32F0\_Discovery\_STM32F051\_↔  
AR/Boot/hooks.c, [1637](#)
  - ARMCM0\_STM32F0\_Discovery\_STM32F051\_↔  
Keil/Boot/hooks.c, [1642](#)
  - ARMCM0\_STM32F0\_Nucleo\_F091RC\_CubeID↔  
E/Boot/App/hooks.c, [1647](#)
  - ARMCM0\_STM32F0\_Nucleo\_F091RC\_GCC/↔  
Boot/hooks.c, [1652](#)
  - ARMCM0\_STM32F0\_Nucleo\_F091RC\_IAR/↔  
Boot/hooks.c, [1657](#)
  - ARMCM0\_STM32F0\_Nucleo\_F091RC\_Keil/↔  
Boot/hooks.c, [1662](#)
  - ARMCM0\_STM32G0/cpu.c, [1142](#)
  - ARMCM0\_STM32G0\_Nucleo\_G071RB\_CubeID↔  
E/Boot/App/hooks.c, [1667](#)
  - ARMCM0\_STM32G0\_Nucleo\_G071RB\_GCC/↔  
Boot/hooks.c, [1672](#)
  - ARMCM0\_STM32G0\_Nucleo\_G071RB\_IAR/↔  
Boot/hooks.c, [1677](#)
  - ARMCM0\_STM32G0\_Nucleo\_G071RB\_Keil/↔  
Boot/hooks.c, [1682](#)
  - ARMCM0\_XMC1/cpu.c, [1145](#)
  - ARMCM0\_XMC1\_XMC1400\_Boot\_Kit\_GCC/↔  
Boot/hooks.c, [1687](#)
  - ARMCM0\_XMC1\_XMC1400\_Boot\_Kit\_IAR/↔  
Boot/hooks.c, [1692](#)
  - ARMCM33\_STM32L5/cpu.c, [1148](#)
  - ARMCM33\_STM32L5\_Nucleo\_L552ZE\_CubeID↔  
E/Boot/App/hooks.c, [1697](#)
  - ARMCM33\_STM32L5\_Nucleo\_L552ZE\_GCC/↔  
Boot/hooks.c, [1703](#)
  - ARMCM33\_STM32L5\_Nucleo\_L552ZE\_IAR/↔  
Boot/hooks.c, [1709](#)
  - ARMCM33\_STM32L5\_Nucleo\_L552ZE\_Keil/↔  
Boot/hooks.c, [1715](#)
  - ARMCM3\_EFM32/cpu.c, [1151](#)
  - ARMCM3\_EFM32\_Olimex\_EM32G880F128ST↔  
K\_GCC/Boot/hooks.c, [1721](#)
  - ARMCM3\_EFM32\_Olimex\_EM32G880F128ST↔  
K\_IAR/Boot/hooks.c, [1726](#)
  - ARMCM3\_LM3S/cpu.c, [1154](#)
  - ARMCM3\_LM3S\_EK\_LM3S6965\_GCC/Boot/hooks.↔  
c, [1731](#)
  - ARMCM3\_LM3S\_EK\_LM3S6965\_IAR/Boot/hooks.↔  
c, [1739](#)
  - ARMCM3\_LM3S\_EK\_LM3S8962\_GCC/Boot/hooks.↔  
c, [1746](#)
  - ARMCM3\_LM3S\_EK\_LM3S8962\_IAR/Boot/hooks.↔  
c, [1752](#)
  - ARMCM3\_STM32F1/cpu.c, [1157](#)
  - ARMCM3\_STM32F1\_Nucleo\_F103RB\_CubeID↔  
E/Boot/App/hooks.c, [1757](#)
  - ARMCM3\_STM32F1\_Nucleo\_F103RB\_GCC/↔  
Boot/hooks.c, [1762](#)
  - ARMCM3\_STM32F1\_Nucleo\_F103RB\_IAR/↔  
Boot/hooks.c, [1767](#)
  - ARMCM3\_STM32F1\_Nucleo\_F103RB\_Keil/↔  
Boot/hooks.c, [1772](#)



- ARMCM3\_STM32F1\_Olimex\_STM32H103\_↔  
CubelIDE/Boot/App/hooks.c, [1777](#)
- ARMCM3\_STM32F1\_Olimex\_STM32H103\_GC↔  
C/Boot/hooks.c, [1783](#)
- ARMCM3\_STM32F1\_Olimex\_STM32H103\_IA↔  
R/Boot/hooks.c, [1789](#)
- ARMCM3\_STM32F1\_Olimex\_STM32H103\_↔  
Keil/Boot/hooks.c, [1795](#)
- ARMCM3\_STM32F1\_Olimex\_STM32P103\_↔  
CubelIDE/Boot/App/hooks.c, [1801](#)
- ARMCM3\_STM32F1\_Olimex\_STM32P103\_GC↔  
C/Boot/hooks.c, [1809](#)
- ARMCM3\_STM32F1\_Olimex\_STM32P103\_IA↔  
R/Boot/hooks.c, [1816](#)
- ARMCM3\_STM32F1\_Olimex\_STM32P103\_Keil/↔  
Boot/hooks.c, [1824](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_Cubel↔  
DE/Boot/App/hooks.c, [1832](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_GCC/↔  
Boot/hooks.c, [1840](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_IAR/↔  
Boot/hooks.c, [1849](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_Keil/↔  
Boot/hooks.c, [1858](#)
- ARMCM3\_STM32F2/cpu.c, [1160](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_↔  
CubelIDE/Boot/App/hooks.c, [1867](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_GC↔  
C/Boot/hooks.c, [1875](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_IA↔  
R/Boot/hooks.c, [1882](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_Keil/↔  
Boot/hooks.c, [1890](#)
- ARMCM4\_S32K14/cpu.c, [1163](#)
- ARMCM4\_S32K14\_S32K144EVB\_GCC/Boot/hooks.↔  
c, [1897](#)
- ARMCM4\_S32K14\_S32K144EVB\_IAR/Boot/hooks.↔  
c, [1903](#)
- ARMCM4\_STM32F3/cpu.c, [1166](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_Cube↔  
IDE/Boot/App/hooks.c, [1908](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_GCC/↔  
Boot/hooks.c, [1914](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_IAR/↔  
Boot/hooks.c, [1920](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_Keil/↔  
Boot/hooks.c, [1926](#)
- ARMCM4\_STM32F3\_Nucleo\_F303K8\_CubelD↔  
E/Boot/App/hooks.c, [1932](#)
- ARMCM4\_STM32F3\_Nucleo\_F303K8\_GCC/↔  
Boot/hooks.c, [1937](#)
- ARMCM4\_STM32F3\_Nucleo\_F303K8\_IAR/↔  
Boot/hooks.c, [1942](#)
- ARMCM4\_STM32F3\_Nucleo\_F303K8\_Keil/↔  
Boot/hooks.c, [1947](#)
- ARMCM4\_STM32F4/cpu.c, [1169](#)
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_CubelD↔  
E/Boot/App/hooks.c, [1952](#)
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_GCC/↔  
Boot/hooks.c, [1958](#)
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_IAR/↔  
Boot/hooks.c, [1964](#)
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_Keil/↔  
Boot/hooks.c, [1970](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_↔  
CubelIDE/Boot/App/hooks.c, [1976](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_GC↔  
C/Boot/hooks.c, [1985](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_IA↔  
R/Boot/hooks.c, [1993](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_Keil/↔  
Boot/hooks.c, [2002](#)
- ARMCM4\_STM32L4/cpu.c, [1172](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_CubelD↔  
E/Boot/App/hooks.c, [2011](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_GCC/↔  
Boot/hooks.c, [2016](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_IAR/↔  
Boot/hooks.c, [2021](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_Keil/↔  
Boot/hooks.c, [2026](#)
- ARMCM4\_TM4C/cpu.c, [1175](#)
- ARMCM4\_TM4C\_DK\_TM4C123G\_IAR/Boot/hooks.↔  
c, [2032](#)
- ARMCM4\_XMC4/cpu.c, [1178](#)
- ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_GCC/↔  
Boot/hooks.c, [2040](#)
- ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_IAR/↔  
Boot/hooks.c, [2048](#)
- ARMCM7\_STM32F7/cpu.c, [1181](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_CubelD↔  
E/Boot/App/hooks.c, [2055](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_GCC/↔  
Boot/hooks.c, [2062](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_IAR/↔  
Boot/hooks.c, [2068](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_Keil/↔  
Boot/hooks.c, [2074](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_CubelD↔  
E/Boot/App/hooks.c, [2080](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_GCC/↔  
Boot/hooks.c, [2085](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_IAR/↔  
Boot/hooks.c, [2090](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_Keil/↔  
Boot/hooks.c, [2095](#)
- ARMCM7\_STM32H7/cpu.c, [1184](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_CubelD↔  
E/Boot/App/hooks.c, [2100](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_GCC/↔  
Boot/hooks.c, [2106](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_IAR/↔  
Boot/hooks.c, [2112](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_Keil/↔  
Boot/hooks.c, [2118](#)
- HCS12/cpu.c, [1187](#)

- HCS12\_DevKit\_S12G128\_CodeWarrior/Boot/hooks.c, [2124](#)
- HCS12\_Evbplus\_Dragon12p\_CodeWarrior/Boot/hooks.c, [2129](#)
- crflg
  - tCanRegs, [361](#)
- crier
  - tCanRegs, [361](#)
- crxerr
  - tCanRegs, [362](#)
- cstart.c, [1247–1249](#), [1251–1253](#)
- ctaak
  - tCanRegs, [362](#)
- ctarq
  - tCanRegs, [362](#)
- ctbsel
  - tCanRegs, [362](#)
- ctflg
  - tCanRegs, [362](#)
- ctier
  - tCanRegs, [362](#)
- ctoData
  - tXcplInfo, [384](#)
- ctoLen
  - tXcplInfo, [384](#)
- ctoPending
  - tXcplInfo, [384](#)
- ctxerr
  - tCanRegs, [362](#)
- data
  - tFifoPipe, [369](#)
  - tSrecLineParseObject, [379](#)
- Demo for Dragon12-plus/CodeWarrior, [321](#)
- Demo for NXP DevKit-S12G128/CodeWarrior, [318](#)
- Demo for Nucleo-F091RC/GCC, [86](#)
- Demo for Nucleo-F091RC/IAR, [89](#)
- Demo for Nucleo-F091RC/Keil, [92](#)
- Demo for Nucleo-F091RC/STM32CubeIDE, [83](#)
- Demo for Nucleo-F103RB/GCC, [146](#)
- Demo for Nucleo-F103RB/Keil, [152](#)
- Demo for Nucleo-F103RB/STM32CubeIDE, [143](#)
- Demo for Nucleo-F103RB, [149](#)
- Demo for Nucleo-F303K8/GCC, [224](#)
- Demo for Nucleo-F303K8/IAR, [227](#)
- Demo for Nucleo-F303K8/Keil, [230](#)
- Demo for Nucleo-F303K8/STM32CubeIDE, [221](#)
- Demo for Nucleo-F429ZI/GCC, [236](#)
- Demo for Nucleo-F429ZI/IAR, [239](#)
- Demo for Nucleo-F429ZI/Keil, [242](#)
- Demo for Nucleo-F429ZI/STM32CubeIDE, [233](#)
- Demo for Nucleo-F746ZG/GCC, [281](#)
- Demo for Nucleo-F746ZG/IAR, [284](#)
- Demo for Nucleo-F746ZG/Keil, [287](#)
- Demo for Nucleo-F746ZG/STM32CubeIDE, [278](#)
- Demo for Nucleo-F767ZI/GCC, [293](#)
- Demo for Nucleo-F767ZI/IAR, [296](#)
- Demo for Nucleo-F767ZI/Keil, [299](#)
- Demo for Nucleo-F767ZI/STM32CubeIDE, [290](#)
- Demo for Nucleo-G071RB/GCC, [98](#)
- Demo for Nucleo-G071RB/IAR, [101](#)
- Demo for Nucleo-G071RB/Keil, [104](#)
- Demo for Nucleo-G071RB/STM32CubeIDE, [95](#)
- Demo for Nucleo-H743ZI/GCC, [305](#)
- Demo for Nucleo-H743ZI/IAR, [308](#)
- Demo for Nucleo-H743ZI/Keil, [311](#)
- Demo for Nucleo-H743ZI/STM32CubeIDE, [302](#)
- Demo for Nucleo-L476RG/GCC, [260](#)
- Demo for Nucleo-L476RG/IAR, [263](#)
- Demo for Nucleo-L476RG/Keil, [266](#)
- Demo for Nucleo-L476RG/STM32CubeIDE, [257](#)
- Demo for Nucleo-L552ZE/GCC, [116](#)
- Demo for Nucleo-L552ZE/IAR, [119](#)
- Demo for Nucleo-L552ZE/Keil, [122](#)
- Demo for Nucleo-L552ZE/STM32CubeIDE, [113](#)
- Demo for Olimex EM-32G880F128-STK/GCC, [125](#)
- Demo for Olimex EM-32G880F128-STK/IAR, [128](#)
- Demo for Olimex STM32-H103/GCC, [158](#)
- Demo for Olimex STM32-H103/IAR, [161](#)
- Demo for Olimex STM32-H103/Keil, [164](#)
- Demo for Olimex STM32-H103/STM32CubeIDE, [155](#)
- Demo for Olimex STM32-P103/GCC, [170](#)
- Demo for Olimex STM32-P103/IAR, [173](#)
- Demo for Olimex STM32-P103/Keil, [176](#)
- Demo for Olimex STM32-P103/STM32CubeIDE, [167](#)
- Demo for Olimex STM32-P207/GCC, [194](#)
- Demo for Olimex STM32-P207/IAR, [197](#)
- Demo for Olimex STM32-P207/Keil, [200](#)
- Demo for Olimex STM32-P207/STM32CubeIDE, [191](#)
- Demo for Olimex STM32-P405/GCC, [248](#)
- Demo for Olimex STM32-P405/IAR, [251](#)
- Demo for Olimex STM32-P405/Keil, [254](#)
- Demo for Olimex STM32-P405/STM32CubeIDE, [245](#)
- Demo for Olimexino-STM32/GCC, [182](#)
- Demo for Olimexino-STM32/IAR, [185](#)
- Demo for Olimexino-STM32/Keil, [188](#)
- Demo for Olimexino-STM32/STM32CubeIDE, [179](#)
- Demo for S32K118EVB/GCC, [65](#)
- Demo for S32K118EVB/IAR, [68](#)
- Demo for S32K144EVB/GCC, [203](#)
- Demo for S32K144EVB/IAR, [206](#)
- Demo for STM32F0-Discovery/GCC, [74](#)
- Demo for STM32F0-Discovery/IAR, [77](#)
- Demo for STM32F0-Discovery/Keil, [80](#)
- Demo for STM32F0-Discovery/STM32CubeIDE, [71](#)
- Demo for STM32F3-Discovery/GCC, [212](#)
- Demo for STM32F3-Discovery/IAR, [215](#)
- Demo for STM32F3-Discovery/Keil, [218](#)
- Demo for STM32F3-Discovery/STM32CubeIDE, [209](#)
- Demo for Texas Instruments DK-TM4C123G/IAR, [269](#)
- Demo for Texas Instruments EK-LM3S6965/GCC, [131](#)
- Demo for Texas Instruments EK-LM3S6965/IAR, [134](#)
- Demo for Texas Instruments EK-LM3S8962/GCC, [137](#)
- Demo for Texas Instruments EK-LM3S8962/IAR, [140](#)
- Demo for XMC1400 Boot Kit/GCC, [107](#)
- Demo for XMC1400 Boot Kit/IAR, [110](#)
- Demo for XMC4700 Relax Kit/GCC, [272](#)

- Demo for XMC4700 Relax Kit/IAR, [275](#)
- Demo/\_template/Prog/boot.c
  - BootActivate, [616](#)
  - BootComCheckActivationRequest, [617](#)
  - BootComInit, [617](#)
  - BootComRs232CheckActivationRequest, [617](#)
  - BootComRs232Init, [618](#)
  - Rs232ReceiveByte, [618](#)
- Demo/\_template/Prog/boot.h
  - BootActivate, [938](#)
  - BootComCheckActivationRequest, [938](#)
  - BootComInit, [938](#)
- Demo/\_template/Prog/timer.c
  - TimerGet, [3242](#)
  - TimerInit, [3243](#)
  - TimerInterrupt, [3243](#)
- Demo/\_template/Prog/timer.h
  - TimerGet, [3416](#)
  - TimerInit, [3416](#)
- Demo/ARMCM0\_STM32F0\_Discovery\_STM32F051↔
  - \_CubeIDE/Prog/App/boot.c
    - BootActivate, [620](#)
    - BootComCheckActivationRequest, [620](#)
    - BootComInit, [620](#)
    - BootComRs232CheckActivationRequest, [620](#)
    - BootComRs232Init, [621](#)
    - Rs232ReceiveByte, [621](#)
- Demo/ARMCM0\_STM32F0\_Discovery\_STM32F051↔
  - \_CubeIDE/Prog/App/boot.h
    - BootActivate, [940](#)
    - BootComCheckActivationRequest, [940](#)
    - BootComInit, [940](#)
- Demo/ARMCM0\_STM32F0\_Discovery\_STM32F051↔
  - \_CubeIDE/Prog/App/timer.c
    - TimerGet, [3244](#)
    - TimerInit, [3244](#)
- Demo/ARMCM0\_STM32F0\_Discovery\_STM32F051↔
  - \_CubeIDE/Prog/App/timer.h
    - TimerGet, [3417](#)
    - TimerInit, [3417](#)
- Demo/ARMCM0\_STM32F0\_Discovery\_STM32F051↔
  - \_GCC/Prog/boot.c
    - BootActivate, [623](#)
    - BootComCheckActivationRequest, [623](#)
    - BootComInit, [623](#)
    - BootComRs232CheckActivationRequest, [623](#)
    - BootComRs232Init, [624](#)
    - Rs232ReceiveByte, [624](#)
- Demo/ARMCM0\_STM32F0\_Discovery\_STM32F051↔
  - \_GCC/Prog/boot.h
    - BootActivate, [941](#)
    - BootComCheckActivationRequest, [941](#)
    - BootComInit, [942](#)
- Demo/ARMCM0\_STM32F0\_Discovery\_STM32F051↔
  - \_GCC/Prog/timer.c
    - SysTick\_Handler, [3245](#)
    - TimerGet, [3246](#)
    - TimerInit, [3246](#)
- Demo/ARMCM0\_STM32F0\_Discovery\_STM32F051↔
  - \_GCC/Prog/timer.h
    - TimerGet, [3419](#)
    - TimerInit, [3419](#)
- Demo/ARMCM0\_STM32F0\_Discovery\_STM32F051↔
  - \_IAR/Prog/boot.c
    - BootActivate, [626](#)
    - BootComCheckActivationRequest, [626](#)
    - BootComInit, [626](#)
    - BootComRs232CheckActivationRequest, [626](#)
    - BootComRs232Init, [627](#)
    - Rs232ReceiveByte, [627](#)
- Demo/ARMCM0\_STM32F0\_Discovery\_STM32F051↔
  - \_IAR/Prog/boot.h
    - BootActivate, [943](#)
    - BootComCheckActivationRequest, [943](#)
    - BootComInit, [943](#)
- Demo/ARMCM0\_STM32F0\_Discovery\_STM32F051↔
  - \_IAR/Prog/timer.c
    - SysTick\_Handler, [3247](#)
    - TimerGet, [3247](#)
    - TimerInit, [3247](#)
- Demo/ARMCM0\_STM32F0\_Discovery\_STM32F051↔
  - \_IAR/Prog/timer.h
    - TimerGet, [3420](#)
    - TimerInit, [3420](#)
- Demo/ARMCM0\_STM32F0\_Discovery\_STM32F051↔
  - \_Keil/Prog/boot.c
    - BootActivate, [629](#)
    - BootComCheckActivationRequest, [629](#)
    - BootComInit, [629](#)
    - BootComRs232CheckActivationRequest, [629](#)
    - BootComRs232Init, [630](#)
    - Rs232ReceiveByte, [630](#)
- Demo/ARMCM0\_STM32F0\_Discovery\_STM32F051↔
  - \_Keil/Prog/boot.h
    - BootActivate, [945](#)
    - BootComCheckActivationRequest, [945](#)
    - BootComInit, [945](#)
- Demo/ARMCM0\_STM32F0\_Discovery\_STM32F051↔
  - \_Keil/Prog/timer.c
    - SysTick\_Handler, [3248](#)
    - TimerGet, [3249](#)
    - TimerInit, [3249](#)
- Demo/ARMCM0\_STM32F0\_Discovery\_STM32F051↔
  - \_Keil/Prog/timer.h
    - TimerGet, [3421](#)
    - TimerInit, [3421](#)
- Demo/ARMCM0\_STM32F0\_Nucleo\_F091RC\_Cube↔
  - DE/Prog/App/boot.c
    - BootActivate, [632](#)
    - BootComCanCheckActivationRequest, [632](#)
    - BootComCanInit, [632](#)
    - BootComCheckActivationRequest, [633](#)
    - BootComInit, [633](#)
    - BootComRs232CheckActivationRequest, [633](#)
    - BootComRs232Init, [633](#)
    - CanGetSpeedConfig, [634](#)



- canTiming, [635](#)
- Rs232ReceiveByte, [634](#)
- Demo/ARMCM0\_STM32F0\_Nucleo\_F091RC\_Cubel↔
  - DE/Prog/App/boot.h
  - BootActivate, [946](#)
  - BootComCheckActivationRequest, [946](#)
  - BootComInit, [947](#)
- Demo/ARMCM0\_STM32F0\_Nucleo\_F091RC\_Cubel↔
  - DE/Prog/App/timer.c
  - TimerGet, [3250](#)
  - TimerInit, [3250](#)
- Demo/ARMCM0\_STM32F0\_Nucleo\_F091RC\_Cubel↔
  - DE/Prog/App/timer.h
  - TimerGet, [3423](#)
  - TimerInit, [3423](#)
- Demo/ARMCM0\_STM32F0\_Nucleo\_F091RC\_GCC/↔
  - Prog/boot.c
  - BootActivate, [637](#)
  - BootComCanCheckActivationRequest, [637](#)
  - BootComCanInit, [637](#)
  - BootComCheckActivationRequest, [638](#)
  - BootComInit, [638](#)
  - BootComRs232CheckActivationRequest, [638](#)
  - BootComRs232Init, [638](#)
  - CanGetSpeedConfig, [639](#)
  - canTiming, [640](#)
  - Rs232ReceiveByte, [639](#)
- Demo/ARMCM0\_STM32F0\_Nucleo\_F091RC\_GCC/↔
  - Prog/boot.h
  - BootActivate, [948](#)
  - BootComCheckActivationRequest, [948](#)
  - BootComInit, [948](#)
- Demo/ARMCM0\_STM32F0\_Nucleo\_F091RC\_GCC/↔
  - Prog/timer.c
  - SysTick\_Handler, [3251](#)
  - TimerGet, [3251](#)
  - TimerInit, [3252](#)
- Demo/ARMCM0\_STM32F0\_Nucleo\_F091RC\_GCC/↔
  - Prog/timer.h
  - TimerGet, [3424](#)
  - TimerInit, [3424](#)
- Demo/ARMCM0\_STM32F0\_Nucleo\_F091RC\_IAR/↔
  - Prog/boot.c
  - BootActivate, [642](#)
  - BootComCanCheckActivationRequest, [642](#)
  - BootComCanInit, [642](#)
  - BootComCheckActivationRequest, [643](#)
  - BootComInit, [643](#)
  - BootComRs232CheckActivationRequest, [643](#)
  - BootComRs232Init, [643](#)
  - CanGetSpeedConfig, [644](#)
  - canTiming, [645](#)
  - Rs232ReceiveByte, [644](#)
- Demo/ARMCM0\_STM32F0\_Nucleo\_F091RC\_IAR/↔
  - Prog/boot.h
  - BootActivate, [950](#)
  - BootComCheckActivationRequest, [950](#)
  - BootComInit, [950](#)
- Demo/ARMCM0\_STM32F0\_Nucleo\_F091RC\_IAR/↔
  - Prog/timer.c
  - SysTick\_Handler, [3253](#)
  - TimerGet, [3253](#)
  - TimerInit, [3253](#)
- Demo/ARMCM0\_STM32F0\_Nucleo\_F091RC\_IAR/↔
  - Prog/timer.h
  - TimerGet, [3425](#)
  - TimerInit, [3425](#)
- Demo/ARMCM0\_STM32F0\_Nucleo\_F091RC\_Keil/↔
  - Prog/boot.c
  - BootActivate, [647](#)
  - BootComCanCheckActivationRequest, [647](#)
  - BootComCanInit, [647](#)
  - BootComCheckActivationRequest, [648](#)
  - BootComInit, [648](#)
  - BootComRs232CheckActivationRequest, [648](#)
  - BootComRs232Init, [648](#)
  - CanGetSpeedConfig, [649](#)
  - canTiming, [650](#)
  - Rs232ReceiveByte, [649](#)
- Demo/ARMCM0\_STM32F0\_Nucleo\_F091RC\_Keil/↔
  - Prog/boot.h
  - BootActivate, [951](#)
  - BootComCheckActivationRequest, [951](#)
  - BootComInit, [952](#)
- Demo/ARMCM0\_STM32F0\_Nucleo\_F091RC\_Keil/↔
  - Prog/timer.c
  - SysTick\_Handler, [3254](#)
  - TimerGet, [3254](#)
  - TimerInit, [3255](#)
- Demo/ARMCM0\_STM32F0\_Nucleo\_F091RC\_Keil/↔
  - Prog/timer.h
  - TimerGet, [3427](#)
  - TimerInit, [3427](#)
- Demo/ARMCM0\_STM32G0\_Nucleo\_G071RB\_Cubel↔
  - DE/Prog/App/boot.c
  - BootActivate, [652](#)
  - BootComCheckActivationRequest, [652](#)
  - BootComInit, [652](#)
  - BootComRs232CheckActivationRequest, [652](#)
  - BootComRs232Init, [653](#)
  - Rs232ReceiveByte, [653](#)
- Demo/ARMCM0\_STM32G0\_Nucleo\_G071RB\_Cubel↔
  - DE/Prog/App/boot.h
  - BootActivate, [953](#)
  - BootComCheckActivationRequest, [953](#)
  - BootComInit, [953](#)
- Demo/ARMCM0\_STM32G0\_Nucleo\_G071RB\_Cubel↔
  - DE/Prog/App/timer.c
  - TimerGet, [3256](#)
  - TimerInit, [3256](#)
- Demo/ARMCM0\_STM32G0\_Nucleo\_G071RB\_Cubel↔
  - DE/Prog/App/timer.h
  - TimerGet, [3428](#)
  - TimerInit, [3428](#)
- Demo/ARMCM0\_STM32G0\_Nucleo\_G071RB\_GCC/↔
  - Prog/boot.c

- BootActivate, [655](#)
- BootComCheckActivationRequest, [655](#)
- BootComInit, [655](#)
- BootComRs232CheckActivationRequest, [655](#)
- BootComRs232Init, [656](#)
- Rs232ReceiveByte, [656](#)
- Demo/ARMCM0\_STM32G0\_Nucleo\_G071RB\_GCC/↔
  - Prog/boot.h
  - BootActivate, [955](#)
  - BootComCheckActivationRequest, [955](#)
  - BootComInit, [955](#)
- Demo/ARMCM0\_STM32G0\_Nucleo\_G071RB\_GCC/↔
  - Prog/timer.c
  - SysTick\_Handler, [3257](#)
  - TimerGet, [3257](#)
  - TimerInit, [3257](#)
- Demo/ARMCM0\_STM32G0\_Nucleo\_G071RB\_GCC/↔
  - Prog/timer.h
  - TimerGet, [3429](#)
  - TimerInit, [3429](#)
- Demo/ARMCM0\_STM32G0\_Nucleo\_G071RB\_IAR/↔
  - Prog/boot.c
  - BootActivate, [658](#)
  - BootComCheckActivationRequest, [658](#)
  - BootComInit, [658](#)
  - BootComRs232CheckActivationRequest, [658](#)
  - BootComRs232Init, [659](#)
  - Rs232ReceiveByte, [659](#)
- Demo/ARMCM0\_STM32G0\_Nucleo\_G071RB\_IAR/↔
  - Prog/boot.h
  - BootActivate, [956](#)
  - BootComCheckActivationRequest, [956](#)
  - BootComInit, [957](#)
- Demo/ARMCM0\_STM32G0\_Nucleo\_G071RB\_IAR/↔
  - Prog/timer.c
  - SysTick\_Handler, [3258](#)
  - TimerGet, [3259](#)
  - TimerInit, [3259](#)
- Demo/ARMCM0\_STM32G0\_Nucleo\_G071RB\_IAR/↔
  - Prog/timer.h
  - TimerGet, [3431](#)
  - TimerInit, [3431](#)
- Demo/ARMCM0\_STM32G0\_Nucleo\_G071RB\_Keil/↔
  - Prog/boot.c
  - BootActivate, [661](#)
  - BootComCheckActivationRequest, [661](#)
  - BootComInit, [661](#)
  - BootComRs232CheckActivationRequest, [661](#)
  - BootComRs232Init, [662](#)
  - Rs232ReceiveByte, [662](#)
- Demo/ARMCM0\_STM32G0\_Nucleo\_G071RB\_Keil/↔
  - Prog/boot.h
  - BootActivate, [958](#)
  - BootComCheckActivationRequest, [958](#)
  - BootComInit, [958](#)
- Demo/ARMCM0\_STM32G0\_Nucleo\_G071RB\_Keil/↔
  - Prog/timer.c
  - SysTick\_Handler, [3260](#)
- TimerGet, [3260](#)
- TimerInit, [3260](#)
- Demo/ARMCM0\_STM32G0\_Nucleo\_G071RB\_Keil/↔
  - Prog/timer.h
  - TimerGet, [3432](#)
  - TimerInit, [3432](#)
- Demo/ARMCM0\_XMC1\_XMC1400\_Boot\_Kit\_GCC/↔
  - Prog/boot.c
  - BootActivate, [664](#)
  - BootComCanCheckActivationRequest, [664](#)
  - BootComCanInit, [664](#)
  - BootComCheckActivationRequest, [665](#)
  - BootComInit, [665](#)
  - BootComRs232CheckActivationRequest, [665](#)
  - BootComRs232Init, [665](#)
  - Rs232ReceiveByte, [666](#)
- Demo/ARMCM0\_XMC1\_XMC1400\_Boot\_Kit\_GCC/↔
  - Prog/boot.h
  - BootActivate, [960](#)
  - BootComCheckActivationRequest, [960](#)
  - BootComInit, [960](#)
- Demo/ARMCM0\_XMC1\_XMC1400\_Boot\_Kit\_GCC/↔
  - Prog/timer.c
  - SysTick\_Handler, [3262](#)
  - TimerDeinit, [3262](#)
  - TimerGet, [3262](#)
  - TimerInit, [3262](#)
  - TimerSet, [3263](#)
- Demo/ARMCM0\_XMC1\_XMC1400\_Boot\_Kit\_GCC/↔
  - Prog/timer.h
  - TimerDeinit, [3433](#)
  - TimerGet, [3434](#)
  - TimerInit, [3434](#)
  - TimerSet, [3434](#)
- Demo/ARMCM0\_XMC1\_XMC1400\_Boot\_Kit\_IAR/↔
  - Prog/boot.c
  - BootActivate, [667](#)
  - BootComCanCheckActivationRequest, [667](#)
  - BootComCanInit, [668](#)
  - BootComCheckActivationRequest, [668](#)
  - BootComInit, [668](#)
  - BootComRs232CheckActivationRequest, [669](#)
  - BootComRs232Init, [669](#)
  - Rs232ReceiveByte, [669](#)
- Demo/ARMCM0\_XMC1\_XMC1400\_Boot\_Kit\_IAR/↔
  - Prog/boot.h
  - BootActivate, [961](#)
  - BootComCheckActivationRequest, [961](#)
  - BootComInit, [962](#)
- Demo/ARMCM0\_XMC1\_XMC1400\_Boot\_Kit\_IAR/↔
  - Prog/timer.c
  - SysTick\_Handler, [3264](#)
  - TimerDeinit, [3264](#)
  - TimerGet, [3264](#)
  - TimerInit, [3265](#)
  - TimerSet, [3265](#)
- Demo/ARMCM0\_XMC1\_XMC1400\_Boot\_Kit\_IAR/↔
  - Prog/timer.h

- TimerDeinit, [3436](#)
- TimerGet, [3436](#)
- TimerInit, [3436](#)
- TimerSet, [3436](#)
- Demo/ARMCM33\_STM32L5\_Nucleo\_L552ZE\_Cubel↔
  - DE/Prog/App/boot.c
  - BootActivate, [671](#)
  - BootComCanCheckActivationRequest, [671](#)
  - BootComCanInit, [672](#)
  - BootComCheckActivationRequest, [672](#)
  - BootComInit, [672](#)
  - BootComRs232CheckActivationRequest, [673](#)
  - BootComRs232Init, [673](#)
  - CanGetSpeedConfig, [673](#)
  - canTiming, [674](#)
  - Rs232ReceiveByte, [674](#)
- Demo/ARMCM33\_STM32L5\_Nucleo\_L552ZE\_Cubel↔
  - DE/Prog/App/boot.h
  - BootActivate, [963](#)
  - BootComCheckActivationRequest, [963](#)
  - BootComInit, [963](#)
- Demo/ARMCM33\_STM32L5\_Nucleo\_L552ZE\_Cubel↔
  - DE/Prog/App/timer.c
  - TimerGet, [3266](#)
  - TimerInit, [3266](#)
- Demo/ARMCM33\_STM32L5\_Nucleo\_L552ZE\_Cubel↔
  - DE/Prog/App/timer.h
  - TimerGet, [3437](#)
  - TimerInit, [3438](#)
- Demo/ARMCM33\_STM32L5\_Nucleo\_L552ZE\_GCC↔
  - Prog/boot.c
  - BootActivate, [676](#)
  - BootComCanCheckActivationRequest, [676](#)
  - BootComCanInit, [677](#)
  - BootComCheckActivationRequest, [677](#)
  - BootComInit, [677](#)
  - BootComRs232CheckActivationRequest, [678](#)
  - BootComRs232Init, [678](#)
  - CanGetSpeedConfig, [678](#)
  - canTiming, [679](#)
  - Rs232ReceiveByte, [679](#)
- Demo/ARMCM33\_STM32L5\_Nucleo\_L552ZE\_GCC↔
  - Prog/boot.h
  - BootActivate, [965](#)
  - BootComCheckActivationRequest, [965](#)
  - BootComInit, [965](#)
- Demo/ARMCM33\_STM32L5\_Nucleo\_L552ZE\_GCC↔
  - Prog/timer.c
  - SysTick\_Handler, [3267](#)
  - TimerGet, [3268](#)
  - TimerInit, [3268](#)
- Demo/ARMCM33\_STM32L5\_Nucleo\_L552ZE\_GCC↔
  - Prog/timer.h
  - TimerGet, [3439](#)
  - TimerInit, [3439](#)
- Demo/ARMCM33\_STM32L5\_Nucleo\_L552ZE\_IAR↔
  - Prog/boot.c
  - BootActivate, [681](#)
  - BootComCanCheckActivationRequest, [681](#)
  - BootComCanInit, [682](#)
  - BootComCheckActivationRequest, [682](#)
  - BootComInit, [682](#)
  - BootComRs232CheckActivationRequest, [683](#)
  - BootComRs232Init, [683](#)
  - CanGetSpeedConfig, [683](#)
  - canTiming, [684](#)
  - Rs232ReceiveByte, [684](#)
- Demo/ARMCM33\_STM32L5\_Nucleo\_L552ZE\_IAR↔
  - Prog/boot.h
  - BootActivate, [966](#)
  - BootComCheckActivationRequest, [966](#)
  - BootComInit, [967](#)
- Demo/ARMCM33\_STM32L5\_Nucleo\_L552ZE\_IAR↔
  - Prog/timer.c
  - SysTick\_Handler, [3269](#)
  - TimerGet, [3269](#)
  - TimerInit, [3269](#)
- Demo/ARMCM33\_STM32L5\_Nucleo\_L552ZE\_IAR↔
  - Prog/timer.h
  - TimerGet, [3440](#)
  - TimerInit, [3440](#)
- Demo/ARMCM33\_STM32L5\_Nucleo\_L552ZE\_Keil↔
  - Prog/boot.c
  - BootActivate, [686](#)
  - BootComCanCheckActivationRequest, [686](#)
  - BootComCanInit, [687](#)
  - BootComCheckActivationRequest, [687](#)
  - BootComInit, [687](#)
  - BootComRs232CheckActivationRequest, [688](#)
  - BootComRs232Init, [688](#)
  - CanGetSpeedConfig, [688](#)
  - canTiming, [689](#)
  - Rs232ReceiveByte, [689](#)
- Demo/ARMCM33\_STM32L5\_Nucleo\_L552ZE\_Keil↔
  - Prog/boot.h
  - BootActivate, [968](#)
  - BootComCheckActivationRequest, [968](#)
  - BootComInit, [968](#)
- Demo/ARMCM33\_STM32L5\_Nucleo\_L552ZE\_Keil↔
  - Prog/timer.c
  - SysTick\_Handler, [3270](#)
  - TimerGet, [3271](#)
  - TimerInit, [3271](#)
- Demo/ARMCM33\_STM32L5\_Nucleo\_L552ZE\_Keil↔
  - Prog/timer.h
  - TimerGet, [3442](#)
  - TimerInit, [3442](#)
- Demo/ARMCM33\_EFM32\_Olimex\_EM32G880F128S↔
  - TK\_GCC/Prog/boot.c
  - BootActivate, [691](#)
  - BootComCheckActivationRequest, [691](#)
  - BootComInit, [691](#)
  - BootComRs232CheckActivationRequest, [692](#)
  - BootComRs232Init, [692](#)
  - Rs232ReceiveByte, [692](#)

- Demo/ARMCM3\_EFM32\_Olimex\_EM32G880F128S↔
  - TK\_GCC/Prog/boot.h
  - BootActivate, [969](#)
  - BootComCheckActivationRequest, [970](#)
  - BootComInit, [970](#)
- Demo/ARMCM3\_EFM32\_Olimex\_EM32G880F128S↔
  - TK\_GCC/Prog/timer.c
  - TimerDeinit, [3272](#)
  - TimerGet, [3272](#)
  - TimerISRHandler, [3273](#)
  - TimerInit, [3272](#)
  - TimerSet, [3273](#)
- Demo/ARMCM3\_EFM32\_Olimex\_EM32G880F128S↔
  - TK\_GCC/Prog/timer.h
  - TimerDeinit, [3443](#)
  - TimerGet, [3443](#)
  - TimerISRHandler, [3444](#)
  - TimerInit, [3443](#)
  - TimerSet, [3444](#)
- Demo/ARMCM3\_EFM32\_Olimex\_EM32G880F128S↔
  - TK\_IAR/Prog/boot.c
  - BootActivate, [694](#)
  - BootComCheckActivationRequest, [694](#)
  - BootComInit, [694](#)
  - BootComRs232CheckActivationRequest, [694](#)
  - BootComRs232Init, [695](#)
  - Rs232ReceiveByte, [695](#)
- Demo/ARMCM3\_EFM32\_Olimex\_EM32G880F128S↔
  - TK\_IAR/Prog/boot.h
  - BootActivate, [971](#)
  - BootComCheckActivationRequest, [971](#)
  - BootComInit, [971](#)
- Demo/ARMCM3\_EFM32\_Olimex\_EM32G880F128S↔
  - TK\_IAR/Prog/timer.c
  - TimerDeinit, [3274](#)
  - TimerGet, [3274](#)
  - TimerISRHandler, [3275](#)
  - TimerInit, [3275](#)
  - TimerSet, [3275](#)
- Demo/ARMCM3\_EFM32\_Olimex\_EM32G880F128S↔
  - TK\_IAR/Prog/timer.h
  - TimerDeinit, [3445](#)
  - TimerGet, [3446](#)
  - TimerISRHandler, [3446](#)
  - TimerInit, [3446](#)
  - TimerSet, [3446](#)
- Demo/ARMCM3\_LM3S\_EK\_LM3S6965\_GCC/↔
  - Prog/boot.c
  - BootActivate, [696](#)
  - BootComCheckActivationRequest, [697](#)
  - BootComInit, [697](#)
  - BootComRs232CheckActivationRequest, [697](#)
  - BootComRs232Init, [697](#)
  - Rs232ReceiveByte, [698](#)
- Demo/ARMCM3\_LM3S\_EK\_LM3S6965\_GCC/↔
  - Prog/boot.h
  - BootActivate, [972](#)
  - BootComCheckActivationRequest, [973](#)
- BootComInit, [973](#)
- Demo/ARMCM3\_LM3S\_EK\_LM3S6965\_GCC/↔
  - Prog/net.c
  - NetApp, [2906](#)
  - NetInit, [2906](#)
  - NetTask, [2906](#)
- Demo/ARMCM3\_LM3S\_EK\_LM3S6965\_GCC/↔
  - Prog/net.h
  - NetApp, [2932](#)
  - NetInit, [2933](#)
  - NetTask, [2933](#)
- Demo/ARMCM3\_LM3S\_EK\_LM3S6965\_IAR/Prog/boot.↔
  - c
  - BootActivate, [699](#)
  - BootComCheckActivationRequest, [699](#)
  - BootComInit, [699](#)
  - BootComRs232CheckActivationRequest, [700](#)
  - BootComRs232Init, [700](#)
  - Rs232ReceiveByte, [700](#)
- Demo/ARMCM3\_LM3S\_EK\_LM3S6965\_IAR/Prog/boot.↔
  - h
  - BootActivate, [974](#)
  - BootComCheckActivationRequest, [974](#)
  - BootComInit, [974](#)
- Demo/ARMCM3\_LM3S\_EK\_LM3S6965\_IAR/Prog/net.↔
  - c
  - NetApp, [2908](#)
  - NetInit, [2908](#)
  - NetTask, [2908](#)
- Demo/ARMCM3\_LM3S\_EK\_LM3S6965\_IAR/Prog/net.↔
  - h
  - NetApp, [2934](#)
  - NetInit, [2935](#)
  - NetTask, [2935](#)
- Demo/ARMCM3\_LM3S\_EK\_LM3S8962\_GCC/↔
  - Prog/boot.c
  - BootActivate, [702](#)
  - BootComCanCheckActivationRequest, [702](#)
  - BootComCanInit, [702](#)
  - BootComCheckActivationRequest, [703](#)
  - BootComInit, [703](#)
  - BootComRs232CheckActivationRequest, [703](#)
  - BootComRs232Init, [703](#)
  - CanSetBittiming, [704](#)
  - Rs232ReceiveByte, [704](#)
- Demo/ARMCM3\_LM3S\_EK\_LM3S8962\_GCC/↔
  - Prog/boot.h
  - BootActivate, [975](#)
  - BootComCheckActivationRequest, [976](#)
  - BootComInit, [976](#)
- Demo/ARMCM3\_LM3S\_EK\_LM3S8962\_IAR/Prog/boot.↔
  - c
  - BootActivate, [706](#)
  - BootComCanCheckActivationRequest, [706](#)
  - BootComCanInit, [706](#)
  - BootComCheckActivationRequest, [706](#)
  - BootComInit, [707](#)
  - BootComRs232CheckActivationRequest, [707](#)

- BootComRs232Init, [707](#)
- CanSetBittiming, [707](#)
- Rs232ReceiveByte, [708](#)
- Demo/ARMCM3\_LM3S\_EK\_LM3S8962\_IAR/Prog/boot.↔  
h
  - BootActivate, [977](#)
  - BootComCheckActivationRequest, [977](#)
  - BootComInit, [977](#)
- Demo/ARMCM3\_STM32F1\_Nucleo\_F103RB\_Cubel↔  
DE/Prog/App/boot.c
  - BootActivate, [710](#)
  - BootComCheckActivationRequest, [710](#)
  - BootComInit, [710](#)
  - BootComRs232CheckActivationRequest, [710](#)
  - BootComRs232Init, [711](#)
  - Rs232ReceiveByte, [711](#)
- Demo/ARMCM3\_STM32F1\_Nucleo\_F103RB\_Cubel↔  
DE/Prog/App/boot.h
  - BootActivate, [979](#)
  - BootComCheckActivationRequest, [979](#)
  - BootComInit, [979](#)
- Demo/ARMCM3\_STM32F1\_Nucleo\_F103RB\_Cubel↔  
DE/Prog/App/timer.c
  - TimerGet, [3276](#)
  - TimerInit, [3276](#)
- Demo/ARMCM3\_STM32F1\_Nucleo\_F103RB\_Cubel↔  
DE/Prog/App/timer.h
  - TimerGet, [3447](#)
  - TimerInit, [3448](#)
- Demo/ARMCM3\_STM32F1\_Nucleo\_F103RB\_GCC/↔  
Prog/boot.c
  - BootActivate, [713](#)
  - BootComCheckActivationRequest, [713](#)
  - BootComInit, [713](#)
  - BootComRs232CheckActivationRequest, [713](#)
  - BootComRs232Init, [714](#)
  - Rs232ReceiveByte, [714](#)
- Demo/ARMCM3\_STM32F1\_Nucleo\_F103RB\_GCC/↔  
Prog/boot.h
  - BootActivate, [980](#)
  - BootComCheckActivationRequest, [980](#)
  - BootComInit, [981](#)
- Demo/ARMCM3\_STM32F1\_Nucleo\_F103RB\_GCC/↔  
Prog/timer.c
  - SysTick\_Handler, [3277](#)
  - TimerGet, [3278](#)
  - TimerInit, [3278](#)
- Demo/ARMCM3\_STM32F1\_Nucleo\_F103RB\_GCC/↔  
Prog/timer.h
  - TimerGet, [3449](#)
  - TimerInit, [3449](#)
- Demo/ARMCM3\_STM32F1\_Nucleo\_F103RB\_IAR/↔  
Prog/boot.c
  - BootActivate, [716](#)
  - BootComCheckActivationRequest, [716](#)
  - BootComInit, [716](#)
  - BootComRs232CheckActivationRequest, [716](#)
  - BootComRs232Init, [717](#)
  - Rs232ReceiveByte, [717](#)
- Demo/ARMCM3\_STM32F1\_Nucleo\_F103RB\_IAR/↔  
Prog/boot.h
  - BootActivate, [982](#)
  - BootComCheckActivationRequest, [982](#)
  - BootComInit, [982](#)
- Demo/ARMCM3\_STM32F1\_Nucleo\_F103RB\_IAR/↔  
Prog/timer.c
  - SysTick\_Handler, [3279](#)
  - TimerGet, [3279](#)
  - TimerInit, [3279](#)
- Demo/ARMCM3\_STM32F1\_Nucleo\_F103RB\_IAR/↔  
Prog/timer.h
  - TimerGet, [3450](#)
  - TimerInit, [3450](#)
- Demo/ARMCM3\_STM32F1\_Nucleo\_F103RB\_Keil/↔  
Prog/boot.c
  - BootActivate, [719](#)
  - BootComCheckActivationRequest, [719](#)
  - BootComInit, [719](#)
  - BootComRs232CheckActivationRequest, [719](#)
  - BootComRs232Init, [720](#)
  - Rs232ReceiveByte, [720](#)
- Demo/ARMCM3\_STM32F1\_Nucleo\_F103RB\_Keil/↔  
Prog/boot.h
  - BootActivate, [984](#)
  - BootComCheckActivationRequest, [984](#)
  - BootComInit, [984](#)
- Demo/ARMCM3\_STM32F1\_Nucleo\_F103RB\_Keil/↔  
Prog/timer.c
  - SysTick\_Handler, [3280](#)
  - TimerGet, [3281](#)
  - TimerInit, [3281](#)
- Demo/ARMCM3\_STM32F1\_Nucleo\_F103RB\_Keil/↔  
Prog/timer.h
  - TimerGet, [3452](#)
  - TimerInit, [3452](#)
- Demo/ARMCM3\_STM32F1\_Olimex\_STM32H103\_↔  
CubelIDE/Prog/App/timer.c
  - TimerGet, [3282](#)
  - TimerInit, [3282](#)
- Demo/ARMCM3\_STM32F1\_Olimex\_STM32H103\_↔  
CubelIDE/Prog/App/timer.h
  - TimerGet, [3453](#)
  - TimerInit, [3453](#)
- Demo/ARMCM3\_STM32F1\_Olimex\_STM32H103\_G↔  
CC/Prog/timer.c
  - SysTick\_Handler, [3284](#)
  - TimerGet, [3284](#)
  - TimerInit, [3284](#)
- Demo/ARMCM3\_STM32F1\_Olimex\_STM32H103\_G↔  
CC/Prog/timer.h
  - TimerGet, [3454](#)
  - TimerInit, [3454](#)
- Demo/ARMCM3\_STM32F1\_Olimex\_STM32H103\_IA↔  
R/Prog/timer.c
  - SysTick\_Handler, [3285](#)
  - TimerGet, [3285](#)



- TimerInit, [3286](#)
- Demo/ARMCM3\_STM32F1\_Olimex\_STM32H103\_IA↔
  - R/Prog/timer.h
  - TimerGet, [3456](#)
  - TimerInit, [3456](#)
- Demo/ARMCM3\_STM32F1\_Olimex\_STM32H103\_↔
  - Keil/Prog/timer.c
  - SysTick\_Handler, [3287](#)
  - TimerGet, [3287](#)
  - TimerInit, [3287](#)
- Demo/ARMCM3\_STM32F1\_Olimex\_STM32H103\_↔
  - Keil/Prog/timer.h
  - TimerGet, [3457](#)
  - TimerInit, [3457](#)
- Demo/ARMCM3\_STM32F1\_Olimex\_STM32P103\_↔
  - CubeIDE/Prog/App/boot.c
  - BootActivate, [722](#)
  - BootComCanCheckActivationRequest, [722](#)
  - BootComCanInit, [722](#)
  - BootComCheckActivationRequest, [723](#)
  - BootComInit, [723](#)
  - BootComRs232CheckActivationRequest, [723](#)
  - BootComRs232Init, [723](#)
  - CanGetSpeedConfig, [724](#)
  - canTiming, [725](#)
  - Rs232ReceiveByte, [724](#)
- Demo/ARMCM3\_STM32F1\_Olimex\_STM32P103\_↔
  - CubeIDE/Prog/App/boot.h
  - BootActivate, [985](#)
  - BootComCheckActivationRequest, [985](#)
  - BootComInit, [986](#)
- Demo/ARMCM3\_STM32F1\_Olimex\_STM32P103\_↔
  - CubeIDE/Prog/App/timer.c
  - TimerGet, [3288](#)
  - TimerInit, [3289](#)
- Demo/ARMCM3\_STM32F1\_Olimex\_STM32P103\_↔
  - CubeIDE/Prog/App/timer.h
  - TimerGet, [3458](#)
  - TimerInit, [3458](#)
- Demo/ARMCM3\_STM32F1\_Olimex\_STM32P103\_G↔
  - CC/Prog/boot.c
  - BootActivate, [727](#)
  - BootComCanCheckActivationRequest, [727](#)
  - BootComCanInit, [727](#)
  - BootComCheckActivationRequest, [728](#)
  - BootComInit, [728](#)
  - BootComRs232CheckActivationRequest, [728](#)
  - BootComRs232Init, [728](#)
  - CanGetSpeedConfig, [729](#)
  - canTiming, [730](#)
  - Rs232ReceiveByte, [729](#)
- Demo/ARMCM3\_STM32F1\_Olimex\_STM32P103\_G↔
  - CC/Prog/boot.h
  - BootActivate, [987](#)
  - BootComCheckActivationRequest, [987](#)
  - BootComInit, [987](#)
- Demo/ARMCM3\_STM32F1\_Olimex\_STM32P103\_G↔
  - CC/Prog/timer.c
  - SysTick\_Handler, [3290](#)
  - TimerGet, [3290](#)
  - TimerInit, [3290](#)
- Demo/ARMCM3\_STM32F1\_Olimex\_STM32P103\_G↔
  - CC/Prog/timer.h
  - TimerGet, [3460](#)
  - TimerInit, [3460](#)
- Demo/ARMCM3\_STM32F1\_Olimex\_STM32P103\_IA↔
  - R/Prog/boot.c
  - BootActivate, [732](#)
  - BootComCanCheckActivationRequest, [732](#)
  - BootComCanInit, [732](#)
  - BootComCheckActivationRequest, [733](#)
  - BootComInit, [733](#)
  - BootComRs232CheckActivationRequest, [733](#)
  - BootComRs232Init, [733](#)
  - CanGetSpeedConfig, [734](#)
  - canTiming, [735](#)
  - Rs232ReceiveByte, [734](#)
- Demo/ARMCM3\_STM32F1\_Olimex\_STM32P103\_IA↔
  - R/Prog/boot.h
  - BootActivate, [989](#)
  - BootComCheckActivationRequest, [989](#)
  - BootComInit, [989](#)
- Demo/ARMCM3\_STM32F1\_Olimex\_STM32P103\_IA↔
  - R/Prog/timer.c
  - SysTick\_Handler, [3291](#)
  - TimerGet, [3291](#)
  - TimerInit, [3292](#)
- Demo/ARMCM3\_STM32F1\_Olimex\_STM32P103\_IA↔
  - R/Prog/timer.h
  - TimerGet, [3461](#)
  - TimerInit, [3461](#)
- Demo/ARMCM3\_STM32F1\_Olimex\_STM32P103\_↔
  - Keil/Prog/boot.c
  - BootActivate, [737](#)
  - BootComCanCheckActivationRequest, [737](#)
  - BootComCanInit, [737](#)
  - BootComCheckActivationRequest, [738](#)
  - BootComInit, [738](#)
  - BootComRs232CheckActivationRequest, [738](#)
  - BootComRs232Init, [738](#)
  - CanGetSpeedConfig, [739](#)
  - canTiming, [740](#)
  - Rs232ReceiveByte, [739](#)
- Demo/ARMCM3\_STM32F1\_Olimex\_STM32P103\_↔
  - Keil/Prog/boot.h
  - BootActivate, [990](#)
  - BootComCheckActivationRequest, [990](#)
  - BootComInit, [991](#)
- Demo/ARMCM3\_STM32F1\_Olimex\_STM32P103\_↔
  - Keil/Prog/timer.c
  - SysTick\_Handler, [3293](#)
  - TimerGet, [3293](#)
  - TimerInit, [3293](#)
- Demo/ARMCM3\_STM32F1\_Olimex\_STM32P103\_↔
  - Keil/Prog/timer.h
  - TimerGet, [3462](#)

- TimerInit, [3462](#)
- Demo/ARMCM3\_STM32F1\_Olimexino\_STM32\_↔
  - CubeIDE/Prog/App/boot.c
  - BootActivate, [742](#)
  - BootComCanCheckActivationRequest, [742](#)
  - BootComCanInit, [742](#)
  - BootComCheckActivationRequest, [742](#)
  - BootComInit, [743](#)
  - CanGetSpeedConfig, [743](#)
  - canTiming, [744](#)
- Demo/ARMCM3\_STM32F1\_Olimexino\_STM32\_↔
  - CubeIDE/Prog/App/boot.h
  - BootActivate, [992](#)
  - BootComCheckActivationRequest, [992](#)
  - BootComInit, [992](#)
- Demo/ARMCM3\_STM32F1\_Olimexino\_STM32\_↔
  - CubeIDE/Prog/App/timer.c
  - TimerGet, [3294](#)
  - TimerInit, [3294](#)
- Demo/ARMCM3\_STM32F1\_Olimexino\_STM32\_↔
  - CubeIDE/Prog/App/timer.h
  - TimerGet, [3464](#)
  - TimerInit, [3464](#)
- Demo/ARMCM3\_STM32F1\_Olimexino\_STM32\_GC↔
  - C/Prog/boot.c
  - BootActivate, [745](#)
  - BootComCanCheckActivationRequest, [745](#)
  - BootComCanInit, [746](#)
  - BootComCheckActivationRequest, [746](#)
  - BootComInit, [746](#)
  - CanGetSpeedConfig, [747](#)
  - canTiming, [747](#)
- Demo/ARMCM3\_STM32F1\_Olimexino\_STM32\_GC↔
  - C/Prog/boot.h
  - BootActivate, [994](#)
  - BootComCheckActivationRequest, [994](#)
  - BootComInit, [994](#)
- Demo/ARMCM3\_STM32F1\_Olimexino\_STM32\_GC↔
  - C/Prog/timer.c
  - SysTick\_Handler, [3295](#)
  - TimerGet, [3296](#)
  - TimerInit, [3296](#)
- Demo/ARMCM3\_STM32F1\_Olimexino\_STM32\_GC↔
  - C/Prog/timer.h
  - TimerGet, [3465](#)
  - TimerInit, [3465](#)
- Demo/ARMCM3\_STM32F1\_Olimexino\_STM32\_IAR/↔
  - Prog/boot.c
  - BootActivate, [749](#)
  - BootComCanCheckActivationRequest, [749](#)
  - BootComCanInit, [750](#)
  - BootComCheckActivationRequest, [750](#)
  - BootComInit, [750](#)
  - CanGetSpeedConfig, [751](#)
  - canTiming, [751](#)
- Demo/ARMCM3\_STM32F1\_Olimexino\_STM32\_IAR/↔
  - Prog/boot.h
  - BootActivate, [995](#)
- BootComCheckActivationRequest, [995](#)
- BootComInit, [996](#)
- Demo/ARMCM3\_STM32F1\_Olimexino\_STM32\_IAR/↔
  - Prog/timer.c
  - SysTick\_Handler, [3297](#)
  - TimerGet, [3297](#)
  - TimerInit, [3297](#)
- Demo/ARMCM3\_STM32F1\_Olimexino\_STM32\_IAR/↔
  - Prog/timer.h
  - TimerGet, [3466](#)
  - TimerInit, [3466](#)
- Demo/ARMCM3\_STM32F1\_Olimexino\_STM32\_Keil/↔
  - Prog/boot.c
  - BootActivate, [753](#)
  - BootComCanCheckActivationRequest, [753](#)
  - BootComCanInit, [754](#)
  - BootComCheckActivationRequest, [754](#)
  - BootComInit, [754](#)
  - CanGetSpeedConfig, [755](#)
  - canTiming, [755](#)
- Demo/ARMCM3\_STM32F1\_Olimexino\_STM32\_Keil/↔
  - Prog/boot.h
  - BootActivate, [997](#)
  - BootComCheckActivationRequest, [997](#)
  - BootComInit, [997](#)
- Demo/ARMCM3\_STM32F1\_Olimexino\_STM32\_Keil/↔
  - Prog/timer.c
  - SysTick\_Handler, [3298](#)
  - TimerGet, [3299](#)
  - TimerInit, [3299](#)
- Demo/ARMCM3\_STM32F1\_Olimexino\_STM32\_Keil/↔
  - Prog/timer.h
  - TimerGet, [3468](#)
  - TimerInit, [3468](#)
- Demo/ARMCM3\_STM32F2\_Olimex\_STM32P207\_↔
  - CubeIDE/Prog/App/boot.c
  - BootActivate, [757](#)
  - BootComCanCheckActivationRequest, [758](#)
  - BootComCanInit, [758](#)
  - BootComCheckActivationRequest, [758](#)
  - BootComInit, [759](#)
  - BootComRs232CheckActivationRequest, [759](#)
  - BootComRs232Init, [759](#)
  - CanGetSpeedConfig, [759](#)
  - canTiming, [760](#)
  - Rs232ReceiveByte, [760](#)
- Demo/ARMCM3\_STM32F2\_Olimex\_STM32P207\_↔
  - CubeIDE/Prog/App/boot.h
  - BootActivate, [999](#)
  - BootComCheckActivationRequest, [999](#)
  - BootComInit, [999](#)
- Demo/ARMCM3\_STM32F2\_Olimex\_STM32P207\_↔
  - CubeIDE/Prog/App/timer.c
  - TimerGet, [3300](#)
  - TimerInit, [3300](#)
- Demo/ARMCM3\_STM32F2\_Olimex\_STM32P207\_↔
  - CubeIDE/Prog/App/timer.h
  - TimerGet, [3469](#)

- TimerInit, [3469](#)
- Demo/ARMCM3\_STM32F2\_Olimex\_STM32P207\_G↔
  - CC/Prog/boot.c
  - BootActivate, [763](#)
  - BootComCanCheckActivationRequest, [763](#)
  - BootComCanInit, [763](#)
  - BootComCheckActivationRequest, [763](#)
  - BootComInit, [764](#)
  - BootComRs232CheckActivationRequest, [764](#)
  - BootComRs232Init, [764](#)
  - CanGetSpeedConfig, [764](#)
  - canTiming, [765](#)
  - Rs232ReceiveByte, [765](#)
- Demo/ARMCM3\_STM32F2\_Olimex\_STM32P207\_G↔
  - CC/Prog/boot.h
  - BootActivate, [1000](#)
  - BootComCheckActivationRequest, [1000](#)
  - BootComInit, [1001](#)
- Demo/ARMCM3\_STM32F2\_Olimex\_STM32P207\_G↔
  - CC/Prog/timer.c
  - SysTick\_Handler, [3301](#)
  - TimerGet, [3301](#)
  - TimerInit, [3302](#)
- Demo/ARMCM3\_STM32F2\_Olimex\_STM32P207\_G↔
  - CC/Prog/timer.h
  - TimerGet, [3470](#)
  - TimerInit, [3470](#)
- Demo/ARMCM3\_STM32F2\_Olimex\_STM32P207\_IA↔
  - R/Prog/boot.c
  - BootActivate, [768](#)
  - BootComCanCheckActivationRequest, [768](#)
  - BootComCanInit, [768](#)
  - BootComCheckActivationRequest, [768](#)
  - BootComInit, [769](#)
  - BootComRs232CheckActivationRequest, [769](#)
  - BootComRs232Init, [769](#)
  - CanGetSpeedConfig, [769](#)
  - canTiming, [770](#)
  - Rs232ReceiveByte, [770](#)
- Demo/ARMCM3\_STM32F2\_Olimex\_STM32P207\_IA↔
  - R/Prog/boot.h
  - BootActivate, [1002](#)
  - BootComCheckActivationRequest, [1002](#)
  - BootComInit, [1002](#)
- Demo/ARMCM3\_STM32F2\_Olimex\_STM32P207\_IA↔
  - R/Prog/timer.c
  - SysTick\_Handler, [3303](#)
  - TimerGet, [3303](#)
  - TimerInit, [3303](#)
- Demo/ARMCM3\_STM32F2\_Olimex\_STM32P207\_IA↔
  - R/Prog/timer.h
  - TimerGet, [3472](#)
  - TimerInit, [3472](#)
- Demo/ARMCM3\_STM32F2\_Olimex\_STM32P207\_↔
  - Keil/Prog/boot.c
  - BootActivate, [773](#)
  - BootComCanCheckActivationRequest, [773](#)
  - BootComCanInit, [773](#)
  - BootComCheckActivationRequest, [773](#)
  - BootComInit, [774](#)
  - BootComRs232CheckActivationRequest, [774](#)
  - BootComRs232Init, [774](#)
  - CanGetSpeedConfig, [774](#)
  - canTiming, [775](#)
  - Rs232ReceiveByte, [775](#)
- Demo/ARMCM3\_STM32F2\_Olimex\_STM32P207\_↔
  - Keil/Prog/boot.h
  - BootActivate, [1004](#)
  - BootComCheckActivationRequest, [1004](#)
  - BootComInit, [1004](#)
- Demo/ARMCM3\_STM32F2\_Olimex\_STM32P207\_↔
  - Keil/Prog/timer.c
  - SysTick\_Handler, [3304](#)
  - TimerGet, [3304](#)
  - TimerInit, [3305](#)
- Demo/ARMCM3\_STM32F2\_Olimex\_STM32P207\_↔
  - Keil/Prog/timer.h
  - TimerGet, [3473](#)
  - TimerInit, [3473](#)
- Demo/ARMCM4\_S32K14\_S32K144EVB\_GCC/↔
  - Prog/boot.c
  - BootActivate, [778](#)
  - BootComCanCheckActivationRequest, [778](#)
  - BootComCanInit, [778](#)
  - BootComCheckActivationRequest, [779](#)
  - BootComInit, [779](#)
  - BootComRs232CheckActivationRequest, [779](#)
  - BootComRs232Init, [779](#)
  - CanDisabledModeEnter, [780](#)
  - CanDisabledModeExit, [780](#)
  - CanFreezeModeEnter, [780](#)
  - CanFreezeModeExit, [781](#)
  - CanGetSpeedConfig, [781](#)
  - canTiming, [782](#)
  - Rs232ReceiveByte, [782](#)
- Demo/ARMCM4\_S32K14\_S32K144EVB\_GCC/↔
  - Prog/boot.h
  - BootActivate, [1005](#)
  - BootComCheckActivationRequest, [1005](#)
  - BootComInit, [1006](#)
- Demo/ARMCM4\_S32K14\_S32K144EVB\_GCC/↔
  - Prog/timer.c
  - SysTick\_Handler, [3306](#)
  - TimerGet, [3306](#)
  - TimerInit, [3306](#)
- Demo/ARMCM4\_S32K14\_S32K144EVB\_GCC/↔
  - Prog/timer.h
  - TimerGet, [3474](#)
  - TimerInit, [3474](#)
- Demo/ARMCM4\_S32K14\_S32K144EVB\_IAR/↔
  - Prog/boot.c
  - BootActivate, [784](#)
  - BootComCanCheckActivationRequest, [785](#)
  - BootComCanInit, [785](#)
  - BootComCheckActivationRequest, [785](#)
  - BootComInit, [786](#)



- BootComRs232CheckActivationRequest, [786](#)
- BootComRs232Init, [786](#)
- CanDisabledModeEnter, [786](#)
- CanDisabledModeExit, [787](#)
- CanFreezeModeEnter, [787](#)
- CanFreezeModeExit, [787](#)
- CanGetSpeedConfig, [788](#)
- canTiming, [789](#)
- Rs232ReceiveByte, [788](#)
- Demo/ARMCM4\_S32K14\_S32K144EVB\_IAR/↔
  - Prog/boot.h
  - BootActivate, [1007](#)
  - BootComCheckActivationRequest, [1007](#)
  - BootComInit, [1007](#)
- Demo/ARMCM4\_S32K14\_S32K144EVB\_IAR/↔
  - Prog/timer.c
  - SysTick\_Handler, [3307](#)
  - TimerGet, [3307](#)
  - TimerInit, [3308](#)
- Demo/ARMCM4\_S32K14\_S32K144EVB\_IAR/↔
  - Prog/timer.h
  - TimerGet, [3476](#)
  - TimerInit, [3476](#)
- Demo/ARMCM4\_STM32F3\_Discovery\_F303VC\_↔
  - CubeIDE/Prog/App/timer.c
  - TimerGet, [3309](#)
  - TimerInit, [3309](#)
- Demo/ARMCM4\_STM32F3\_Discovery\_F303VC\_↔
  - CubeIDE/Prog/App/timer.h
  - TimerGet, [3477](#)
  - TimerInit, [3477](#)
- Demo/ARMCM4\_STM32F3\_Discovery\_F303VC\_GC↔
  - C/Prog/timer.c
  - SysTick\_Handler, [3310](#)
  - TimerGet, [3310](#)
  - TimerInit, [3311](#)
- Demo/ARMCM4\_STM32F3\_Discovery\_F303VC\_GC↔
  - C/Prog/timer.h
  - TimerGet, [3478](#)
  - TimerInit, [3478](#)
- Demo/ARMCM4\_STM32F3\_Discovery\_F303VC\_IA↔
  - R/Prog/timer.c
  - SysTick\_Handler, [3312](#)
  - TimerGet, [3312](#)
  - TimerInit, [3312](#)
- Demo/ARMCM4\_STM32F3\_Discovery\_F303VC\_IA↔
  - R/Prog/timer.h
  - TimerGet, [3480](#)
  - TimerInit, [3480](#)
- Demo/ARMCM4\_STM32F3\_Discovery\_F303VC\_↔
  - Keil/Prog/timer.c
  - SysTick\_Handler, [3314](#)
  - TimerGet, [3314](#)
  - TimerInit, [3314](#)
- Demo/ARMCM4\_STM32F3\_Discovery\_F303VC\_↔
  - Keil/Prog/timer.h
  - TimerGet, [3481](#)
  - TimerInit, [3481](#)
- Demo/ARMCM4\_STM32F3\_Nucleo\_F303K8\_Cubel↔
  - DE/Prog/App/boot.c
  - BootActivate, [791](#)
  - BootComCanCheckActivationRequest, [791](#)
  - BootComCanInit, [791](#)
  - BootComCheckActivationRequest, [792](#)
  - BootComInit, [792](#)
  - BootComRs232CheckActivationRequest, [792](#)
  - BootComRs232Init, [792](#)
  - CanGetSpeedConfig, [793](#)
  - canTiming, [794](#)
  - Rs232ReceiveByte, [793](#)
- Demo/ARMCM4\_STM32F3\_Nucleo\_F303K8\_Cubel↔
  - DE/Prog/App/boot.h
  - BootActivate, [1009](#)
  - BootComCheckActivationRequest, [1009](#)
  - BootComInit, [1009](#)
- Demo/ARMCM4\_STM32F3\_Nucleo\_F303K8\_Cubel↔
  - DE/Prog/App/timer.c
  - TimerGet, [3315](#)
  - TimerInit, [3315](#)
- Demo/ARMCM4\_STM32F3\_Nucleo\_F303K8\_Cubel↔
  - DE/Prog/App/timer.h
  - TimerGet, [3482](#)
  - TimerInit, [3482](#)
- Demo/ARMCM4\_STM32F3\_Nucleo\_F303K8\_GCC/↔
  - Prog/boot.c
  - BootActivate, [796](#)
  - BootComCanCheckActivationRequest, [796](#)
  - BootComCanInit, [796](#)
  - BootComCheckActivationRequest, [797](#)
  - BootComInit, [797](#)
  - BootComRs232CheckActivationRequest, [797](#)
  - BootComRs232Init, [797](#)
  - CanGetSpeedConfig, [798](#)
  - canTiming, [799](#)
  - Rs232ReceiveByte, [798](#)
- Demo/ARMCM4\_STM32F3\_Nucleo\_F303K8\_GCC/↔
  - Prog/boot.h
  - BootActivate, [1010](#)
  - BootComCheckActivationRequest, [1010](#)
  - BootComInit, [1011](#)
- Demo/ARMCM4\_STM32F3\_Nucleo\_F303K8\_GCC/↔
  - Prog/timer.c
  - SysTick\_Handler, [3316](#)
  - TimerGet, [3317](#)
  - TimerInit, [3317](#)
- Demo/ARMCM4\_STM32F3\_Nucleo\_F303K8\_GCC/↔
  - Prog/timer.h
  - TimerGet, [3484](#)
  - TimerInit, [3484](#)
- Demo/ARMCM4\_STM32F3\_Nucleo\_F303K8\_IAR/↔
  - Prog/boot.c
  - BootActivate, [801](#)
  - BootComCanCheckActivationRequest, [801](#)
  - BootComCanInit, [801](#)
  - BootComCheckActivationRequest, [802](#)
  - BootComInit, [802](#)

- BootComRs232CheckActivationRequest, [802](#)
- BootComRs232Init, [802](#)
- CanGetSpeedConfig, [803](#)
- canTiming, [804](#)
- Rs232ReceiveByte, [803](#)
- Demo/ARMCM4\_STM32F3\_Nucleo\_F303K8\_IAR/↔
  - Prog/boot.h
  - BootActivate, [1012](#)
  - BootComCheckActivationRequest, [1012](#)
  - BootComInit, [1012](#)
- Demo/ARMCM4\_STM32F3\_Nucleo\_F303K8\_IAR/↔
  - Prog/timer.c
  - SysTick\_Handler, [3318](#)
  - TimerGet, [3318](#)
  - TimerInit, [3318](#)
- Demo/ARMCM4\_STM32F3\_Nucleo\_F303K8\_IAR/↔
  - Prog/timer.h
  - TimerGet, [3485](#)
  - TimerInit, [3485](#)
- Demo/ARMCM4\_STM32F3\_Nucleo\_F303K8\_Keil/↔
  - Prog/boot.c
  - BootActivate, [806](#)
  - BootComCanCheckActivationRequest, [806](#)
  - BootComCanInit, [806](#)
  - BootComCheckActivationRequest, [807](#)
  - BootComInit, [807](#)
  - BootComRs232CheckActivationRequest, [807](#)
  - BootComRs232Init, [807](#)
  - CanGetSpeedConfig, [808](#)
  - canTiming, [809](#)
  - Rs232ReceiveByte, [808](#)
- Demo/ARMCM4\_STM32F3\_Nucleo\_F303K8\_Keil/↔
  - Prog/boot.h
  - BootActivate, [1014](#)
  - BootComCheckActivationRequest, [1014](#)
  - BootComInit, [1014](#)
- Demo/ARMCM4\_STM32F3\_Nucleo\_F303K8\_Keil/↔
  - Prog/timer.c
  - SysTick\_Handler, [3319](#)
  - TimerGet, [3320](#)
  - TimerInit, [3320](#)
- Demo/ARMCM4\_STM32F3\_Nucleo\_F303K8\_Keil/↔
  - Prog/timer.h
  - TimerGet, [3486](#)
  - TimerInit, [3486](#)
- Demo/ARMCM4\_STM32F4\_Nucleo\_F429ZI\_CubeID↔
  - E/Prog/App/boot.c
  - BootActivate, [811](#)
  - BootComCanCheckActivationRequest, [811](#)
  - BootComCanInit, [811](#)
  - BootComCheckActivationRequest, [812](#)
  - BootComInit, [812](#)
  - BootComRs232CheckActivationRequest, [812](#)
  - BootComRs232Init, [812](#)
  - CanGetSpeedConfig, [813](#)
  - canTiming, [814](#)
  - Rs232ReceiveByte, [813](#)
- Demo/ARMCM4\_STM32F4\_Nucleo\_F429ZI\_CubeID↔
  - E/Prog/App/boot.h
  - BootActivate, [1015](#)
  - BootComCheckActivationRequest, [1015](#)
  - BootComInit, [1016](#)
- Demo/ARMCM4\_STM32F4\_Nucleo\_F429ZI\_CubeID↔
  - E/Prog/App/net.c
  - NetApp, [2910](#)
  - NetInit, [2910](#)
  - NetTask, [2910](#)
- Demo/ARMCM4\_STM32F4\_Nucleo\_F429ZI\_CubeID↔
  - E/Prog/App/net.h
  - NetApp, [2936](#)
  - NetInit, [2937](#)
  - NetTask, [2937](#)
- Demo/ARMCM4\_STM32F4\_Nucleo\_F429ZI\_CubeID↔
  - E/Prog/App/timer.c
  - TimerGet, [3321](#)
  - TimerInit, [3321](#)
- Demo/ARMCM4\_STM32F4\_Nucleo\_F429ZI\_CubeID↔
  - E/Prog/App/timer.h
  - TimerGet, [3488](#)
  - TimerInit, [3488](#)
- Demo/ARMCM4\_STM32F4\_Nucleo\_F429ZI\_GCC/↔
  - Prog/boot.c
  - BootActivate, [816](#)
  - BootComCanCheckActivationRequest, [816](#)
  - BootComCanInit, [816](#)
  - BootComCheckActivationRequest, [817](#)
  - BootComInit, [817](#)
  - BootComRs232CheckActivationRequest, [817](#)
  - BootComRs232Init, [817](#)
  - CanGetSpeedConfig, [818](#)
  - canTiming, [819](#)
  - Rs232ReceiveByte, [818](#)
- Demo/ARMCM4\_STM32F4\_Nucleo\_F429ZI\_GCC/↔
  - Prog/boot.h
  - BootActivate, [1017](#)
  - BootComCheckActivationRequest, [1017](#)
  - BootComInit, [1017](#)
- Demo/ARMCM4\_STM32F4\_Nucleo\_F429ZI\_GCC/↔
  - Prog/net.c
  - NetApp, [2912](#)
  - NetInit, [2912](#)
  - NetTask, [2912](#)
- Demo/ARMCM4\_STM32F4\_Nucleo\_F429ZI\_GCC/↔
  - Prog/net.h
  - NetApp, [2938](#)
  - NetInit, [2939](#)
  - NetTask, [2939](#)
- Demo/ARMCM4\_STM32F4\_Nucleo\_F429ZI\_GCC/↔
  - Prog/timer.c
  - SysTick\_Handler, [3322](#)
  - TimerGet, [3322](#)
  - TimerInit, [3323](#)
- Demo/ARMCM4\_STM32F4\_Nucleo\_F429ZI\_GCC/↔
  - Prog/timer.h
  - TimerGet, [3489](#)

- TimerInit, [3489](#)
- Demo/ARMCM4\_STM32F4\_Nucleo\_F429ZI\_IAR/↔
  - Prog/boot.c
  - BootActivate, [821](#)
  - BootComCanCheckActivationRequest, [821](#)
  - BootComCanInit, [821](#)
  - BootComCheckActivationRequest, [822](#)
  - BootComInit, [822](#)
  - BootComRs232CheckActivationRequest, [822](#)
  - BootComRs232Init, [822](#)
  - CanGetSpeedConfig, [823](#)
  - canTiming, [824](#)
  - Rs232ReceiveByte, [823](#)
- Demo/ARMCM4\_STM32F4\_Nucleo\_F429ZI\_IAR/↔
  - Prog/boot.h
  - BootActivate, [1019](#)
  - BootComCheckActivationRequest, [1019](#)
  - BootComInit, [1019](#)
- Demo/ARMCM4\_STM32F4\_Nucleo\_F429ZI\_IAR/↔
  - Prog/net.c
  - NetApp, [2914](#)
  - NetInit, [2914](#)
  - NetTask, [2914](#)
- Demo/ARMCM4\_STM32F4\_Nucleo\_F429ZI\_IAR/↔
  - Prog/net.h
  - NetApp, [2940](#)
  - NetInit, [2941](#)
  - NetTask, [2941](#)
- Demo/ARMCM4\_STM32F4\_Nucleo\_F429ZI\_IAR/↔
  - Prog/timer.c
  - SysTick\_Handler, [3324](#)
  - TimerGet, [3324](#)
  - TimerInit, [3324](#)
- Demo/ARMCM4\_STM32F4\_Nucleo\_F429ZI\_IAR/↔
  - Prog/timer.h
  - TimerGet, [3490](#)
  - TimerInit, [3490](#)
- Demo/ARMCM4\_STM32F4\_Nucleo\_F429ZI\_Keil/↔
  - Prog/boot.c
  - BootActivate, [826](#)
  - BootComCanCheckActivationRequest, [826](#)
  - BootComCanInit, [826](#)
  - BootComCheckActivationRequest, [827](#)
  - BootComInit, [827](#)
  - BootComRs232CheckActivationRequest, [827](#)
  - BootComRs232Init, [827](#)
  - CanGetSpeedConfig, [828](#)
  - canTiming, [829](#)
  - Rs232ReceiveByte, [828](#)
- Demo/ARMCM4\_STM32F4\_Nucleo\_F429ZI\_Keil/↔
  - Prog/boot.h
  - BootActivate, [1020](#)
  - BootComCheckActivationRequest, [1020](#)
  - BootComInit, [1021](#)
- Demo/ARMCM4\_STM32F4\_Nucleo\_F429ZI\_Keil/↔
  - Prog/net.c
  - NetApp, [2916](#)
  - NetInit, [2916](#)
- NetTask, [2916](#)
- Demo/ARMCM4\_STM32F4\_Nucleo\_F429ZI\_Keil/↔
  - Prog/net.h
  - NetApp, [2942](#)
  - NetInit, [2943](#)
  - NetTask, [2943](#)
- Demo/ARMCM4\_STM32F4\_Nucleo\_F429ZI\_Keil/↔
  - Prog/timer.c
  - SysTick\_Handler, [3325](#)
  - TimerGet, [3325](#)
  - TimerInit, [3326](#)
- Demo/ARMCM4\_STM32F4\_Nucleo\_F429ZI\_Keil/↔
  - Prog/timer.h
  - TimerGet, [3492](#)
  - TimerInit, [3492](#)
- Demo/ARMCM4\_STM32F4\_Olimex\_STM32P405\_↔
  - CubeIDE/Prog/App/boot.c
  - BootActivate, [831](#)
  - BootComCanCheckActivationRequest, [831](#)
  - BootComCanInit, [831](#)
  - BootComCheckActivationRequest, [832](#)
  - BootComInit, [832](#)
  - BootComRs232CheckActivationRequest, [832](#)
  - BootComRs232Init, [832](#)
  - CanGetSpeedConfig, [833](#)
  - canTiming, [834](#)
  - Rs232ReceiveByte, [833](#)
- Demo/ARMCM4\_STM32F4\_Olimex\_STM32P405\_↔
  - CubeIDE/Prog/App/boot.h
  - BootActivate, [1022](#)
  - BootComCheckActivationRequest, [1022](#)
  - BootComInit, [1022](#)
- Demo/ARMCM4\_STM32F4\_Olimex\_STM32P405\_↔
  - CubeIDE/Prog/App/timer.c
  - TimerGet, [3327](#)
  - TimerInit, [3327](#)
- Demo/ARMCM4\_STM32F4\_Olimex\_STM32P405\_↔
  - CubeIDE/Prog/App/timer.h
  - TimerGet, [3493](#)
  - TimerInit, [3493](#)
- Demo/ARMCM4\_STM32F4\_Olimex\_STM32P405\_G↔
  - CC/Prog/boot.c
  - BootActivate, [836](#)
  - BootComCanCheckActivationRequest, [836](#)
  - BootComCanInit, [836](#)
  - BootComCheckActivationRequest, [837](#)
  - BootComInit, [837](#)
  - BootComRs232CheckActivationRequest, [837](#)
  - BootComRs232Init, [837](#)
  - CanGetSpeedConfig, [838](#)
  - canTiming, [839](#)
  - Rs232ReceiveByte, [838](#)
- Demo/ARMCM4\_STM32F4\_Olimex\_STM32P405\_G↔
  - CC/Prog/boot.h
  - BootActivate, [1024](#)
  - BootComCheckActivationRequest, [1024](#)
  - BootComInit, [1024](#)

- Demo/ARMCM4\_STM32F4\_Olimex\_STM32P405\_G↔  
 CC/Prog/timer.c  
 SysTick\_Handler, [3328](#)  
 TimerGet, [3328](#)  
 TimerInit, [3328](#)
- Demo/ARMCM4\_STM32F4\_Olimex\_STM32P405\_G↔  
 CC/Prog/timer.h  
 TimerGet, [3494](#)  
 TimerInit, [3494](#)
- Demo/ARMCM4\_STM32F4\_Olimex\_STM32P405\_IA↔  
 R/Prog/boot.c  
 BootActivate, [841](#)  
 BootComCanCheckActivationRequest, [841](#)  
 BootComCanInit, [841](#)  
 BootComCheckActivationRequest, [842](#)  
 BootComInit, [842](#)  
 BootComRs232CheckActivationRequest, [842](#)  
 BootComRs232Init, [842](#)  
 CanGetSpeedConfig, [843](#)  
 canTiming, [844](#)  
 Rs232ReceiveByte, [843](#)
- Demo/ARMCM4\_STM32F4\_Olimex\_STM32P405\_IA↔  
 R/Prog/boot.h  
 BootActivate, [1025](#)  
 BootComCheckActivationRequest, [1025](#)  
 BootComInit, [1026](#)
- Demo/ARMCM4\_STM32F4\_Olimex\_STM32P405\_IA↔  
 R/Prog/timer.c  
 SysTick\_Handler, [3329](#)  
 TimerGet, [3330](#)  
 TimerInit, [3330](#)
- Demo/ARMCM4\_STM32F4\_Olimex\_STM32P405\_IA↔  
 R/Prog/timer.h  
 TimerGet, [3496](#)  
 TimerInit, [3496](#)
- Demo/ARMCM4\_STM32F4\_Olimex\_STM32P405\_↔  
 Keil/Prog/boot.c  
 BootActivate, [846](#)  
 BootComCanCheckActivationRequest, [846](#)  
 BootComCanInit, [846](#)  
 BootComCheckActivationRequest, [847](#)  
 BootComInit, [847](#)  
 BootComRs232CheckActivationRequest, [847](#)  
 BootComRs232Init, [847](#)  
 CanGetSpeedConfig, [848](#)  
 canTiming, [849](#)  
 Rs232ReceiveByte, [848](#)
- Demo/ARMCM4\_STM32F4\_Olimex\_STM32P405\_↔  
 Keil/Prog/boot.h  
 BootActivate, [1027](#)  
 BootComCheckActivationRequest, [1027](#)  
 BootComInit, [1027](#)
- Demo/ARMCM4\_STM32F4\_Olimex\_STM32P405\_↔  
 Keil/Prog/timer.c  
 SysTick\_Handler, [3331](#)  
 TimerGet, [3331](#)  
 TimerInit, [3331](#)
- Demo/ARMCM4\_STM32F4\_Olimex\_STM32P405\_↔  
 Keil/Prog/timer.h  
 TimerGet, [3497](#)  
 TimerInit, [3497](#)
- Demo/ARMCM4\_STM32L4\_Nucleo\_L476RG\_Cubel↔  
 DE/Prog/App/boot.c  
 BootActivate, [851](#)  
 BootComCanCheckActivationRequest, [851](#)  
 BootComCanInit, [851](#)  
 BootComCheckActivationRequest, [852](#)  
 BootComInit, [852](#)  
 BootComRs232CheckActivationRequest, [852](#)  
 BootComRs232Init, [852](#)  
 CanGetSpeedConfig, [853](#)  
 canTiming, [854](#)  
 Rs232ReceiveByte, [853](#)
- Demo/ARMCM4\_STM32L4\_Nucleo\_L476RG\_Cubel↔  
 DE/Prog/App/boot.h  
 BootActivate, [1029](#)  
 BootComCheckActivationRequest, [1029](#)  
 BootComInit, [1029](#)
- Demo/ARMCM4\_STM32L4\_Nucleo\_L476RG\_Cubel↔  
 DE/Prog/App/timer.c  
 TimerGet, [3332](#)  
 TimerInit, [3333](#)
- Demo/ARMCM4\_STM32L4\_Nucleo\_L476RG\_Cubel↔  
 DE/Prog/App/timer.h  
 TimerGet, [3498](#)  
 TimerInit, [3498](#)
- Demo/ARMCM4\_STM32L4\_Nucleo\_L476RG\_GCC/↔  
 Prog/boot.c  
 BootActivate, [856](#)  
 BootComCanCheckActivationRequest, [856](#)  
 BootComCanInit, [856](#)  
 BootComCheckActivationRequest, [857](#)  
 BootComInit, [857](#)  
 BootComRs232CheckActivationRequest, [857](#)  
 BootComRs232Init, [857](#)  
 CanGetSpeedConfig, [858](#)  
 canTiming, [859](#)  
 Rs232ReceiveByte, [858](#)
- Demo/ARMCM4\_STM32L4\_Nucleo\_L476RG\_GCC/↔  
 Prog/boot.h  
 BootActivate, [1030](#)  
 BootComCheckActivationRequest, [1030](#)  
 BootComInit, [1031](#)
- Demo/ARMCM4\_STM32L4\_Nucleo\_L476RG\_GCC/↔  
 Prog/timer.c  
 SysTick\_Handler, [3334](#)  
 TimerGet, [3334](#)  
 TimerInit, [3334](#)
- Demo/ARMCM4\_STM32L4\_Nucleo\_L476RG\_GCC/↔  
 Prog/timer.h  
 TimerGet, [3500](#)  
 TimerInit, [3500](#)
- Demo/ARMCM4\_STM32L4\_Nucleo\_L476RG\_IAR/↔  
 Prog/boot.c  
 BootActivate, [861](#)

- BootComCanCheckActivationRequest, [861](#)
  - BootComCanInit, [861](#)
  - BootComCheckActivationRequest, [862](#)
  - BootComInit, [862](#)
  - BootComRs232CheckActivationRequest, [862](#)
  - BootComRs232Init, [862](#)
  - CanGetSpeedConfig, [863](#)
  - canTiming, [864](#)
  - Rs232ReceiveByte, [863](#)
- Demo/ARMCM4\_STM32L4\_Nucleo\_L476RG\_IAR/↔
  - Prog/boot.h
  - BootActivate, [1032](#)
  - BootComCheckActivationRequest, [1032](#)
  - BootComInit, [1032](#)
- Demo/ARMCM4\_STM32L4\_Nucleo\_L476RG\_IAR/↔
  - Prog/timer.c
  - SysTick\_Handler, [3335](#)
  - TimerGet, [3335](#)
  - TimerInit, [3336](#)
- Demo/ARMCM4\_STM32L4\_Nucleo\_L476RG\_IAR/↔
  - Prog/timer.h
  - TimerGet, [3501](#)
  - TimerInit, [3501](#)
- Demo/ARMCM4\_STM32L4\_Nucleo\_L476RG\_Keil/↔
  - Prog/boot.c
  - BootActivate, [866](#)
  - BootComCanCheckActivationRequest, [866](#)
  - BootComCanInit, [866](#)
  - BootComCheckActivationRequest, [867](#)
  - BootComInit, [867](#)
  - BootComRs232CheckActivationRequest, [867](#)
  - BootComRs232Init, [867](#)
  - CanGetSpeedConfig, [868](#)
  - canTiming, [869](#)
  - Rs232ReceiveByte, [868](#)
- Demo/ARMCM4\_STM32L4\_Nucleo\_L476RG\_Keil/↔
  - Prog/boot.h
  - BootActivate, [1034](#)
  - BootComCheckActivationRequest, [1034](#)
  - BootComInit, [1034](#)
- Demo/ARMCM4\_STM32L4\_Nucleo\_L476RG\_Keil/↔
  - Prog/timer.c
  - SysTick\_Handler, [3337](#)
  - TimerGet, [3337](#)
  - TimerInit, [3337](#)
- Demo/ARMCM4\_STM32L4\_Nucleo\_L476RG\_Keil/↔
  - Prog/timer.h
  - TimerGet, [3502](#)
  - TimerInit, [3502](#)
- Demo/ARMCM4\_TM4C\_DK\_TM4C123G\_IAR/↔
  - Prog/boot.c
  - BootActivate, [870](#)
  - BootComCheckActivationRequest, [870](#)
  - BootComInit, [871](#)
  - BootComRs232CheckActivationRequest, [871](#)
  - BootComRs232Init, [871](#)
  - Rs232ReceiveByte, [871](#)
- Demo/ARMCM4\_TM4C\_DK\_TM4C123G\_IAR/↔
  - Prog/boot.h
  - BootActivate, [1035](#)
  - BootComCheckActivationRequest, [1035](#)
  - BootComInit, [1036](#)
- Demo/ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_GCC/↔
  - Prog/boot.c
  - BootActivate, [873](#)
  - BootComCanCheckActivationRequest, [873](#)
  - BootComCanInit, [873](#)
  - BootComCheckActivationRequest, [874](#)
  - BootComInit, [874](#)
  - BootComRs232CheckActivationRequest, [874](#)
  - BootComRs232Init, [874](#)
  - Rs232ReceiveByte, [875](#)
- Demo/ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_GCC/↔
  - Prog/boot.h
  - BootActivate, [1037](#)
  - BootComCheckActivationRequest, [1037](#)
  - BootComInit, [1037](#)
- Demo/ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_GCC/↔
  - Prog/net.c
  - NetApp, [2918](#)
  - NetInit, [2918](#)
  - NetTask, [2918](#)
- Demo/ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_GCC/↔
  - Prog/net.h
  - NetApp, [2944](#)
  - NetInit, [2945](#)
  - NetTask, [2945](#)
- Demo/ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_GCC/↔
  - Prog/timer.c
  - SysTick\_Handler, [3338](#)
  - TimerDeinit, [3339](#)
  - TimerGet, [3339](#)
  - TimerInit, [3339](#)
  - TimerSet, [3339](#)
- Demo/ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_GCC/↔
  - Prog/timer.h
  - TimerDeinit, [3504](#)
  - TimerGet, [3504](#)
  - TimerInit, [3504](#)
  - TimerSet, [3504](#)
- Demo/ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_IAR/↔
  - Prog/boot.c
  - BootActivate, [877](#)
  - BootComCanCheckActivationRequest, [877](#)
  - BootComCanInit, [877](#)
  - BootComCheckActivationRequest, [877](#)
  - BootComInit, [878](#)
  - BootComRs232CheckActivationRequest, [878](#)
  - BootComRs232Init, [878](#)
  - Rs232ReceiveByte, [878](#)
- Demo/ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_IAR/↔
  - Prog/boot.h
  - BootActivate, [1039](#)
  - BootComCheckActivationRequest, [1039](#)
  - BootComInit, [1039](#)



- Demo/ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_IAR/↔
  - Prog/net.c
  - NetApp, [2920](#)
  - NetInit, [2920](#)
  - NetTask, [2920](#)
- Demo/ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_IAR/↔
  - Prog/net.h
  - NetApp, [2946](#)
  - NetInit, [2947](#)
  - NetTask, [2947](#)
- Demo/ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_IAR/↔
  - Prog/timer.c
  - SysTick\_Handler, [3341](#)
  - TimerDeinit, [3341](#)
  - TimerGet, [3341](#)
  - TimerInit, [3341](#)
  - TimerSet, [3342](#)
- Demo/ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_IAR/↔
  - Prog/timer.h
  - TimerDeinit, [3506](#)
  - TimerGet, [3506](#)
  - TimerInit, [3506](#)
  - TimerSet, [3506](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F746ZG\_Cubel↔
  - DE/Prog/App/boot.c
  - BootActivate, [880](#)
  - BootComCanCheckActivationRequest, [880](#)
  - BootComCanInit, [881](#)
  - BootComCheckActivationRequest, [881](#)
  - BootComInit, [881](#)
  - BootComRs232CheckActivationRequest, [882](#)
  - BootComRs232Init, [882](#)
  - CanGetSpeedConfig, [882](#)
  - canTiming, [883](#)
  - Rs232ReceiveByte, [883](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F746ZG\_Cubel↔
  - DE/Prog/App/boot.h
  - BootActivate, [1040](#)
  - BootComCheckActivationRequest, [1040](#)
  - BootComInit, [1041](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F746ZG\_Cubel↔
  - DE/Prog/App/timer.c
  - TimerGet, [3343](#)
  - TimerInit, [3343](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F746ZG\_Cubel↔
  - DE/Prog/App/timer.h
  - TimerGet, [3507](#)
  - TimerInit, [3508](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F746ZG\_GCC/↔
  - Prog/boot.c
  - BootActivate, [885](#)
  - BootComCanCheckActivationRequest, [885](#)
  - BootComCanInit, [886](#)
  - BootComCheckActivationRequest, [886](#)
  - BootComInit, [886](#)
  - BootComRs232CheckActivationRequest, [887](#)
  - BootComRs232Init, [887](#)
  - CanGetSpeedConfig, [887](#)
- canTiming, [888](#)
- Rs232ReceiveByte, [888](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F746ZG\_GCC/↔
  - Prog/boot.h
  - BootActivate, [1042](#)
  - BootComCheckActivationRequest, [1042](#)
  - BootComInit, [1042](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F746ZG\_GCC/↔
  - Prog/timer.c
  - SysTick\_Handler, [3344](#)
  - TimerGet, [3344](#)
  - TimerInit, [3345](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F746ZG\_GCC/↔
  - Prog/timer.h
  - TimerGet, [3509](#)
  - TimerInit, [3509](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F746ZG\_IAR/↔
  - Prog/boot.c
  - BootActivate, [890](#)
  - BootComCanCheckActivationRequest, [890](#)
  - BootComCanInit, [891](#)
  - BootComCheckActivationRequest, [891](#)
  - BootComInit, [891](#)
  - BootComRs232CheckActivationRequest, [892](#)
  - BootComRs232Init, [892](#)
  - CanGetSpeedConfig, [892](#)
  - canTiming, [893](#)
  - Rs232ReceiveByte, [893](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F746ZG\_IAR/↔
  - Prog/boot.h
  - BootActivate, [1044](#)
  - BootComCheckActivationRequest, [1044](#)
  - BootComInit, [1044](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F746ZG\_IAR/↔
  - Prog/timer.c
  - SysTick\_Handler, [3346](#)
  - TimerGet, [3346](#)
  - TimerInit, [3346](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F746ZG\_IAR/↔
  - Prog/timer.h
  - TimerGet, [3510](#)
  - TimerInit, [3510](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F746ZG\_Keil/↔
  - Prog/boot.c
  - BootActivate, [895](#)
  - BootComCanCheckActivationRequest, [895](#)
  - BootComCanInit, [896](#)
  - BootComCheckActivationRequest, [896](#)
  - BootComInit, [896](#)
  - BootComRs232CheckActivationRequest, [897](#)
  - BootComRs232Init, [897](#)
  - CanGetSpeedConfig, [897](#)
  - canTiming, [898](#)
  - Rs232ReceiveByte, [898](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F746ZG\_Keil/↔
  - Prog/boot.h
  - BootActivate, [1045](#)
  - BootComCheckActivationRequest, [1045](#)

- BootComInit, [1046](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F746ZG\_Keil/↔
  - Prog/timer.c
  - SysTick\_Handler, [3347](#)
  - TimerGet, [3347](#)
  - TimerInit, [3348](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F746ZG\_Keil/↔
  - Prog/timer.h
  - TimerGet, [3512](#)
  - TimerInit, [3512](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F767ZI\_CubeID↔
  - E/Prog/App/boot.c
  - BootActivate, [900](#)
  - BootComCheckActivationRequest, [900](#)
  - BootComInit, [901](#)
  - BootComRs232CheckActivationRequest, [901](#)
  - BootComRs232Init, [901](#)
  - Rs232ReceiveByte, [901](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F767ZI\_CubeID↔
  - E/Prog/App/boot.h
  - BootActivate, [1047](#)
  - BootComCheckActivationRequest, [1047](#)
  - BootComInit, [1047](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F767ZI\_CubeID↔
  - E/Prog/App/net.c
  - NetApp, [2922](#)
  - NetInit, [2922](#)
  - NetTask, [2922](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F767ZI\_CubeID↔
  - E/Prog/App/net.h
  - NetApp, [2948](#)
  - NetInit, [2949](#)
  - NetTask, [2949](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F767ZI\_CubeID↔
  - E/Prog/App/timer.c
  - TimerGet, [3349](#)
  - TimerInit, [3349](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F767ZI\_CubeID↔
  - E/Prog/App/timer.h
  - TimerGet, [3513](#)
  - TimerInit, [3513](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F767ZI\_GCC/↔
  - Prog/boot.c
  - BootActivate, [903](#)
  - BootComCheckActivationRequest, [903](#)
  - BootComInit, [903](#)
  - BootComRs232CheckActivationRequest, [904](#)
  - BootComRs232Init, [904](#)
  - Rs232ReceiveByte, [904](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F767ZI\_GCC/↔
  - Prog/boot.h
  - BootActivate, [1049](#)
  - BootComCheckActivationRequest, [1049](#)
  - BootComInit, [1049](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F767ZI\_GCC/↔
  - Prog/net.c
  - NetApp, [2924](#)
  - NetInit, [2924](#)
- NetTask, [2924](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F767ZI\_GCC/↔
  - Prog/net.h
  - NetApp, [2950](#)
  - NetInit, [2951](#)
  - NetTask, [2951](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F767ZI\_GCC/↔
  - Prog/timer.c
  - SysTick\_Handler, [3350](#)
  - TimerGet, [3350](#)
  - TimerInit, [3350](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F767ZI\_GCC/↔
  - Prog/timer.h
  - TimerGet, [3514](#)
  - TimerInit, [3514](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F767ZI\_IAR/↔
  - Prog/boot.c
  - BootActivate, [906](#)
  - BootComCheckActivationRequest, [906](#)
  - BootComInit, [906](#)
  - BootComRs232CheckActivationRequest, [907](#)
  - BootComRs232Init, [907](#)
  - Rs232ReceiveByte, [907](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F767ZI\_IAR/↔
  - Prog/boot.h
  - BootActivate, [1050](#)
  - BootComCheckActivationRequest, [1050](#)
  - BootComInit, [1051](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F767ZI\_IAR/↔
  - Prog/net.c
  - NetApp, [2926](#)
  - NetInit, [2926](#)
  - NetTask, [2926](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F767ZI\_IAR/↔
  - Prog/net.h
  - NetApp, [2952](#)
  - NetInit, [2953](#)
  - NetTask, [2953](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F767ZI\_IAR/↔
  - Prog/timer.c
  - SysTick\_Handler, [3351](#)
  - TimerGet, [3352](#)
  - TimerInit, [3352](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F767ZI\_IAR/↔
  - Prog/timer.h
  - TimerGet, [3516](#)
  - TimerInit, [3516](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F767ZI\_Keil/↔
  - Prog/boot.c
  - BootActivate, [909](#)
  - BootComCheckActivationRequest, [909](#)
  - BootComInit, [909](#)
  - BootComRs232CheckActivationRequest, [910](#)
  - BootComRs232Init, [910](#)
  - Rs232ReceiveByte, [910](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F767ZI\_Keil/↔
  - Prog/boot.h
  - BootActivate, [1052](#)

- BootComCheckActivationRequest, [1052](#)
- BootComInit, [1052](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F767ZI\_Keil/↔
  - Prog/net.c
  - NetApp, [2928](#)
  - NetInit, [2928](#)
  - NetTask, [2928](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F767ZI\_Keil/↔
  - Prog/net.h
  - NetApp, [2954](#)
  - NetInit, [2955](#)
  - NetTask, [2955](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F767ZI\_Keil/↔
  - Prog/timer.c
  - SysTick\_Handler, [3353](#)
  - TimerGet, [3353](#)
  - TimerInit, [3353](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F767ZI\_Keil/↔
  - Prog/timer.h
  - TimerGet, [3517](#)
  - TimerInit, [3517](#)
- Demo/ARMCM7\_STM32H7\_Nucleo\_H743ZI\_Cubel/↔
  - DE/Prog/App/boot.c
  - BootActivate, [912](#)
  - BootComCanCheckActivationRequest, [912](#)
  - BootComCanInit, [913](#)
  - BootComCheckActivationRequest, [913](#)
  - BootComInit, [913](#)
  - BootComRs232CheckActivationRequest, [914](#)
  - BootComRs232Init, [914](#)
  - CanGetSpeedConfig, [914](#)
  - canTiming, [915](#)
  - Rs232ReceiveByte, [915](#)
- Demo/ARMCM7\_STM32H7\_Nucleo\_H743ZI\_Cubel/↔
  - DE/Prog/App/boot.h
  - BootActivate, [1054](#)
  - BootComCheckActivationRequest, [1054](#)
  - BootComInit, [1054](#)
- Demo/ARMCM7\_STM32H7\_Nucleo\_H743ZI\_Cubel/↔
  - DE/Prog/App/timer.c
  - TimerGet, [3354](#)
  - TimerInit, [3355](#)
- Demo/ARMCM7\_STM32H7\_Nucleo\_H743ZI\_Cubel/↔
  - DE/Prog/App/timer.h
  - TimerGet, [3518](#)
  - TimerInit, [3518](#)
- Demo/ARMCM7\_STM32H7\_Nucleo\_H743ZI\_GCC/↔
  - Prog/boot.c
  - BootActivate, [917](#)
  - BootComCanCheckActivationRequest, [917](#)
  - BootComCanInit, [918](#)
  - BootComCheckActivationRequest, [918](#)
  - BootComInit, [918](#)
  - BootComRs232CheckActivationRequest, [919](#)
  - BootComRs232Init, [919](#)
  - CanGetSpeedConfig, [919](#)
  - canTiming, [920](#)
  - Rs232ReceiveByte, [920](#)
- Demo/ARMCM7\_STM32H7\_Nucleo\_H743ZI\_GCC/↔
  - Prog/boot.h
  - BootActivate, [1055](#)
  - BootComCheckActivationRequest, [1055](#)
  - BootComInit, [1056](#)
- Demo/ARMCM7\_STM32H7\_Nucleo\_H743ZI\_GCC/↔
  - Prog/timer.c
  - SysTick\_Handler, [3356](#)
  - TimerGet, [3356](#)
  - TimerInit, [3356](#)
- Demo/ARMCM7\_STM32H7\_Nucleo\_H743ZI\_GCC/↔
  - Prog/timer.h
  - TimerGet, [3520](#)
  - TimerInit, [3520](#)
- Demo/ARMCM7\_STM32H7\_Nucleo\_H743ZI\_IAR/↔
  - Prog/boot.c
  - BootActivate, [922](#)
  - BootComCanCheckActivationRequest, [922](#)
  - BootComCanInit, [923](#)
  - BootComCheckActivationRequest, [923](#)
  - BootComInit, [923](#)
  - BootComRs232CheckActivationRequest, [924](#)
  - BootComRs232Init, [924](#)
  - CanGetSpeedConfig, [924](#)
  - canTiming, [925](#)
  - Rs232ReceiveByte, [925](#)
- Demo/ARMCM7\_STM32H7\_Nucleo\_H743ZI\_IAR/↔
  - Prog/boot.h
  - BootActivate, [1057](#)
  - BootComCheckActivationRequest, [1057](#)
  - BootComInit, [1057](#)
- Demo/ARMCM7\_STM32H7\_Nucleo\_H743ZI\_IAR/↔
  - Prog/timer.c
  - SysTick\_Handler, [3357](#)
  - TimerGet, [3357](#)
  - TimerInit, [3358](#)
- Demo/ARMCM7\_STM32H7\_Nucleo\_H743ZI\_IAR/↔
  - Prog/timer.h
  - TimerGet, [3521](#)
  - TimerInit, [3521](#)
- Demo/ARMCM7\_STM32H7\_Nucleo\_H743ZI\_Keil/↔
  - Prog/boot.c
  - BootActivate, [927](#)
  - BootComCanCheckActivationRequest, [927](#)
  - BootComCanInit, [928](#)
  - BootComCheckActivationRequest, [928](#)
  - BootComInit, [928](#)
  - BootComRs232CheckActivationRequest, [929](#)
  - BootComRs232Init, [929](#)
  - CanGetSpeedConfig, [929](#)
  - canTiming, [930](#)
  - Rs232ReceiveByte, [930](#)
- Demo/ARMCM7\_STM32H7\_Nucleo\_H743ZI\_Keil/↔
  - Prog/boot.h
  - BootActivate, [1059](#)
  - BootComCheckActivationRequest, [1059](#)
  - BootComInit, [1059](#)



- Demo/ARMCM7\_STM32H7\_Nucleo\_H743ZI\_Keil/↔
  - Prog/timer.c
  - SysTick\_Handler, [3359](#)
  - TimerGet, [3359](#)
  - TimerInit, [3359](#)
- Demo/ARMCM7\_STM32H7\_Nucleo\_H743ZI\_Keil/↔
  - Prog/timer.h
  - TimerGet, [3522](#)
  - TimerInit, [3522](#)
- Demo/HCS12\_DevKit\_S12G128\_CodeWarrior/↔
  - Prog/timer.c
  - TimerDeinit, [3361](#)
  - TimerGet, [3361](#)
  - TimerISRHandler, [3361](#)
  - TimerInit, [3361](#)
  - TimerSet, [3361](#)
- Demo/HCS12\_DevKit\_S12G128\_CodeWarrior/↔
  - Prog/timer.h
  - TimerDeinit, [3524](#)
  - TimerGet, [3524](#)
  - TimerISRHandler, [3524](#)
  - TimerInit, [3524](#)
  - TimerSet, [3525](#)
- Demo/HCS12\_Evbplus\_Dragon12p\_CodeWarrior/↔
  - Prog/boot.c
  - BootActivate, [932](#)
  - BootComCanCheckActivationRequest, [932](#)
  - BootComCanInit, [933](#)
  - BootComCheckActivationRequest, [933](#)
  - BootComInit, [933](#)
  - BootComRs232CheckActivationRequest, [934](#)
  - BootComRs232Init, [934](#)
  - CanGetSpeedConfig, [934](#)
  - canTiming, [935](#)
  - Rs232ReceiveByte, [935](#)
- Demo/HCS12\_Evbplus\_Dragon12p\_CodeWarrior/↔
  - Prog/boot.h
  - BootActivate, [1060](#)
  - BootComCheckActivationRequest, [1060](#)
  - BootComInit, [1061](#)
- Demo/HCS12\_Evbplus\_Dragon12p\_CodeWarrior/↔
  - Prog/timer.c
  - TimerDeinit, [3363](#)
  - TimerGet, [3363](#)
  - TimerISRHandler, [3363](#)
  - TimerInit, [3363](#)
  - TimerSet, [3364](#)
- Demo/HCS12\_Evbplus\_Dragon12p\_CodeWarrior/↔
  - Prog/timer.h
  - TimerDeinit, [3526](#)
  - TimerGet, [3526](#)
  - TimerISRHandler, [3527](#)
  - TimerInit, [3526](#)
  - TimerSet, [3527](#)
- dfprot
  - tFlashRegs, [372](#)
- dls
  - tCanRxMsgSlot, [363](#)
- tCanTxMsgSlot, [365](#)
- dsr
  - tCanRxMsgSlot, [364](#)
  - tCanTxMsgSlot, [365](#)
- dummy
  - tCanRxMsgSlot, [364](#)
- dummy1
  - tCanRegs, [362](#)
- endptr
  - tFifoCtrl, [367](#)
- entries
  - tFifoCtrl, [367](#)
- fccob
  - tFlashRegs, [372](#)
- fccobix
  - tFlashRegs, [372](#)
- fcldiv
  - tFlashRegs, [373](#)
- fcmd
  - tFlashRegs, [373](#)
- fcnf
  - tFlashRegs, [373](#)
- fercnf
  - tFlashRegs, [373](#)
- ferstat
  - tFlashRegs, [373](#)
- fifoctrlptr
  - tFifoCtrl, [367](#)
- file
  - tFatFsObjects, [366](#)
- file.c, [1254](#)
  - FileFirmwareUpdateCompletedHook, [1256](#)
  - FileFirmwareUpdateErrorHook, [1257](#)
  - FileFirmwareUpdateLogHook, [1257](#)
  - FileFirmwareUpdateStartedHook, [1257](#)
  - FileGetFirmwareFilenameHook, [1258](#)
  - FileHandleFirmwareUpdateRequest, [1258](#)
  - FileInit, [1258](#)
  - FileIsFirmwareUpdateRequestedHook, [1259](#)
  - FileIsIdle, [1259](#)
  - FileLibByteNibbleToChar, [1259](#)
  - FileLibByteToHexString, [1260](#)
  - FileLibHexStringToByte, [1260](#)
  - FileLibLongToIntString, [1261](#)
  - FileSrecGetLineType, [1261](#)
  - FileSrecParseLine, [1261](#)
  - FileSrecVerifyChecksum, [1262](#)
  - FileTask, [1262](#)
  - tFirmwareUpdateState, [1256](#)
- file.h, [1263](#)
  - FileHandleFirmwareUpdateRequest, [1265](#)
  - FileInit, [1265](#)
  - FileIsIdle, [1265](#)
  - FileSrecGetLineType, [1266](#)
  - FileSrecParseLine, [1266](#)
  - FileSrecVerifyChecksum, [1267](#)
  - FileTask, [1267](#)

- tSrecLineType, [1264](#)
- FileFirmwareUpdateCompletedHook
- ARMCM3\_LM3S\_EK\_LM3S6965\_GCC/Boot/hooks.c, [1732](#)
  - ARMCM3\_LM3S\_EK\_LM3S6965\_IAR/Boot/hooks.c, [1739](#)
  - ARMCM3\_STM32F1\_Olimex\_STM32P103\_CubeIDE/Boot/App/hooks.c, [1801](#)
  - ARMCM3\_STM32F1\_Olimex\_STM32P103\_GC/C/Boot/hooks.c, [1809](#)
  - ARMCM3\_STM32F1\_Olimex\_STM32P103\_IAR/Boot/hooks.c, [1817](#)
  - ARMCM3\_STM32F1\_Olimex\_STM32P103\_Keil/Boot/hooks.c, [1824](#)
  - ARMCM3\_STM32F1\_Olimexino\_STM32\_CubeIDE/Boot/App/hooks.c, [1832](#)
  - ARMCM3\_STM32F1\_Olimexino\_STM32\_GCC/Boot/hooks.c, [1841](#)
  - ARMCM3\_STM32F1\_Olimexino\_STM32\_IAR/Boot/hooks.c, [1850](#)
  - ARMCM3\_STM32F1\_Olimexino\_STM32\_Keil/Boot/hooks.c, [1859](#)
  - ARMCM3\_STM32F2\_Olimex\_STM32P207\_CubeIDE/Boot/App/hooks.c, [1867](#)
  - ARMCM3\_STM32F2\_Olimex\_STM32P207\_GC/C/Boot/hooks.c, [1875](#)
  - ARMCM3\_STM32F2\_Olimex\_STM32P207\_IAR/Boot/hooks.c, [1883](#)
  - ARMCM3\_STM32F2\_Olimex\_STM32P207\_Keil/Boot/hooks.c, [1891](#)
  - ARMCM4\_STM32F4\_Olimex\_STM32P405\_CubeIDE/Boot/App/hooks.c, [1977](#)
  - ARMCM4\_STM32F4\_Olimex\_STM32P405\_GC/C/Boot/hooks.c, [1985](#)
  - ARMCM4\_STM32F4\_Olimex\_STM32P405\_IAR/Boot/hooks.c, [1994](#)
  - ARMCM4\_STM32F4\_Olimex\_STM32P405\_Keil/Boot/hooks.c, [2003](#)
  - ARMCM4\_TM4C\_DK\_TM4C123G\_IAR/Boot/hooks.c, [2032](#)
  - ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_GCC/Boot/hooks.c, [2041](#)
  - ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_IAR/Boot/hooks.c, [2048](#)
  - file.c, [1257](#)
- FileFirmwareUpdateLogHook
- ARMCM3\_LM3S\_EK\_LM3S6965\_GCC/Boot/hooks.c, [1732](#)
  - ARMCM3\_LM3S\_EK\_LM3S6965\_IAR/Boot/hooks.c, [1740](#)
  - ARMCM3\_STM32F1\_Olimex\_STM32P103\_CubeIDE/Boot/App/hooks.c, [1802](#)
  - ARMCM3\_STM32F1\_Olimex\_STM32P103\_GC/C/Boot/hooks.c, [1810](#)
  - ARMCM3\_STM32F1\_Olimex\_STM32P103\_IAR/Boot/hooks.c, [1817](#)
  - ARMCM3\_STM32F1\_Olimex\_STM32P103\_Keil/Boot/hooks.c, [1825](#)
  - ARMCM3\_STM32F1\_Olimexino\_STM32\_CubeIDE/Boot/App/hooks.c, [1833](#)
  - ARMCM3\_STM32F1\_Olimexino\_STM32\_GCC/Boot/hooks.c, [1841](#)
  - ARMCM3\_STM32F1\_Olimexino\_STM32\_IAR/Boot/hooks.c, [1850](#)
  - ARMCM3\_STM32F1\_Olimexino\_STM32\_Keil/Boot/hooks.c, [1859](#)
  - ARMCM3\_STM32F2\_Olimex\_STM32P207\_CubeIDE/Boot/App/hooks.c, [1868](#)
  - ARMCM3\_STM32F2\_Olimex\_STM32P207\_GC/C/Boot/hooks.c, [1876](#)
  - ARMCM3\_STM32F2\_Olimex\_STM32P207\_IAR/Boot/hooks.c, [1883](#)
- FileFirmwareUpdateErrorHook
- ARMCM3\_LM3S\_EK\_LM3S6965\_GCC/Boot/hooks.c, [1732](#)
  - ARMCM3\_LM3S\_EK\_LM3S6965\_IAR/Boot/hooks.c, [1740](#)
  - ARMCM3\_STM32F1\_Olimex\_STM32P103\_CubeIDE/Boot/App/hooks.c, [1802](#)
  - ARMCM3\_STM32F1\_Olimex\_STM32P103\_GC/C/Boot/hooks.c, [1809](#)
  - ARMCM3\_STM32F1\_Olimex\_STM32P103\_IAR/Boot/hooks.c, [1817](#)
  - ARMCM3\_STM32F1\_Olimex\_STM32P103\_Keil/Boot/hooks.c, [1825](#)

- ARMCM3\_STM32F2\_Olimex\_STM32P207\_Keil/↔  
Boot/hooks.c, [1891](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_↔  
CubeIDE/Boot/App/hooks.c, [1977](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_GC↔  
C/Boot/hooks.c, [1985](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_IA↔  
R/Boot/hooks.c, [1994](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_Keil/↔  
Boot/hooks.c, [2003](#)
- ARMCM4\_TM4C\_DK\_TM4C123G\_IAR/Boot/hooks.↔  
c, [2032](#)
- ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_GCC/↔  
Boot/hooks.c, [2041](#)
- ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_IAR/↔  
Boot/hooks.c, [2049](#)  
file.c, [1257](#)
- FileFirmwareUpdateStartedHook
  - ARMCM3\_LM3S\_EK\_LM3S6965\_GCC/Boot/hooks.↔  
c, [1733](#)
  - ARMCM3\_LM3S\_EK\_LM3S6965\_IAR/Boot/hooks.↔  
c, [1740](#)
  - ARMCM3\_STM32F1\_Olimex\_STM32P103\_↔  
CubeIDE/Boot/App/hooks.c, [1802](#)
  - ARMCM3\_STM32F1\_Olimex\_STM32P103\_GC↔  
C/Boot/hooks.c, [1810](#)
  - ARMCM3\_STM32F1\_Olimex\_STM32P103\_IA↔  
R/Boot/hooks.c, [1818](#)
  - ARMCM3\_STM32F1\_Olimex\_STM32P103\_Keil/↔  
Boot/hooks.c, [1825](#)
  - ARMCM3\_STM32F1\_Olimexino\_STM32\_Cubel↔  
DE/Boot/App/hooks.c, [1833](#)
  - ARMCM3\_STM32F1\_Olimexino\_STM32\_GCC/↔  
Boot/hooks.c, [1842](#)
  - ARMCM3\_STM32F1\_Olimexino\_STM32\_IAR/↔  
Boot/hooks.c, [1851](#)
  - ARMCM3\_STM32F1\_Olimexino\_STM32\_Keil/↔  
Boot/hooks.c, [1860](#)
  - ARMCM3\_STM32F2\_Olimex\_STM32P207\_↔  
CubeIDE/Boot/App/hooks.c, [1868](#)
  - ARMCM3\_STM32F2\_Olimex\_STM32P207\_GC↔  
C/Boot/hooks.c, [1876](#)
  - ARMCM3\_STM32F2\_Olimex\_STM32P207\_IA↔  
R/Boot/hooks.c, [1884](#)
  - ARMCM3\_STM32F2\_Olimex\_STM32P207\_Keil/↔  
Boot/hooks.c, [1892](#)
  - ARMCM4\_STM32F4\_Olimex\_STM32P405\_↔  
CubeIDE/Boot/App/hooks.c, [1978](#)
  - ARMCM4\_STM32F4\_Olimex\_STM32P405\_GC↔  
C/Boot/hooks.c, [1986](#)
  - ARMCM4\_STM32F4\_Olimex\_STM32P405\_IA↔  
R/Boot/hooks.c, [1995](#)
  - ARMCM4\_STM32F4\_Olimex\_STM32P405\_Keil/↔  
Boot/hooks.c, [2004](#)
  - ARMCM4\_TM4C\_DK\_TM4C123G\_IAR/Boot/hooks.↔  
c, [2033](#)
  - ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_GCC/↔  
Boot/hooks.c, [2041](#)
- ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_IAR/↔  
Boot/hooks.c, [2049](#)  
file.c, [1257](#)
- FileGetFirmwareFilenameHook
  - ARMCM3\_LM3S\_EK\_LM3S6965\_GCC/Boot/hooks.↔  
c, [1733](#)
  - ARMCM3\_LM3S\_EK\_LM3S6965\_IAR/Boot/hooks.↔  
c, [1741](#)
  - ARMCM3\_STM32F1\_Olimex\_STM32P103\_↔  
CubeIDE/Boot/App/hooks.c, [1803](#)
  - ARMCM3\_STM32F1\_Olimex\_STM32P103\_GC↔  
C/Boot/hooks.c, [1810](#)
  - ARMCM3\_STM32F1\_Olimex\_STM32P103\_IA↔  
R/Boot/hooks.c, [1818](#)
  - ARMCM3\_STM32F1\_Olimex\_STM32P103\_Keil/↔  
Boot/hooks.c, [1826](#)
  - ARMCM3\_STM32F1\_Olimexino\_STM32\_Cubel↔  
DE/Boot/App/hooks.c, [1833](#)
  - ARMCM3\_STM32F1\_Olimexino\_STM32\_GCC/↔  
Boot/hooks.c, [1842](#)
  - ARMCM3\_STM32F1\_Olimexino\_STM32\_IAR/↔  
Boot/hooks.c, [1851](#)
  - ARMCM3\_STM32F1\_Olimexino\_STM32\_Keil/↔  
Boot/hooks.c, [1860](#)
  - ARMCM3\_STM32F2\_Olimex\_STM32P207\_↔  
CubeIDE/Boot/App/hooks.c, [1869](#)
  - ARMCM3\_STM32F2\_Olimex\_STM32P207\_GC↔  
C/Boot/hooks.c, [1876](#)
  - ARMCM3\_STM32F2\_Olimex\_STM32P207\_IA↔  
R/Boot/hooks.c, [1884](#)
  - ARMCM3\_STM32F2\_Olimex\_STM32P207\_Keil/↔  
Boot/hooks.c, [1892](#)
  - ARMCM4\_STM32F4\_Olimex\_STM32P405\_↔  
CubeIDE/Boot/App/hooks.c, [1978](#)
  - ARMCM4\_STM32F4\_Olimex\_STM32P405\_GC↔  
C/Boot/hooks.c, [1986](#)
  - ARMCM4\_STM32F4\_Olimex\_STM32P405\_IA↔  
R/Boot/hooks.c, [1995](#)
  - ARMCM4\_STM32F4\_Olimex\_STM32P405\_Keil/↔  
Boot/hooks.c, [2004](#)
  - ARMCM4\_TM4C\_DK\_TM4C123G\_IAR/Boot/hooks.↔  
c, [2033](#)
  - ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_GCC/↔  
Boot/hooks.c, [2042](#)
  - ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_IAR/↔  
Boot/hooks.c, [2049](#)  
file.c, [1258](#)
- FileHandleFirmwareUpdateRequest
  - file.c, [1258](#)
  - file.h, [1265](#)
- FileInit
  - file.c, [1258](#)
  - file.h, [1265](#)
- FilesFirmwareUpdateRequestedHook
  - ARMCM3\_LM3S\_EK\_LM3S6965\_GCC/Boot/hooks.↔  
c, [1733](#)
  - ARMCM3\_LM3S\_EK\_LM3S6965\_IAR/Boot/hooks.↔  
c, [1741](#)

- ARMCM3\_STM32F1\_Olimex\_STM32P103\_↔  
CubelIDE/Boot/App/hooks.c, [1803](#)
- ARMCM3\_STM32F1\_Olimex\_STM32P103\_GC↔  
C/Boot/hooks.c, [1810](#)
- ARMCM3\_STM32F1\_Olimex\_STM32P103\_IA↔  
R/Boot/hooks.c, [1818](#)
- ARMCM3\_STM32F1\_Olimex\_STM32P103\_Keil/↔  
Boot/hooks.c, [1826](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_Cubel↔  
DE/Boot/App/hooks.c, [1833](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_GCC/↔  
Boot/hooks.c, [1842](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_IAR/↔  
Boot/hooks.c, [1851](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_Keil/↔  
Boot/hooks.c, [1860](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_↔  
CubelIDE/Boot/App/hooks.c, [1869](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_GC↔  
C/Boot/hooks.c, [1876](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_IA↔  
R/Boot/hooks.c, [1884](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_Keil/↔  
Boot/hooks.c, [1892](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_↔  
CubelIDE/Boot/App/hooks.c, [1978](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_GC↔  
C/Boot/hooks.c, [1986](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_IA↔  
R/Boot/hooks.c, [1995](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_Keil/↔  
Boot/hooks.c, [2004](#)
- ARMCM4\_TM4C\_DK\_TM4C123G\_IAR/Boot/hooks.↔  
c, [2033](#)
- ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_GCC/↔  
Boot/hooks.c, [2042](#)
- ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_IAR/↔  
Boot/hooks.c, [2050](#)
- file.c, [1259](#)
- FileIdle  
file.c, [1259](#)  
file.h, [1265](#)
- FileLibByteNibbleToChar  
file.c, [1259](#)
- FileLibByteToHexString  
file.c, [1260](#)
- FileLibHexStringToByte  
file.c, [1260](#)
- FileLibLongToIntString  
file.c, [1261](#)
- FileSrecGetLineType  
file.c, [1261](#)  
file.h, [1266](#)
- FileSrecParseLine  
file.c, [1261](#)  
file.h, [1266](#)
- FileSrecVerifyChecksum  
file.c, [1262](#)
- file.h, [1267](#)
- FileTask  
file.c, [1262](#)  
file.h, [1267](#)
- Flash driver of a port, [325](#)
- flash.c, [1268](#), [1276](#), [1285](#), [1293](#), [1302](#), [1311](#), [1320](#),  
[1329](#), [1338](#), [1345](#), [1354](#), [1362](#), [1369](#), [1377](#),  
[1386](#), [1395](#), [1405](#), [1414](#), [1422](#)
- flash.h, [1432](#), [1436](#), [1440](#), [1443](#), [1447](#), [1451](#), [1454](#),  
[1458](#), [1462](#), [1465](#), [1469](#), [1473](#), [1476](#), [1480](#),  
[1484](#), [1487](#), [1491](#), [1495](#), [1498](#)
- flash\_ecc.c, [1502](#)
- blockInfo, [1511](#)
- bootBlockInfo, [1512](#)
- FlashAddToBlock, [1505](#)
- FlashDone, [1506](#)
- FlashErase, [1506](#)
- flashExecCmd, [1512](#)
- FlashExecuteCommand, [1506](#)
- FlashGetGlobalAddrByte, [1507](#)
- FlashGetPhysAddr, [1507](#)
- FlashGetPhysPage, [1508](#)
- FlashGetUserProgBaseAddress, [1508](#)
- FlashInit, [1508](#)
- FlashInitBlock, [1508](#)
- flashLayout, [1513](#)
- FlashOperate, [1509](#)
- FlashReinit, [1509](#)
- FlashSwitchBlock, [1510](#)
- FlashVerifyChecksum, [1510](#)
- FlashWrite, [1510](#)
- FlashWriteBlock, [1511](#)
- FlashWriteChecksum, [1511](#)
- flash\_layout.c, [1513](#)–[1534](#)
- FlashAddToBlock  
\_template/flash.c, [1270](#)
- ARMCM0\_S32K11/flash.c, [1278](#)
- ARMCM0\_STM32F0/flash.c, [1287](#)
- ARMCM0\_STM32G0/flash.c, [1295](#)
- ARMCM0\_XMC1/flash.c, [1304](#)
- ARMCM33\_STM32L5/flash.c, [1313](#)
- ARMCM3\_EFM32/flash.c, [1322](#)
- ARMCM3\_LM3S/flash.c, [1331](#)
- ARMCM3\_STM32F1/flash.c, [1340](#)
- ARMCM3\_STM32F2/flash.c, [1347](#)
- ARMCM4\_S32K14/flash.c, [1356](#)
- ARMCM4\_STM32F3/flash.c, [1364](#)
- ARMCM4\_STM32F4/flash.c, [1371](#)
- ARMCM4\_STM32L4/flash.c, [1379](#)
- ARMCM4\_TM4C/flash.c, [1388](#)
- ARMCM4\_XMC4/flash.c, [1397](#)
- ARMCM7\_STM32F7/flash.c, [1407](#)
- ARMCM7\_STM32H7/flash.c, [1416](#)
- flash\_ecc.c, [1505](#)
- HCS12/flash.c, [1425](#)
- FlashCalcPageSize  
ARMCM3\_EFM32/flash.c, [1323](#)
- FlashCommandSequence

- ARMCM0\_S32K11/flash.c, [1279](#)
- ARMCM4\_S32K14/flash.c, [1356](#)
- FlashDone
  - \_template/flash.c, [1270](#)
  - \_template/flash.h, [1433](#)
  - ARMCM0\_S32K11/flash.c, [1279](#)
  - ARMCM0\_S32K11/flash.h, [1437](#)
  - ARMCM0\_STM32F0/flash.c, [1287](#)
  - ARMCM0\_STM32F0/flash.h, [1440](#)
  - ARMCM0\_STM32G0/flash.c, [1296](#)
  - ARMCM0\_STM32G0/flash.h, [1444](#)
  - ARMCM0\_XMC1/flash.c, [1305](#)
  - ARMCM0\_XMC1/flash.h, [1448](#)
  - ARMCM33\_STM32L5/flash.c, [1313](#)
  - ARMCM33\_STM32L5/flash.h, [1451](#)
  - ARMCM3\_EFM32/flash.c, [1323](#)
  - ARMCM3\_EFM32/flash.h, [1455](#)
  - ARMCM3\_LM3S/flash.c, [1332](#)
  - ARMCM3\_LM3S/flash.h, [1459](#)
  - ARMCM3\_STM32F1/flash.c, [1341](#)
  - ARMCM3\_STM32F1/flash.h, [1462](#)
  - ARMCM3\_STM32F2/flash.c, [1348](#)
  - ARMCM3\_STM32F2/flash.h, [1466](#)
  - ARMCM4\_S32K14/flash.c, [1356](#)
  - ARMCM4\_S32K14/flash.h, [1470](#)
  - ARMCM4\_STM32F3/flash.c, [1364](#)
  - ARMCM4\_STM32F3/flash.h, [1473](#)
  - ARMCM4\_STM32F4/flash.c, [1371](#)
  - ARMCM4\_STM32F4/flash.h, [1477](#)
  - ARMCM4\_STM32L4/flash.c, [1380](#)
  - ARMCM4\_STM32L4/flash.h, [1481](#)
  - ARMCM4\_TM4C/flash.c, [1389](#)
  - ARMCM4\_TM4C/flash.h, [1485](#)
  - ARMCM4\_XMC4/flash.c, [1398](#)
  - ARMCM4\_XMC4/flash.h, [1488](#)
  - ARMCM7\_STM32F7/flash.c, [1407](#)
  - ARMCM7\_STM32F7/flash.h, [1492](#)
  - ARMCM7\_STM32H7/flash.c, [1416](#)
  - ARMCM7\_STM32H7/flash.h, [1495](#)
  - flash\_ecc.c, [1506](#)
  - HCS12/flash.c, [1425](#)
  - HCS12/flash.h, [1499](#)
- FlashErase
  - \_template/flash.c, [1270](#)
  - \_template/flash.h, [1433](#)
  - ARMCM0\_S32K11/flash.c, [1280](#)
  - ARMCM0\_S32K11/flash.h, [1437](#)
  - ARMCM0\_STM32F0/flash.c, [1287](#)
  - ARMCM0\_STM32F0/flash.h, [1441](#)
  - ARMCM0\_STM32G0/flash.c, [1296](#)
  - ARMCM0\_STM32G0/flash.h, [1444](#)
  - ARMCM0\_XMC1/flash.c, [1305](#)
  - ARMCM0\_XMC1/flash.h, [1448](#)
  - ARMCM33\_STM32L5/flash.c, [1313](#)
  - ARMCM33\_STM32L5/flash.h, [1452](#)
  - ARMCM3\_EFM32/flash.c, [1323](#)
  - ARMCM3\_EFM32/flash.h, [1455](#)
  - ARMCM3\_LM3S/flash.c, [1332](#)
- ARMCM3\_LM3S/flash.h, [1459](#)
- ARMCM3\_STM32F1/flash.c, [1341](#)
- ARMCM3\_STM32F1/flash.h, [1463](#)
- ARMCM3\_STM32F2/flash.c, [1348](#)
- ARMCM3\_STM32F2/flash.h, [1466](#)
- ARMCM4\_S32K14/flash.c, [1357](#)
- ARMCM4\_S32K14/flash.h, [1470](#)
- ARMCM4\_STM32F3/flash.c, [1364](#)
- ARMCM4\_STM32F3/flash.h, [1474](#)
- ARMCM4\_STM32F4/flash.c, [1372](#)
- ARMCM4\_STM32F4/flash.h, [1477](#)
- ARMCM4\_STM32L4/flash.c, [1380](#)
- ARMCM4\_STM32L4/flash.h, [1481](#)
- ARMCM4\_TM4C/flash.c, [1389](#)
- ARMCM4\_TM4C/flash.h, [1485](#)
- ARMCM4\_XMC4/flash.c, [1398](#)
- ARMCM4\_XMC4/flash.h, [1488](#)
- ARMCM7\_STM32F7/flash.c, [1408](#)
- ARMCM7\_STM32F7/flash.h, [1492](#)
- ARMCM7\_STM32H7/flash.c, [1416](#)
- ARMCM7\_STM32H7/flash.h, [1496](#)
- flash\_ecc.c, [1506](#)
- HCS12/flash.c, [1425](#)
- HCS12/flash.h, [1499](#)
- FlashEraseSectors
  - \_template/flash.c, [1271](#)
  - ARMCM0\_S32K11/flash.c, [1280](#)
  - ARMCM0\_STM32F0/flash.c, [1288](#)
  - ARMCM0\_STM32G0/flash.c, [1296](#)
  - ARMCM0\_XMC1/flash.c, [1305](#)
  - ARMCM33\_STM32L5/flash.c, [1314](#)
  - ARMCM3\_EFM32/flash.c, [1324](#)
  - ARMCM3\_LM3S/flash.c, [1332](#)
  - ARMCM3\_STM32F2/flash.c, [1348](#)
  - ARMCM4\_S32K14/flash.c, [1357](#)
  - ARMCM4\_STM32F4/flash.c, [1372](#)
  - ARMCM4\_TM4C/flash.c, [1389](#)
  - ARMCM4\_XMC4/flash.c, [1399](#)
  - ARMCM7\_STM32F7/flash.c, [1408](#)
  - ARMCM7\_STM32H7/flash.c, [1417](#)
- flashExecCmd
  - flash\_ecc.c, [1512](#)
  - HCS12/flash.c, [1431](#)
- FlashExecuteCommand
  - flash\_ecc.c, [1506](#)
  - HCS12/flash.c, [1426](#)
- FlashGetBank
  - ARMCM33\_STM32L5/flash.c, [1314](#)
  - ARMCM4\_STM32L4/flash.c, [1380](#)
- FlashGetGlobalAddrByte
  - flash\_ecc.c, [1507](#)
- FlashGetLinearAddrByte
  - HCS12/flash.c, [1426](#)
- FlashGetPage
  - ARMCM33\_STM32L5/flash.c, [1314](#)
  - ARMCM4\_STM32L4/flash.c, [1382](#)
- FlashGetPageSize
  - ARMCM33\_STM32L5/flash.c, [1315](#)



- FlashGetPhysAddr
  - flash\_ecc.c, [1507](#)
  - HCS12/flash.c, [1426](#)
- FlashGetPhysPage
  - flash\_ecc.c, [1508](#)
  - HCS12/flash.c, [1427](#)
- FlashGetSector
  - ARMCM0\_STM32F0/flash.c, [1288](#)
  - ARMCM0\_STM32G0/flash.c, [1297](#)
  - ARMCM0\_XMC1/flash.c, [1306](#)
  - ARMCM3\_EFM32/flash.c, [1324](#)
  - ARMCM3\_LM3S/flash.c, [1333](#)
  - ARMCM3\_STM32F2/flash.c, [1349](#)
  - ARMCM4\_STM32F4/flash.c, [1373](#)
  - ARMCM4\_TM4C/flash.c, [1390](#)
  - ARMCM4\_XMC4/flash.c, [1399](#)
  - ARMCM7\_STM32F7/flash.c, [1409](#)
- FlashGetSectorBaseAddr
  - ARMCM0\_STM32F0/flash.c, [1288](#)
  - ARMCM0\_STM32G0/flash.c, [1297](#)
  - ARMCM0\_XMC1/flash.c, [1306](#)
  - ARMCM3\_EFM32/flash.c, [1324](#)
  - ARMCM3\_LM3S/flash.c, [1333](#)
  - ARMCM4\_TM4C/flash.c, [1390](#)
  - ARMCM4\_XMC4/flash.c, [1399](#)
- FlashGetSectorIdx
  - \_template/flash.c, [1271](#)
  - ARMCM0\_S32K11/flash.c, [1281](#)
  - ARMCM33\_STM32L5/flash.c, [1315](#)
  - ARMCM4\_S32K14/flash.c, [1358](#)
  - ARMCM7\_STM32H7/flash.c, [1417](#)
- FlashGetSectorSize
  - ARMCM0\_STM32F0/flash.c, [1289](#)
  - ARMCM0\_STM32G0/flash.c, [1298](#)
  - ARMCM3\_EFM32/flash.c, [1325](#)
  - ARMCM3\_LM3S/flash.c, [1334](#)
  - ARMCM4\_TM4C/flash.c, [1391](#)
- FlashGetUserProgBaseAddress
  - \_template/flash.c, [1272](#)
  - \_template/flash.h, [1434](#)
  - ARMCM0\_S32K11/flash.c, [1281](#)
  - ARMCM0\_S32K11/flash.h, [1438](#)
  - ARMCM0\_STM32F0/flash.c, [1289](#)
  - ARMCM0\_STM32F0/flash.h, [1441](#)
  - ARMCM0\_STM32G0/flash.c, [1298](#)
  - ARMCM0\_STM32G0/flash.h, [1445](#)
  - ARMCM0\_XMC1/flash.c, [1307](#)
  - ARMCM0\_XMC1/flash.h, [1449](#)
  - ARMCM33\_STM32L5/flash.c, [1316](#)
  - ARMCM33\_STM32L5/flash.h, [1452](#)
  - ARMCM3\_EFM32/flash.c, [1325](#)
  - ARMCM3\_EFM32/flash.h, [1456](#)
  - ARMCM3\_LM3S/flash.c, [1334](#)
  - ARMCM3\_LM3S/flash.h, [1460](#)
  - ARMCM3\_STM32F1/flash.c, [1341](#)
  - ARMCM3\_STM32F1/flash.h, [1463](#)
  - ARMCM3\_STM32F2/flash.c, [1349](#)
  - ARMCM3\_STM32F2/flash.h, [1467](#)
- ARMCM4\_S32K14/flash.c, [1358](#)
- ARMCM4\_S32K14/flash.h, [1471](#)
- ARMCM4\_STM32F3/flash.c, [1365](#)
- ARMCM4\_STM32F3/flash.h, [1474](#)
- ARMCM4\_STM32F4/flash.c, [1373](#)
- ARMCM4\_STM32F4/flash.h, [1478](#)
- ARMCM4\_STM32L4/flash.c, [1382](#)
- ARMCM4\_STM32L4/flash.h, [1482](#)
- ARMCM4\_TM4C/flash.c, [1391](#)
- ARMCM4\_TM4C/flash.h, [1485](#)
- ARMCM4\_XMC4/flash.c, [1400](#)
- ARMCM4\_XMC4/flash.h, [1489](#)
- ARMCM7\_STM32F7/flash.c, [1409](#)
- ARMCM7\_STM32F7/flash.h, [1493](#)
- ARMCM7\_STM32H7/flash.c, [1417](#)
- ARMCM7\_STM32H7/flash.h, [1496](#)
- flash\_ecc.c, [1508](#)
- HCS12/flash.c, [1427](#)
- HCS12/flash.h, [1500](#)
- FlashInit
  - \_template/flash.c, [1272](#)
  - \_template/flash.h, [1434](#)
  - ARMCM0\_S32K11/flash.c, [1281](#)
  - ARMCM0\_S32K11/flash.h, [1438](#)
  - ARMCM0\_STM32F0/flash.c, [1289](#)
  - ARMCM0\_STM32F0/flash.h, [1442](#)
  - ARMCM0\_STM32G0/flash.c, [1298](#)
  - ARMCM0\_STM32G0/flash.h, [1445](#)
  - ARMCM0\_XMC1/flash.c, [1307](#)
  - ARMCM0\_XMC1/flash.h, [1449](#)
  - ARMCM33\_STM32L5/flash.c, [1316](#)
  - ARMCM33\_STM32L5/flash.h, [1453](#)
  - ARMCM3\_EFM32/flash.c, [1325](#)
  - ARMCM3\_EFM32/flash.h, [1456](#)
  - ARMCM3\_LM3S/flash.c, [1334](#)
  - ARMCM3\_LM3S/flash.h, [1460](#)
  - ARMCM3\_STM32F1/flash.c, [1341](#)
  - ARMCM3\_STM32F1/flash.h, [1464](#)
  - ARMCM3\_STM32F2/flash.c, [1349](#)
  - ARMCM3\_STM32F2/flash.h, [1467](#)
  - ARMCM4\_S32K14/flash.c, [1358](#)
  - ARMCM4\_S32K14/flash.h, [1471](#)
  - ARMCM4\_STM32F3/flash.c, [1365](#)
  - ARMCM4\_STM32F3/flash.h, [1475](#)
  - ARMCM4\_STM32F4/flash.c, [1373](#)
  - ARMCM4\_STM32F4/flash.h, [1478](#)
  - ARMCM4\_STM32L4/flash.c, [1382](#)
  - ARMCM4\_STM32L4/flash.h, [1482](#)
  - ARMCM4\_TM4C/flash.c, [1391](#)
  - ARMCM4\_TM4C/flash.h, [1486](#)
  - ARMCM4\_XMC4/flash.c, [1400](#)
  - ARMCM4\_XMC4/flash.h, [1489](#)
  - ARMCM7\_STM32F7/flash.c, [1409](#)
  - ARMCM7\_STM32F7/flash.h, [1493](#)
  - ARMCM7\_STM32H7/flash.c, [1418](#)
  - ARMCM7\_STM32H7/flash.h, [1497](#)
  - flash\_ecc.c, [1508](#)
  - HCS12/flash.c, [1427](#)

- HCS12/flash.h, [1500](#)
- FlashInitBlock
  - \_template/flash.c, [1272](#)
  - ARMCM0\_S32K11/flash.c, [1281](#)
  - ARMCM0\_STM32F0/flash.c, [1290](#)
  - ARMCM0\_STM32G0/flash.c, [1298](#)
  - ARMCM0\_XMC1/flash.c, [1307](#)
  - ARMCM33\_STM32L5/flash.c, [1316](#)
  - ARMCM3\_EFM32/flash.c, [1326](#)
  - ARMCM3\_LM3S/flash.c, [1334](#)
  - ARMCM3\_STM32F1/flash.c, [1342](#)
  - ARMCM3\_STM32F2/flash.c, [1350](#)
  - ARMCM4\_S32K14/flash.c, [1358](#)
  - ARMCM4\_STM32F3/flash.c, [1365](#)
  - ARMCM4\_STM32F4/flash.c, [1373](#)
  - ARMCM4\_STM32L4/flash.c, [1383](#)
  - ARMCM4\_TM4C/flash.c, [1391](#)
  - ARMCM4\_XMC4/flash.c, [1400](#)
  - ARMCM7\_STM32F7/flash.c, [1409](#)
  - ARMCM7\_STM32H7/flash.c, [1418](#)
  - flash\_ecc.c, [1508](#)
  - HCS12/flash.c, [1428](#)
- FlashIsDualBankMode
  - ARMCM33\_STM32L5/flash.c, [1317](#)
- FlashIsSingleBankMode
  - ARMCM7\_STM32F7/flash.c, [1410](#)
- flashLayout
  - \_template/flash.c, [1275](#)
  - ARMCM0\_STM32F0/flash.c, [1293](#)
  - ARMCM0\_STM32F0\_Discovery\_STM32F051\_↔  
CubeIDE/Boot/App/flash\_layout.c, [1513](#)
  - ARMCM0\_STM32F0\_Discovery\_STM32F051\_↔  
GCC/Boot/flash\_layout.c, [1514](#)
  - ARMCM0\_STM32F0\_Discovery\_STM32F051\_I↔  
AR/Boot/flash\_layout.c, [1515](#)
  - ARMCM0\_STM32F0\_Discovery\_STM32F051\_↔  
Keil/Boot/flash\_layout.c, [1516](#)
  - ARMCM0\_STM32G0/flash.c, [1302](#)
  - ARMCM0\_XMC1/flash.c, [1310](#)
  - ARMCM33\_STM32L5/flash.c, [1320](#)
  - ARMCM3\_EFM32/flash.c, [1329](#)
  - ARMCM3\_LM3S/flash.c, [1338](#)
  - ARMCM3\_LM3S\_EK\_LM3S8962\_GCC/Boot/flash↔  
\_layout.c, [1517](#)
  - ARMCM3\_LM3S\_EK\_LM3S8962\_IAR/Boot/flash↔  
\_layout.c, [1518](#)
  - ARMCM3\_STM32F1/flash.c, [1345](#)
  - ARMCM3\_STM32F1\_Nucleo\_F103RB\_CubeID↔  
E/Boot/App/flash\_layout.c, [1519](#)
  - ARMCM3\_STM32F1\_Nucleo\_F103RB\_GCC/↔  
Boot/flash\_layout.c, [1520](#)
  - ARMCM3\_STM32F1\_Nucleo\_F103RB\_IAR/↔  
Boot/flash\_layout.c, [1521](#)
  - ARMCM3\_STM32F1\_Nucleo\_F103RB\_Keil/↔  
Boot/flash\_layout.c, [1522](#)
  - ARMCM3\_STM32F1\_Olimex\_STM32H103\_↔  
CubeIDE/Boot/App/flash\_layout.c, [1523](#)
  - ARMCM3\_STM32F1\_Olimex\_STM32H103\_GC↔  
C/Boot/flash\_layout.c, [1524](#)
  - ARMCM3\_STM32F1\_Olimex\_STM32H103\_IA↔  
R/Boot/flash\_layout.c, [1525](#)
  - ARMCM3\_STM32F1\_Olimex\_STM32H103\_↔  
Keil/Boot/flash\_layout.c, [1526](#)
  - ARMCM3\_STM32F1\_Olimexino\_STM32\_Cubel↔  
DE/Boot/App/flash\_layout.c, [1527](#)
  - ARMCM3\_STM32F1\_Olimexino\_STM32\_GCC/↔  
Boot/flash\_layout.c, [1528](#)
  - ARMCM3\_STM32F1\_Olimexino\_STM32\_IAR/↔  
Boot/flash\_layout.c, [1529](#)
  - ARMCM3\_STM32F1\_Olimexino\_STM32\_Keil/↔  
Boot/flash\_layout.c, [1530](#)
  - ARMCM3\_STM32F2/flash.c, [1353](#)
  - ARMCM4\_STM32F3/flash.c, [1368](#)
  - ARMCM4\_STM32F3\_Discovery\_F303VC\_Cube↔  
IDE/Boot/App/flash\_layout.c, [1531](#)
  - ARMCM4\_STM32F3\_Discovery\_F303VC\_GCC/↔  
Boot/flash\_layout.c, [1532](#)
  - ARMCM4\_STM32F3\_Discovery\_F303VC\_IAR/↔  
Boot/flash\_layout.c, [1533](#)
  - ARMCM4\_STM32F3\_Discovery\_F303VC\_Keil/↔  
Boot/flash\_layout.c, [1534](#)
  - ARMCM4\_STM32F4/flash.c, [1376](#)
  - ARMCM4\_STM32L4/flash.c, [1386](#)
  - ARMCM4\_TM4C/flash.c, [1395](#)
  - ARMCM4\_XMC4/flash.c, [1404](#)
  - ARMCM7\_STM32F7/flash.c, [1413](#)
  - ARMCM7\_STM32H7/flash.c, [1421](#)
  - flash\_ecc.c, [1513](#)
  - HCS12/flash.c, [1432](#)
- FlashOperate
  - flash\_ecc.c, [1509](#)
  - HCS12/flash.c, [1428](#)
- FlashReinit
  - \_template/flash.c, [1273](#)
  - \_template/flash.h, [1434](#)
  - ARMCM0\_S32K11/flash.c, [1282](#)
  - ARMCM0\_S32K11/flash.h, [1438](#)
  - ARMCM0\_STM32F0/flash.c, [1290](#)
  - ARMCM0\_STM32F0/flash.h, [1442](#)
  - ARMCM0\_STM32G0/flash.c, [1299](#)
  - ARMCM0\_STM32G0/flash.h, [1445](#)
  - ARMCM0\_XMC1/flash.c, [1308](#)
  - ARMCM0\_XMC1/flash.h, [1449](#)
  - ARMCM33\_STM32L5/flash.c, [1317](#)
  - ARMCM33\_STM32L5/flash.h, [1453](#)
  - ARMCM3\_EFM32/flash.c, [1326](#)
  - ARMCM3\_EFM32/flash.h, [1456](#)
  - ARMCM3\_LM3S/flash.c, [1335](#)
  - ARMCM3\_LM3S/flash.h, [1460](#)
  - ARMCM3\_STM32F1/flash.c, [1342](#)
  - ARMCM3\_STM32F1/flash.h, [1464](#)
  - ARMCM3\_STM32F2/flash.c, [1350](#)
  - ARMCM3\_STM32F2/flash.h, [1467](#)
  - ARMCM4\_S32K14/flash.c, [1359](#)
  - ARMCM4\_S32K14/flash.h, [1471](#)

- ARMCM4\_STM32F3/flash.c, [1366](#)
- ARMCM4\_STM32F3/flash.h, [1475](#)
- ARMCM4\_STM32F4/flash.c, [1374](#)
- ARMCM4\_STM32F4/flash.h, [1478](#)
- ARMCM4\_STM32L4/flash.c, [1383](#)
- ARMCM4\_STM32L4/flash.h, [1482](#)
- ARMCM4\_TM4C/flash.c, [1392](#)
- ARMCM4\_TM4C/flash.h, [1486](#)
- ARMCM4\_XMC4/flash.c, [1401](#)
- ARMCM4\_XMC4/flash.h, [1489](#)
- ARMCM7\_STM32F7/flash.c, [1410](#)
- ARMCM7\_STM32F7/flash.h, [1493](#)
- ARMCM7\_STM32H7/flash.c, [1418](#)
- ARMCM7\_STM32H7/flash.h, [1497](#)
- flash\_ecc.c, [1509](#)
- HCS12/flash.c, [1429](#)
- HCS12/flash.h, [1501](#)
- FlashSwitchBlock
  - \_template/flash.c, [1273](#)
  - ARMCM0\_S32K11/flash.c, [1282](#)
  - ARMCM0\_STM32F0/flash.c, [1290](#)
  - ARMCM0\_STM32G0/flash.c, [1299](#)
  - ARMCM0\_XMC1/flash.c, [1308](#)
  - ARMCM33\_STM32L5/flash.c, [1317](#)
  - ARMCM3\_EFM32/flash.c, [1326](#)
  - ARMCM3\_LM3S/flash.c, [1335](#)
  - ARMCM3\_STM32F1/flash.c, [1342](#)
  - ARMCM3\_STM32F2/flash.c, [1350](#)
  - ARMCM4\_S32K14/flash.c, [1359](#)
  - ARMCM4\_STM32F3/flash.c, [1366](#)
  - ARMCM4\_STM32F4/flash.c, [1374](#)
  - ARMCM4\_STM32L4/flash.c, [1383](#)
  - ARMCM4\_TM4C/flash.c, [1392](#)
  - ARMCM4\_XMC4/flash.c, [1401](#)
  - ARMCM7\_STM32F7/flash.c, [1410](#)
  - ARMCM7\_STM32H7/flash.c, [1419](#)
  - flash\_ecc.c, [1510](#)
  - HCS12/flash.c, [1429](#)
- FlashTranslateToNonCachedAddress
  - ARMCM4\_XMC4/flash.c, [1401](#)
- FlashVerifyChecksum
  - \_template/flash.c, [1273](#)
  - \_template/flash.h, [1435](#)
  - ARMCM0\_S32K11/flash.c, [1282](#)
  - ARMCM0\_S32K11/flash.h, [1438](#)
  - ARMCM0\_STM32F0/flash.c, [1291](#)
  - ARMCM0\_STM32F0/flash.h, [1442](#)
  - ARMCM0\_STM32G0/flash.c, [1300](#)
  - ARMCM0\_STM32G0/flash.h, [1446](#)
  - ARMCM0\_XMC1/flash.c, [1308](#)
  - ARMCM0\_XMC1/flash.h, [1449](#)
  - ARMCM33\_STM32L5/flash.c, [1318](#)
  - ARMCM33\_STM32L5/flash.h, [1453](#)
  - ARMCM3\_EFM32/flash.c, [1327](#)
  - ARMCM3\_EFM32/flash.h, [1457](#)
  - ARMCM3\_LM3S/flash.c, [1336](#)
  - ARMCM3\_LM3S/flash.h, [1461](#)
  - ARMCM3\_STM32F1/flash.c, [1343](#)
  - ARMCM3\_STM32F1/flash.h, [1464](#)
  - ARMCM3\_STM32F2/flash.c, [1351](#)
  - ARMCM3\_STM32F2/flash.h, [1468](#)
  - ARMCM4\_S32K14/flash.c, [1360](#)
  - ARMCM4\_S32K14/flash.h, [1472](#)
  - ARMCM4\_STM32F3/flash.c, [1367](#)
  - ARMCM4\_STM32F3/flash.h, [1475](#)
  - ARMCM4\_STM32F4/flash.c, [1375](#)
  - ARMCM4\_STM32F4/flash.h, [1479](#)
  - ARMCM4\_STM32L4/flash.c, [1384](#)
  - ARMCM4\_STM32L4/flash.h, [1483](#)
  - ARMCM4\_TM4C/flash.c, [1393](#)
  - ARMCM4\_TM4C/flash.h, [1486](#)
  - ARMCM4\_XMC4/flash.c, [1402](#)
  - ARMCM4\_XMC4/flash.h, [1490](#)
  - ARMCM7\_STM32F7/flash.c, [1411](#)
  - ARMCM7\_STM32F7/flash.h, [1494](#)
  - ARMCM7\_STM32H7/flash.c, [1419](#)
- ARMCM3\_STM32F1/flash.h, [1464](#)
- ARMCM3\_STM32F2/flash.c, [1351](#)
- ARMCM3\_STM32F2/flash.h, [1468](#)
- ARMCM4\_S32K14/flash.c, [1359](#)
- ARMCM4\_S32K14/flash.h, [1471](#)
- ARMCM4\_STM32F3/flash.c, [1366](#)
- ARMCM4\_STM32F3/flash.h, [1475](#)
- ARMCM4\_STM32F4/flash.c, [1374](#)
- ARMCM4\_STM32F4/flash.h, [1479](#)
- ARMCM4\_STM32L4/flash.c, [1384](#)
- ARMCM4\_STM32L4/flash.h, [1482](#)
- ARMCM4\_TM4C/flash.c, [1393](#)
- ARMCM4\_TM4C/flash.h, [1486](#)
- ARMCM4\_XMC4/flash.c, [1402](#)
- ARMCM4\_XMC4/flash.h, [1490](#)
- ARMCM7\_STM32F7/flash.c, [1411](#)
- ARMCM7\_STM32F7/flash.h, [1493](#)
- ARMCM7\_STM32H7/flash.c, [1419](#)
- ARMCM7\_STM32H7/flash.h, [1497](#)
- flash\_ecc.c, [1510](#)
- HCS12/flash.c, [1429](#)
- HCS12/flash.h, [1501](#)
- FlashWrite
  - \_template/flash.c, [1274](#)
  - \_template/flash.h, [1435](#)
  - ARMCM0\_S32K11/flash.c, [1283](#)
  - ARMCM0\_S32K11/flash.h, [1439](#)
  - ARMCM0\_STM32F0/flash.c, [1291](#)
  - ARMCM0\_STM32F0/flash.h, [1442](#)
  - ARMCM0\_STM32G0/flash.c, [1300](#)
  - ARMCM0\_STM32G0/flash.h, [1446](#)
  - ARMCM0\_XMC1/flash.c, [1308](#)
  - ARMCM0\_XMC1/flash.h, [1450](#)
  - ARMCM33\_STM32L5/flash.c, [1318](#)
  - ARMCM33\_STM32L5/flash.h, [1453](#)
  - ARMCM3\_EFM32/flash.c, [1327](#)
  - ARMCM3\_EFM32/flash.h, [1457](#)
  - ARMCM3\_LM3S/flash.c, [1336](#)
  - ARMCM3\_LM3S/flash.h, [1461](#)
  - ARMCM3\_STM32F1/flash.c, [1343](#)
  - ARMCM3\_STM32F1/flash.h, [1464](#)
  - ARMCM3\_STM32F2/flash.c, [1351](#)
  - ARMCM3\_STM32F2/flash.h, [1468](#)
  - ARMCM4\_S32K14/flash.c, [1360](#)
  - ARMCM4\_S32K14/flash.h, [1472](#)
  - ARMCM4\_STM32F3/flash.c, [1367](#)
  - ARMCM4\_STM32F3/flash.h, [1475](#)
  - ARMCM4\_STM32F4/flash.c, [1375](#)
  - ARMCM4\_STM32F4/flash.h, [1479](#)
  - ARMCM4\_STM32L4/flash.c, [1384](#)
  - ARMCM4\_STM32L4/flash.h, [1483](#)
  - ARMCM4\_TM4C/flash.c, [1393](#)
  - ARMCM4\_TM4C/flash.h, [1486](#)
  - ARMCM4\_XMC4/flash.c, [1402](#)
  - ARMCM4\_XMC4/flash.h, [1490](#)
  - ARMCM7\_STM32F7/flash.c, [1411](#)
  - ARMCM7\_STM32F7/flash.h, [1494](#)
  - ARMCM7\_STM32H7/flash.c, [1419](#)



- ARMCM7\_STM32H7/flash.h, [1497](#)
- flash\_ecc.c, [1510](#)
- HCS12/flash.c, [1429](#)
- HCS12/flash.h, [1501](#)
- FlashWriteBlock
  - \_template/flash.c, [1274](#)
  - ARMCM0\_S32K11/flash.c, [1283](#)
  - ARMCM0\_STM32F0/flash.c, [1291](#)
  - ARMCM0\_STM32G0/flash.c, [1300](#)
  - ARMCM0\_XMC1/flash.c, [1309](#)
  - ARMCM33\_STM32L5/flash.c, [1318](#)
  - ARMCM3\_EFM32/flash.c, [1327](#)
  - ARMCM3\_LM3S/flash.c, [1336](#)
  - ARMCM3\_STM32F1/flash.c, [1343](#)
  - ARMCM3\_STM32F2/flash.c, [1351](#)
  - ARMCM4\_S32K14/flash.c, [1360](#)
  - ARMCM4\_STM32F3/flash.c, [1367](#)
  - ARMCM4\_STM32F4/flash.c, [1375](#)
  - ARMCM4\_STM32L4/flash.c, [1384](#)
  - ARMCM4\_TM4C/flash.c, [1393](#)
  - ARMCM4\_XMC4/flash.c, [1403](#)
  - ARMCM7\_STM32F7/flash.c, [1411](#)
  - ARMCM7\_STM32H7/flash.c, [1420](#)
  - flash\_ecc.c, [1511](#)
  - HCS12/flash.c, [1430](#)
- FlashWriteChecksum
  - \_template/flash.c, [1274](#)
  - \_template/flash.h, [1435](#)
  - ARMCM0\_S32K11/flash.c, [1284](#)
  - ARMCM0\_S32K11/flash.h, [1439](#)
  - ARMCM0\_STM32F0/flash.c, [1292](#)
  - ARMCM0\_STM32F0/flash.h, [1443](#)
  - ARMCM0\_STM32G0/flash.c, [1301](#)
  - ARMCM0\_STM32G0/flash.h, [1446](#)
  - ARMCM0\_XMC1/flash.c, [1309](#)
  - ARMCM0\_XMC1/flash.h, [1450](#)
  - ARMCM33\_STM32L5/flash.c, [1319](#)
  - ARMCM33\_STM32L5/flash.h, [1454](#)
  - ARMCM3\_EFM32/flash.c, [1328](#)
  - ARMCM3\_EFM32/flash.h, [1457](#)
  - ARMCM3\_LM3S/flash.c, [1337](#)
  - ARMCM3\_LM3S/flash.h, [1461](#)
  - ARMCM3\_STM32F1/flash.c, [1344](#)
  - ARMCM3\_STM32F1/flash.h, [1465](#)
  - ARMCM3\_STM32F2/flash.c, [1352](#)
  - ARMCM3\_STM32F2/flash.h, [1468](#)
  - ARMCM4\_S32K14/flash.c, [1361](#)
  - ARMCM4\_S32K14/flash.h, [1472](#)
  - ARMCM4\_STM32F3/flash.c, [1368](#)
  - ARMCM4\_STM32F3/flash.h, [1476](#)
  - ARMCM4\_STM32F4/flash.c, [1376](#)
  - ARMCM4\_STM32F4/flash.h, [1479](#)
  - ARMCM4\_STM32L4/flash.c, [1385](#)
  - ARMCM4\_STM32L4/flash.h, [1483](#)
  - ARMCM4\_TM4C/flash.c, [1394](#)
  - ARMCM4\_TM4C/flash.h, [1487](#)
  - ARMCM4\_XMC4/flash.c, [1403](#)
  - ARMCM4\_XMC4/flash.h, [1490](#)
- ARMCM7\_STM32F7/flash.c, [1412](#)
- ARMCM7\_STM32F7/flash.h, [1494](#)
- ARMCM7\_STM32H7/flash.c, [1420](#)
- ARMCM7\_STM32H7/flash.h, [1498](#)
- flash\_ecc.c, [1511](#)
- HCS12/flash.c, [1430](#)
- HCS12/flash.h, [1502](#)
- fopt
  - tFlashRegs, [373](#)
- fprot
  - tFlashRegs, [373](#)
- frsv0
  - tFlashRegs, [374](#)
- frsv1
  - tFlashRegs, [374](#)
- frsv2
  - tFlashRegs, [374](#)
- frsv3
  - tFlashRegs, [374](#)
- frsv4
  - tFlashRegs, [374](#)
- frsv5
  - tFlashRegs, [374](#)
- frsv6
  - tFlashRegs, [374](#)
- frsv7
  - tFlashRegs, [375](#)
- fs
  - tFatFsObjects, [366](#)
- fsec
  - tFlashRegs, [375](#)
- fstat
  - tFlashRegs, [375](#)
- ftstmod
  - tFlashRegs, [375](#)
- func
  - tlsrFunc, [377](#)
- HAL\_GetTick
  - Source/ARMCM0\_STM32F0/timer.c, [3369](#)
  - Source/ARMCM0\_STM32G0/timer.c, [3372](#)
  - Source/ARMCM33\_STM32L5/timer.c, [3378](#)
  - Source/ARMCM3\_STM32F1/timer.c, [3386](#)
  - Source/ARMCM3\_STM32F2/timer.c, [3389](#)
  - Source/ARMCM4\_STM32F3/timer.c, [3394](#)
  - Source/ARMCM4\_STM32F4/timer.c, [3397](#)
  - Source/ARMCM4\_STM32L4/timer.c, [3400](#)
  - Source/ARMCM7\_STM32F7/timer.c, [3408](#)
  - Source/ARMCM7\_STM32H7/timer.c, [3411](#)
- HAL\_MspDeInit
  - ARMCM0\_STM32F0\_Discovery\_STM32F051\_↔  
GCC/Boot/main.c, [2609](#)
  - ARMCM0\_STM32F0\_Discovery\_STM32F051\_↔  
GCC/Prog/main.c, [2611](#)
  - ARMCM0\_STM32F0\_Discovery\_STM32F051\_I↔  
AR/Boot/main.c, [2614](#)
  - ARMCM0\_STM32F0\_Discovery\_STM32F051\_I↔  
AR/Prog/main.c, [2616](#)

- ARMCM0\_STM32F0\_Discovery\_STM32F051\_↔  
Keil/Boot/main.c, [2618](#)
- ARMCM0\_STM32F0\_Discovery\_STM32F051\_↔  
Keil/Prog/main.c, [2621](#)
- ARMCM0\_STM32F0\_Nucleo\_F091RC\_GCC/↔  
Boot/main.c, [2623](#)
- ARMCM0\_STM32F0\_Nucleo\_F091RC\_GCC/↔  
Prog/main.c, [2625](#)
- ARMCM0\_STM32F0\_Nucleo\_F091RC\_IAR/↔  
Boot/main.c, [2628](#)
- ARMCM0\_STM32F0\_Nucleo\_F091RC\_IAR/↔  
Prog/main.c, [2630](#)
- ARMCM0\_STM32F0\_Nucleo\_F091RC\_Keil/↔  
Boot/main.c, [2632](#)
- ARMCM0\_STM32F0\_Nucleo\_F091RC\_Keil/↔  
Prog/main.c, [2635](#)
- ARMCM0\_STM32G0\_Nucleo\_G071RB\_GCC/↔  
Boot/main.c, [2637](#)
- ARMCM0\_STM32G0\_Nucleo\_G071RB\_GCC/↔  
Prog/main.c, [2639](#)
- ARMCM0\_STM32G0\_Nucleo\_G071RB\_IAR/↔  
Boot/main.c, [2642](#)
- ARMCM0\_STM32G0\_Nucleo\_G071RB\_IAR/↔  
Prog/main.c, [2644](#)
- ARMCM0\_STM32G0\_Nucleo\_G071RB\_Keil/↔  
Boot/main.c, [2647](#)
- ARMCM0\_STM32G0\_Nucleo\_G071RB\_Keil/↔  
Prog/main.c, [2649](#)
- ARMCM33\_STM32L5\_Nucleo\_L552ZE\_GCC/↔  
Boot/main.c, [2658](#)
- ARMCM33\_STM32L5\_Nucleo\_L552ZE\_GCC/↔  
Prog/main.c, [2660](#)
- ARMCM33\_STM32L5\_Nucleo\_L552ZE\_IAR/↔  
Boot/main.c, [2663](#)
- ARMCM33\_STM32L5\_Nucleo\_L552ZE\_IAR/↔  
Prog/main.c, [2665](#)
- ARMCM33\_STM32L5\_Nucleo\_L552ZE\_Keil/↔  
Boot/main.c, [2668](#)
- ARMCM33\_STM32L5\_Nucleo\_L552ZE\_Keil/↔  
Prog/main.c, [2670](#)
- ARMCM3\_STM32F1\_Nucleo\_F103RB\_GCC/↔  
Boot/main.c, [2688](#)
- ARMCM3\_STM32F1\_Nucleo\_F103RB\_GCC/↔  
Prog/main.c, [2690](#)
- ARMCM3\_STM32F1\_Nucleo\_F103RB\_IAR/↔  
Boot/main.c, [2693](#)
- ARMCM3\_STM32F1\_Nucleo\_F103RB\_IAR/↔  
Prog/main.c, [2695](#)
- ARMCM3\_STM32F1\_Nucleo\_F103RB\_Keil/↔  
Boot/main.c, [2698](#)
- ARMCM3\_STM32F1\_Nucleo\_F103RB\_Keil/↔  
Prog/main.c, [2700](#)
- ARMCM3\_STM32F1\_Olimex\_STM32H103\_GC↔  
C/Boot/main.c, [2703](#)
- ARMCM3\_STM32F1\_Olimex\_STM32H103\_GC↔  
C/Prog/main.c, [2705](#)
- ARMCM3\_STM32F1\_Olimex\_STM32H103\_IA↔  
R/Boot/main.c, [2708](#)
- ARMCM3\_STM32F1\_Olimex\_STM32H103\_IA↔  
R/Prog/main.c, [2710](#)
- ARMCM3\_STM32F1\_Olimex\_STM32H103\_↔  
Keil/Boot/main.c, [2713](#)
- ARMCM3\_STM32F1\_Olimex\_STM32H103\_↔  
Keil/Prog/main.c, [2715](#)
- ARMCM3\_STM32F1\_Olimex\_STM32P103\_GC↔  
C/Boot/main.c, [2718](#)
- ARMCM3\_STM32F1\_Olimex\_STM32P103\_GC↔  
C/Prog/main.c, [2720](#)
- ARMCM3\_STM32F1\_Olimex\_STM32P103\_IA↔  
R/Boot/main.c, [2723](#)
- ARMCM3\_STM32F1\_Olimex\_STM32P103\_IA↔  
R/Prog/main.c, [2725](#)
- ARMCM3\_STM32F1\_Olimex\_STM32P103\_Keil/↔  
Boot/main.c, [2728](#)
- ARMCM3\_STM32F1\_Olimex\_STM32P103\_Keil/↔  
Prog/main.c, [2730](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_GCC/↔  
Boot/main.c, [2733](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_GCC/↔  
Prog/main.c, [2735](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_IAR/↔  
Boot/main.c, [2738](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_IAR/↔  
Prog/main.c, [2740](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_Keil/↔  
Boot/main.c, [2743](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_Keil/↔  
Prog/main.c, [2745](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_GC↔  
C/Boot/main.c, [2748](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_GC↔  
C/Prog/main.c, [2750](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_IA↔  
R/Boot/main.c, [2753](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_IA↔  
R/Prog/main.c, [2755](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_Keil/↔  
Boot/main.c, [2758](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_Keil/↔  
Prog/main.c, [2760](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_GCC/↔  
Boot/main.c, [2770](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_GCC/↔  
Prog/main.c, [2773](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_IAR/↔  
Boot/main.c, [2775](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_IAR/↔  
Prog/main.c, [2778](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_Keil/↔  
Boot/main.c, [2781](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_Keil/↔  
Prog/main.c, [2783](#)
- ARMCM4\_STM32F3\_Nucleo\_F303K8\_GCC/↔  
Boot/main.c, [2786](#)
- ARMCM4\_STM32F3\_Nucleo\_F303K8\_GCC/↔  
Prog/main.c, [2788](#)

- ARMCM4\_STM32F3\_Nucleo\_F303K8\_IAR/↔  
Boot/main.c, [2791](#)
- ARMCM4\_STM32F3\_Nucleo\_F303K8\_IAR/↔  
Prog/main.c, [2793](#)
- ARMCM4\_STM32F3\_Nucleo\_F303K8\_Keil/↔  
Boot/main.c, [2796](#)
- ARMCM4\_STM32F3\_Nucleo\_F303K8\_Keil/↔  
Prog/main.c, [2798](#)
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_GCC/↔  
Boot/main.c, [2801](#)
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_GCC/↔  
Prog/main.c, [2803](#)
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_IAR/↔  
Boot/main.c, [2806](#)
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_IAR/↔  
Prog/main.c, [2808](#)
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_Keil/↔  
Boot/main.c, [2811](#)
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_Keil/↔  
Prog/main.c, [2813](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_GC↔  
C/Boot/main.c, [2816](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_GC↔  
C/Prog/main.c, [2818](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_IA↔  
R/Boot/main.c, [2821](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_IA↔  
R/Prog/main.c, [2823](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_Keil/↔  
Boot/main.c, [2826](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_Keil/↔  
Prog/main.c, [2828](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_GCC/↔  
Boot/main.c, [2831](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_GCC/↔  
Prog/main.c, [2833](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_IAR/↔  
Boot/main.c, [2836](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_IAR/↔  
Prog/main.c, [2838](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_Keil/↔  
Boot/main.c, [2841](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_Keil/↔  
Prog/main.c, [2843](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_GCC/↔  
Boot/main.c, [2854](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_GCC/↔  
Prog/main.c, [2856](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_IAR/↔  
Boot/main.c, [2859](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_IAR/↔  
Prog/main.c, [2861](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_Keil/↔  
Boot/main.c, [2864](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_Keil/↔  
Prog/main.c, [2866](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_GCC/↔  
Boot/main.c, [2869](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_GCC/↔  
Prog/main.c, [2871](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_IAR/↔  
Boot/main.c, [2874](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_IAR/↔  
Prog/main.c, [2876](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_Keil/↔  
Boot/main.c, [2879](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_Keil/↔  
Prog/main.c, [2881](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_GCC/↔  
Boot/main.c, [2884](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_GCC/↔  
Prog/main.c, [2886](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_IAR/↔  
Boot/main.c, [2889](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_IAR/↔  
Prog/main.c, [2891](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_Keil/↔  
Boot/main.c, [2894](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_Keil/↔  
Prog/main.c, [2896](#)
- HAL\_Msplnit
- ARMCM0\_STM32F0\_Discovery\_STM32F051\_↔  
GCC/Boot/main.c, [2609](#)
- ARMCM0\_STM32F0\_Discovery\_STM32F051\_↔  
GCC/Prog/main.c, [2612](#)
- ARMCM0\_STM32F0\_Discovery\_STM32F051\_I↔  
AR/Boot/main.c, [2614](#)
- ARMCM0\_STM32F0\_Discovery\_STM32F051\_I↔  
AR/Prog/main.c, [2616](#)
- ARMCM0\_STM32F0\_Discovery\_STM32F051\_↔  
Keil/Boot/main.c, [2619](#)
- ARMCM0\_STM32F0\_Discovery\_STM32F051\_↔  
Keil/Prog/main.c, [2621](#)
- ARMCM0\_STM32F0\_Nucleo\_F091RC\_GCC/↔  
Boot/main.c, [2623](#)
- ARMCM0\_STM32F0\_Nucleo\_F091RC\_GCC/↔  
Prog/main.c, [2626](#)
- ARMCM0\_STM32F0\_Nucleo\_F091RC\_IAR/↔  
Boot/main.c, [2628](#)
- ARMCM0\_STM32F0\_Nucleo\_F091RC\_IAR/↔  
Prog/main.c, [2630](#)
- ARMCM0\_STM32F0\_Nucleo\_F091RC\_Keil/↔  
Boot/main.c, [2633](#)
- ARMCM0\_STM32F0\_Nucleo\_F091RC\_Keil/↔  
Prog/main.c, [2635](#)
- ARMCM0\_STM32G0\_Nucleo\_G071RB\_GCC/↔  
Boot/main.c, [2637](#)
- ARMCM0\_STM32G0\_Nucleo\_G071RB\_GCC/↔  
Prog/main.c, [2640](#)
- ARMCM0\_STM32G0\_Nucleo\_G071RB\_IAR/↔  
Boot/main.c, [2642](#)
- ARMCM0\_STM32G0\_Nucleo\_G071RB\_IAR/↔  
Prog/main.c, [2645](#)
- ARMCM0\_STM32G0\_Nucleo\_G071RB\_Keil/↔  
Boot/main.c, [2647](#)
- ARMCM0\_STM32G0\_Nucleo\_G071RB\_Keil/↔

- Prog/main.c, [2650](#)
- ARMCM33\_STM32L5\_Nucleo\_L552ZE\_GCC/↔  
Boot/main.c, [2658](#)
- ARMCM33\_STM32L5\_Nucleo\_L552ZE\_GCC/↔  
Prog/main.c, [2660](#)
- ARMCM33\_STM32L5\_Nucleo\_L552ZE\_IAR/↔  
Boot/main.c, [2663](#)
- ARMCM33\_STM32L5\_Nucleo\_L552ZE\_IAR/↔  
Prog/main.c, [2665](#)
- ARMCM33\_STM32L5\_Nucleo\_L552ZE\_Keil/↔  
Boot/main.c, [2668](#)
- ARMCM33\_STM32L5\_Nucleo\_L552ZE\_Keil/↔  
Prog/main.c, [2670](#)
- ARMCM3\_STM32F1\_Nucleo\_F103RB\_GCC/↔  
Boot/main.c, [2688](#)
- ARMCM3\_STM32F1\_Nucleo\_F103RB\_GCC/↔  
Prog/main.c, [2691](#)
- ARMCM3\_STM32F1\_Nucleo\_F103RB\_IAR/↔  
Boot/main.c, [2693](#)
- ARMCM3\_STM32F1\_Nucleo\_F103RB\_IAR/↔  
Prog/main.c, [2696](#)
- ARMCM3\_STM32F1\_Nucleo\_F103RB\_Keil/↔  
Boot/main.c, [2698](#)
- ARMCM3\_STM32F1\_Nucleo\_F103RB\_Keil/↔  
Prog/main.c, [2701](#)
- ARMCM3\_STM32F1\_Olimex\_STM32H103\_GC↔  
C/Boot/main.c, [2703](#)
- ARMCM3\_STM32F1\_Olimex\_STM32H103\_GC↔  
C/Prog/main.c, [2706](#)
- ARMCM3\_STM32F1\_Olimex\_STM32H103\_IA↔  
R/Boot/main.c, [2708](#)
- ARMCM3\_STM32F1\_Olimex\_STM32H103\_IA↔  
R/Prog/main.c, [2711](#)
- ARMCM3\_STM32F1\_Olimex\_STM32H103\_↔  
Keil/Boot/main.c, [2713](#)
- ARMCM3\_STM32F1\_Olimex\_STM32H103\_↔  
Keil/Prog/main.c, [2716](#)
- ARMCM3\_STM32F1\_Olimex\_STM32P103\_GC↔  
C/Boot/main.c, [2718](#)
- ARMCM3\_STM32F1\_Olimex\_STM32P103\_GC↔  
C/Prog/main.c, [2721](#)
- ARMCM3\_STM32F1\_Olimex\_STM32P103\_IA↔  
R/Boot/main.c, [2723](#)
- ARMCM3\_STM32F1\_Olimex\_STM32P103\_IA↔  
R/Prog/main.c, [2726](#)
- ARMCM3\_STM32F1\_Olimex\_STM32P103\_Keil/↔  
Boot/main.c, [2728](#)
- ARMCM3\_STM32F1\_Olimex\_STM32P103\_Keil/↔  
Prog/main.c, [2731](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_GCC/↔  
Boot/main.c, [2733](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_GCC/↔  
Prog/main.c, [2736](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_IAR/↔  
Boot/main.c, [2738](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_IAR/↔  
Prog/main.c, [2741](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_Keil/↔  
Boot/main.c, [2743](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_Keil/↔  
Prog/main.c, [2746](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_GC↔  
C/Boot/main.c, [2748](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_GC↔  
C/Prog/main.c, [2751](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_IA↔  
R/Boot/main.c, [2753](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_IA↔  
R/Prog/main.c, [2756](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_Keil/↔  
Boot/main.c, [2758](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_Keil/↔  
Prog/main.c, [2761](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_GCC/↔  
Boot/main.c, [2770](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_GCC/↔  
Prog/main.c, [2773](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_IAR/↔  
Boot/main.c, [2776](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_IAR/↔  
Prog/main.c, [2778](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_Keil/↔  
Boot/main.c, [2781](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_Keil/↔  
Prog/main.c, [2783](#)
- ARMCM4\_STM32F3\_Nucleo\_F303K8\_GCC/↔  
Boot/main.c, [2786](#)
- ARMCM4\_STM32F3\_Nucleo\_F303K8\_GCC/↔  
Prog/main.c, [2788](#)
- ARMCM4\_STM32F3\_Nucleo\_F303K8\_IAR/↔  
Boot/main.c, [2791](#)
- ARMCM4\_STM32F3\_Nucleo\_F303K8\_IAR/↔  
Prog/main.c, [2793](#)
- ARMCM4\_STM32F3\_Nucleo\_F303K8\_Keil/↔  
Boot/main.c, [2796](#)
- ARMCM4\_STM32F3\_Nucleo\_F303K8\_Keil/↔  
Prog/main.c, [2798](#)
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_GCC/↔  
Boot/main.c, [2801](#)
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_GCC/↔  
Prog/main.c, [2803](#)
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_IAR/↔  
Boot/main.c, [2806](#)
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_IAR/↔  
Prog/main.c, [2808](#)
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_Keil/↔  
Boot/main.c, [2811](#)
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_Keil/↔  
Prog/main.c, [2813](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_GC↔  
C/Boot/main.c, [2816](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_GC↔  
C/Prog/main.c, [2818](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_IA↔  
R/Boot/main.c, [2821](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_IA↔

- R/Prog/main.c, [2823](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_Keil/↔
  - Boot/main.c, [2826](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_Keil/↔
  - Prog/main.c, [2828](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_GCC/↔
  - Boot/main.c, [2831](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_GCC/↔
  - Prog/main.c, [2833](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_IAR/↔
  - Boot/main.c, [2836](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_IAR/↔
  - Prog/main.c, [2838](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_Keil/↔
  - Boot/main.c, [2841](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_Keil/↔
  - Prog/main.c, [2843](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_GCC/↔
  - Boot/main.c, [2854](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_GCC/↔
  - Prog/main.c, [2857](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_IAR/↔
  - Boot/main.c, [2859](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_IAR/↔
  - Prog/main.c, [2862](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_Keil/↔
  - Boot/main.c, [2864](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_Keil/↔
  - Prog/main.c, [2867](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_GCC/↔
  - Boot/main.c, [2869](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_GCC/↔
  - Prog/main.c, [2872](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_IAR/↔
  - Boot/main.c, [2874](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_IAR/↔
  - Prog/main.c, [2877](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_Keil/↔
  - Boot/main.c, [2879](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_Keil/↔
  - Prog/main.c, [2882](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_GCC/↔
  - Boot/main.c, [2884](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_GCC/↔
  - Prog/main.c, [2887](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_IAR/↔
  - Boot/main.c, [2889](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_IAR/↔
  - Prog/main.c, [2892](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_Keil/↔
  - Boot/main.c, [2894](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_Keil/↔
  - Prog/main.c, [2897](#)
- HCS12/CodeWarrior/cpu\_comp.c
  - CpuIrqDisable, [1246](#)
  - CpuIrqEnable, [1246](#)
- HCS12/can.c
  - CanGetSpeedConfig, [1117](#)
  - CanInit, [1117](#)
  - CanReceivePacket, [1117](#)
  - canTiming, [1118](#)
  - CanTransmitPacket, [1118](#)
- HCS12/cpu.c
  - CPU\_USER\_PROGRAM\_STARTADDR\_PTR, [1185](#)
  - CpuInit, [1186](#)
  - CpuMemCopy, [1186](#)
  - CpuMemSet, [1186](#)
  - CpuStartUserProgram, [1187](#)
  - CpuUserProgramStartHook, [1187](#)
- HCS12/flash.c
  - blockInfo, [1431](#)
  - bootBlockInfo, [1431](#)
  - FlashAddToBlock, [1425](#)
  - FlashDone, [1425](#)
  - FlashErase, [1425](#)
  - flashExecCmd, [1431](#)
  - FlashExecuteCommand, [1426](#)
  - FlashGetLinearAddrByte, [1426](#)
  - FlashGetPhysAddr, [1426](#)
  - FlashGetPhysPage, [1427](#)
  - FlashGetUserProgBaseAddress, [1427](#)
  - FlashInit, [1427](#)
  - FlashInitBlock, [1428](#)
  - flashLayout, [1432](#)
  - FlashOperate, [1428](#)
  - FlashReinit, [1429](#)
  - FlashSwitchBlock, [1429](#)
  - FlashVerifyChecksum, [1429](#)
  - FlashWrite, [1429](#)
  - FlashWriteBlock, [1430](#)
  - FlashWriteChecksum, [1430](#)
- HCS12/flash.h
  - FlashDone, [1499](#)
  - FlashErase, [1499](#)
  - FlashGetUserProgBaseAddress, [1500](#)
  - FlashInit, [1500](#)
  - FlashReinit, [1501](#)
  - FlashVerifyChecksum, [1501](#)
  - FlashWrite, [1501](#)
  - FlashWriteChecksum, [1502](#)
- HCS12/nvm.c
  - NvmDone, [3051](#)
  - NvmDoneHook, [3051](#)
  - NvmErase, [3051](#)
  - NvmEraseHook, [3052](#)
  - NvmGetUserProgBaseAddress, [3052](#)
  - NvmInit, [3052](#)
  - NvmInitHook, [3053](#)
  - NvmReinit, [3053](#)
  - NvmReinitHook, [3053](#)
  - NvmVerifyChecksum, [3053](#)
  - NvmWrite, [3054](#)
  - NvmWriteHook, [3054](#)
- HCS12/types.h
  - blt\_addr, [3566](#)



- blt\_bool, [3566](#)
- blt\_char, [3566](#)
- blt\_int16s, [3566](#)
- blt\_int16u, [3567](#)
- blt\_int32s, [3567](#)
- blt\_int32u, [3567](#)
- blt\_int8s, [3567](#)
- blt\_int8u, [3567](#)
- HCS12\_DevKit\_S12G128\_CodeWarrior/Boot/hooks.c
  - BackDoorEntryHook, [2123](#)
  - BackDoorInitHook, [2123](#)
  - CopInitHook, [2123](#)
  - CopServiceHook, [2123](#)
  - CpuUserProgramStartHook, [2124](#)
  - NvmDoneHook, [2124](#)
  - NvmEraseHook, [2124](#)
  - NvmInitHook, [2125](#)
  - NvmReinitHook, [2125](#)
  - NvmWriteHook, [2125](#)
  - XcpGetSeedHook, [2126](#)
  - XcpVerifyKeyHook, [2126](#)
- HCS12\_DevKit\_S12G128\_CodeWarrior/Boot/led.c
  - LedBlinkExit, [2368](#)
  - LedBlinkInit, [2368](#)
  - LedBlinkTask, [2369](#)
- HCS12\_DevKit\_S12G128\_CodeWarrior/Boot/led.h
  - LedBlinkExit, [2602](#)
  - LedBlinkInit, [2602](#)
  - LedBlinkTask, [2603](#)
- HCS12\_DevKit\_S12G128\_CodeWarrior/Boot/main.c
  - Init, [2899](#)
  - main, [2899](#)
  - SystemClockInit, [2899](#)
- HCS12\_DevKit\_S12G128\_CodeWarrior/Boot/vectors.c↵
  - VCT\_USER\_PROGRAM\_VECTOR\_TABLE\_ST↵
    - ARTADDR, [3647](#)
  - Vector0\_handler, [3647](#)
  - Vector10\_handler, [3647](#)
  - Vector11\_handler, [3648](#)
  - Vector12\_handler, [3648](#)
  - Vector13\_handler, [3648](#)
  - Vector14\_handler, [3648](#)
  - Vector15\_handler, [3649](#)
  - Vector16\_handler, [3649](#)
  - Vector17\_handler, [3649](#)
  - Vector18\_handler, [3649](#)
  - Vector19\_handler, [3650](#)
  - Vector1\_handler, [3650](#)
  - Vector20\_handler, [3650](#)
  - Vector21\_handler, [3650](#)
  - Vector22\_handler, [3651](#)
  - Vector23\_handler, [3651](#)
  - Vector24\_handler, [3651](#)
  - Vector25\_handler, [3651](#)
  - Vector26\_handler, [3652](#)
  - Vector27\_handler, [3652](#)
  - Vector28\_handler, [3652](#)
  - Vector29\_handler, [3652](#)
  - Vector2\_handler, [3653](#)
  - Vector30\_handler, [3653](#)
  - Vector31\_handler, [3653](#)
  - Vector32\_handler, [3653](#)
  - Vector33\_handler, [3654](#)
  - Vector34\_handler, [3654](#)
  - Vector35\_handler, [3654](#)
  - Vector36\_handler, [3654](#)
  - Vector37\_handler, [3655](#)
  - Vector38\_handler, [3655](#)
  - Vector39\_handler, [3655](#)
  - Vector3\_handler, [3655](#)
  - Vector40\_handler, [3656](#)
  - Vector41\_handler, [3656](#)
  - Vector42\_handler, [3656](#)
  - Vector43\_handler, [3656](#)
  - Vector44\_handler, [3657](#)
  - Vector45\_handler, [3657](#)
  - Vector46\_handler, [3657](#)
  - Vector47\_handler, [3657](#)
  - Vector48\_handler, [3658](#)
  - Vector49\_handler, [3658](#)
  - Vector4\_handler, [3658](#)
  - Vector50\_handler, [3658](#)
  - Vector51\_handler, [3659](#)
  - Vector52\_handler, [3659](#)
  - Vector53\_handler, [3659](#)
  - Vector54\_handler, [3659](#)
  - Vector55\_handler, [3660](#)
  - Vector56\_handler, [3660](#)
  - Vector57\_handler, [3660](#)
  - Vector58\_handler, [3660](#)
  - Vector59\_handler, [3661](#)
  - Vector5\_handler, [3661](#)
  - Vector60\_handler, [3661](#)
  - Vector61\_handler, [3661](#)
  - Vector62\_handler, [3662](#)
  - Vector6\_handler, [3662](#)
  - Vector7\_handler, [3662](#)
  - Vector8\_handler, [3662](#)
  - Vector9\_handler, [3663](#)
- HCS12\_DevKit\_S12G128\_CodeWarrior/Prog/header.h
  - BDM\_DEBUGGING\_ENABLED, [1618](#)
- HCS12\_DevKit\_S12G128\_CodeWarrior/Prog/led.c
  - LedInit, [2370](#)
  - LedToggle, [2370](#)
- HCS12\_DevKit\_S12G128\_CodeWarrior/Prog/led.h
  - LedInit, [2604](#)
  - LedToggle, [2604](#)
- HCS12\_DevKit\_S12G128\_CodeWarrior/Prog/main.c
  - Init, [2901](#)
  - main, [2901](#)
  - SystemClockInit, [2901](#)
- HCS12\_DevKit\_S12G128\_CodeWarrior/Prog/vectors.c↵
  - \_vectab, [3664](#)

- HCS12\_Evbplus\_Dragon12p\_CodeWarrior/Boot/hooks.↵
  - c
  - BackDoorEntryHook, [2128](#)
  - BackDoorInitHook, [2128](#)
  - CopInitHook, [2128](#)
  - CopServiceHook, [2129](#)
  - CpuUserProgramStartHook, [2129](#)
  - NvmDoneHook, [2129](#)
  - NvmEraseHook, [2130](#)
  - NvmInitHook, [2130](#)
  - NvmReinitHook, [2130](#)
  - NvmWriteHook, [2131](#)
  - XcpGetSeedHook, [2131](#)
  - XcpVerifyKeyHook, [2132](#)
- HCS12\_Evbplus\_Dragon12p\_CodeWarrior/Boot/main.↵
  - c
  - Init, [2902](#)
  - main, [2902](#)
- HCS12\_Evbplus\_Dragon12p\_CodeWarrior/Boot/vectors.↵
  - c
  - main, [3669](#)
  - reset\_handler, [3670](#)
  - VCT\_USER\_PROGRAM\_VECTOR\_TABLE\_ST↵
    - ARTADDR, [3669](#)
  - Vector0\_handler, [3670](#)
  - Vector10\_handler, [3670](#)
  - Vector11\_handler, [3671](#)
  - Vector12\_handler, [3671](#)
  - Vector13\_handler, [3671](#)
  - Vector14\_handler, [3671](#)
  - Vector15\_handler, [3672](#)
  - Vector16\_handler, [3672](#)
  - Vector17\_handler, [3672](#)
  - Vector18\_handler, [3672](#)
  - Vector19\_handler, [3673](#)
  - Vector1\_handler, [3673](#)
  - Vector20\_handler, [3673](#)
  - Vector21\_handler, [3673](#)
  - Vector22\_handler, [3674](#)
  - Vector23\_handler, [3674](#)
  - Vector24\_handler, [3674](#)
  - Vector25\_handler, [3674](#)
  - Vector26\_handler, [3675](#)
  - Vector27\_handler, [3675](#)
  - Vector28\_handler, [3675](#)
  - Vector29\_handler, [3675](#)
  - Vector2\_handler, [3676](#)
  - Vector30\_handler, [3676](#)
  - Vector31\_handler, [3676](#)
  - Vector32\_handler, [3676](#)
  - Vector33\_handler, [3677](#)
  - Vector34\_handler, [3677](#)
  - Vector35\_handler, [3677](#)
  - Vector36\_handler, [3677](#)
  - Vector37\_handler, [3678](#)
  - Vector38\_handler, [3678](#)
  - Vector39\_handler, [3678](#)
  - Vector3\_handler, [3678](#)
  - Vector40\_handler, [3679](#)
  - Vector41\_handler, [3679](#)
  - Vector42\_handler, [3679](#)
  - Vector43\_handler, [3679](#)
  - Vector44\_handler, [3680](#)
  - Vector45\_handler, [3680](#)
  - Vector46\_handler, [3680](#)
  - Vector47\_handler, [3680](#)
  - Vector48\_handler, [3681](#)
  - Vector49\_handler, [3681](#)
  - Vector4\_handler, [3681](#)
  - Vector50\_handler, [3681](#)
  - Vector51\_handler, [3682](#)
  - Vector52\_handler, [3682](#)
  - Vector53\_handler, [3682](#)
  - Vector54\_handler, [3682](#)
  - Vector55\_handler, [3683](#)
  - Vector56\_handler, [3683](#)
  - Vector57\_handler, [3683](#)
  - Vector58\_handler, [3683](#)
  - Vector59\_handler, [3684](#)
  - Vector5\_handler, [3684](#)
  - Vector60\_handler, [3684](#)
  - Vector61\_handler, [3684](#)
  - Vector62\_handler, [3685](#)
  - Vector6\_handler, [3685](#)
  - Vector7\_handler, [3685](#)
  - Vector8\_handler, [3685](#)
  - Vector9\_handler, [3686](#)
- HCS12\_Evbplus\_Dragon12p\_CodeWarrior/Prog/header.↵
  - h
  - BDM\_DEBUGGING\_ENABLED, [1619](#)
- HCS12\_Evbplus\_Dragon12p\_CodeWarrior/Prog/led.c
  - LedInit, [2371](#)
  - LedToggle, [2371](#)
- HCS12\_Evbplus\_Dragon12p\_CodeWarrior/Prog/led.h
  - LedInit, [2605](#)
  - LedToggle, [2605](#)
- HCS12\_Evbplus\_Dragon12p\_CodeWarrior/Prog/main.↵
  - c
  - Init, [2904](#)
  - main, [2904](#)
  - SysClockInit, [2904](#)
- HCS12\_Evbplus\_Dragon12p\_CodeWarrior/Prog/vectors.↵
  - c
  - \_vectab, [3687](#)
- handle
  - ARMCM3\_LM3S\_EK\_LM3S6965\_GCC/Boot/hooks.↵
    - c, [1736](#)
  - ARMCM3\_LM3S\_EK\_LM3S6965\_IAR/Boot/hooks.↵
    - c, [1744](#)
  - ARMCM3\_STM32F1\_Olimex\_STM32P103\_↵
    - CubeIDE/Boot/App/hooks.c, [1806](#)
  - ARMCM3\_STM32F1\_Olimex\_STM32P103\_GC↵
    - C/Boot/hooks.c, [1813](#)
  - ARMCM3\_STM32F1\_Olimex\_STM32P103\_IA↵
    - R/Boot/hooks.c, [1821](#)

- ARMCM3\_STM32F1\_Olimex\_STM32P103\_Keil/↔  
Boot/hooks.c, [1829](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_Cubel↔  
DE/Boot/App/hooks.c, [1837](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_GCC/↔  
Boot/hooks.c, [1846](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_IAR/↔  
Boot/hooks.c, [1855](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_Keil/↔  
Boot/hooks.c, [1864](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_↔  
CubelDE/Boot/App/hooks.c, [1872](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_GC↔  
C/Boot/hooks.c, [1879](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_IA↔  
R/Boot/hooks.c, [1887](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_Keil/↔  
Boot/hooks.c, [1895](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_↔  
CubelDE/Boot/App/hooks.c, [1981](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_GC↔  
C/Boot/hooks.c, [1990](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_IA↔  
R/Boot/hooks.c, [1999](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_Keil/↔  
Boot/hooks.c, [2008](#)
- ARMCM4\_TM4C\_DK\_TM4C123G\_IAR/Boot/hooks.↔  
c, [2037](#)
- ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_GCC/↔  
Boot/hooks.c, [2045](#)
- ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_IAR/↔  
Boot/hooks.c, [2053](#)
- tFifoCtrl, [367](#)
- tFifoPipe, [369](#)
- header.h, [1535–1617](#), [1619](#)
- hooks.c, [1620](#), [1625](#), [1630](#), [1635](#), [1640](#), [1645](#), [1650](#),  
[1655](#), [1660](#), [1665](#), [1670](#), [1675](#), [1680](#), [1685](#),  
[1690](#), [1695](#), [1701](#), [1706](#), [1712](#), [1718](#), [1724](#),  
[1729](#), [1737](#), [1744](#), [1749](#), [1755](#), [1760](#), [1765](#),  
[1770](#), [1775](#), [1780](#), [1786](#), [1792](#), [1798](#), [1806](#),  
[1814](#), [1822](#), [1829](#), [1838](#), [1847](#), [1856](#), [1865](#),  
[1872](#), [1880](#), [1888](#), [1895](#), [1900](#), [1906](#), [1911](#),  
[1917](#), [1923](#), [1929](#), [1935](#), [1940](#), [1945](#), [1950](#),  
[1955](#), [1961](#), [1967](#), [1973](#), [1982](#), [1991](#), [2000](#),  
[2009](#), [2014](#), [2019](#), [2024](#), [2029](#), [2038](#), [2045](#),  
[2053](#), [2059](#), [2065](#), [2071](#), [2077](#), [2083](#), [2088](#),  
[2093](#), [2098](#), [2103](#), [2109](#), [2115](#), [2121](#), [2127](#)
- idr
- tCanRxMsgSlot, [364](#)
- tCanTxMsgSlot, [365](#)
- Init
- \_template/Boot/main.c, [2606](#)
- \_template/Prog/main.c, [2608](#)
- ARMCM0\_STM32F0\_Discovery\_STM32F051\_↔  
GCC/Boot/main.c, [2609](#)
- ARMCM0\_STM32F0\_Discovery\_STM32F051\_↔  
GCC/Prog/main.c, [2612](#)
- ARMCM0\_STM32F0\_Discovery\_STM32F051\_I↔  
AR/Boot/main.c, [2614](#)
- ARMCM0\_STM32F0\_Discovery\_STM32F051\_I↔  
AR/Prog/main.c, [2616](#)
- ARMCM0\_STM32F0\_Discovery\_STM32F051\_↔  
Keil/Boot/main.c, [2619](#)
- ARMCM0\_STM32F0\_Discovery\_STM32F051\_↔  
Keil/Prog/main.c, [2621](#)
- ARMCM0\_STM32F0\_Nucleo\_F091RC\_GCC/↔  
Boot/main.c, [2623](#)
- ARMCM0\_STM32F0\_Nucleo\_F091RC\_GCC/↔  
Prog/main.c, [2626](#)
- ARMCM0\_STM32F0\_Nucleo\_F091RC\_IAR/↔  
Boot/main.c, [2628](#)
- ARMCM0\_STM32F0\_Nucleo\_F091RC\_IAR/↔  
Prog/main.c, [2630](#)
- ARMCM0\_STM32F0\_Nucleo\_F091RC\_Keil/↔  
Boot/main.c, [2633](#)
- ARMCM0\_STM32F0\_Nucleo\_F091RC\_Keil/↔  
Prog/main.c, [2635](#)
- ARMCM0\_STM32G0\_Nucleo\_G071RB\_GCC/↔  
Boot/main.c, [2637](#)
- ARMCM0\_STM32G0\_Nucleo\_G071RB\_GCC/↔  
Prog/main.c, [2640](#)
- ARMCM0\_STM32G0\_Nucleo\_G071RB\_IAR/↔  
Boot/main.c, [2642](#)
- ARMCM0\_STM32G0\_Nucleo\_G071RB\_IAR/↔  
Prog/main.c, [2645](#)
- ARMCM0\_STM32G0\_Nucleo\_G071RB\_Keil/↔  
Boot/main.c, [2647](#)
- ARMCM0\_STM32G0\_Nucleo\_G071RB\_Keil/↔  
Prog/main.c, [2650](#)
- ARMCM0\_XMC1\_XMC1400\_Boot\_Kit\_GCC/↔  
Prog/main.c, [2653](#)
- ARMCM0\_XMC1\_XMC1400\_Boot\_Kit\_IAR/↔  
Boot/main.c, [2655](#)
- ARMCM0\_XMC1\_XMC1400\_Boot\_Kit\_IAR/↔  
Prog/main.c, [2656](#)
- ARMCM33\_STM32L5\_Nucleo\_L552ZE\_GCC/↔  
Boot/main.c, [2658](#)
- ARMCM33\_STM32L5\_Nucleo\_L552ZE\_GCC/↔  
Prog/main.c, [2660](#)
- ARMCM33\_STM32L5\_Nucleo\_L552ZE\_IAR/↔  
Boot/main.c, [2663](#)
- ARMCM33\_STM32L5\_Nucleo\_L552ZE\_IAR/↔  
Prog/main.c, [2665](#)
- ARMCM33\_STM32L5\_Nucleo\_L552ZE\_Keil/↔  
Boot/main.c, [2668](#)
- ARMCM33\_STM32L5\_Nucleo\_L552ZE\_Keil/↔  
Prog/main.c, [2670](#)
- ARMCM3\_EFM32\_Olimex\_EM32G880F128ST↔  
K\_GCC/Boot/main.c, [2673](#)
- ARMCM3\_EFM32\_Olimex\_EM32G880F128ST↔  
K\_GCC/Prog/main.c, [2674](#)
- ARMCM3\_EFM32\_Olimex\_EM32G880F128ST↔  
K\_IAR/Boot/main.c, [2675](#)
- ARMCM3\_EFM32\_Olimex\_EM32G880F128ST↔  
K\_IAR/Prog/main.c, [2676](#)



- ARMCM3\_LM3S\_EK\_LM3S6965\_GCC/Boot/main.c, [2678](#)
- ARMCM3\_LM3S\_EK\_LM3S6965\_GCC/Prog/main.c, [2679](#)
- ARMCM3\_LM3S\_EK\_LM3S6965\_IAR/Boot/main.c, [2680](#)
- ARMCM3\_LM3S\_EK\_LM3S6965\_IAR/Prog/main.c, [2682](#)
- ARMCM3\_LM3S\_EK\_LM3S8962\_GCC/Boot/main.c, [2683](#)
- ARMCM3\_LM3S\_EK\_LM3S8962\_GCC/Prog/main.c, [2684](#)
- ARMCM3\_LM3S\_EK\_LM3S8962\_IAR/Boot/main.c, [2685](#)
- ARMCM3\_LM3S\_EK\_LM3S8962\_IAR/Prog/main.c, [2687](#)
- ARMCM3\_STM32F1\_Nucleo\_F103RB\_GCC/Boot/main.c, [2688](#)
- ARMCM3\_STM32F1\_Nucleo\_F103RB\_GCC/Prog/main.c, [2691](#)
- ARMCM3\_STM32F1\_Nucleo\_F103RB\_IAR/Boot/main.c, [2693](#)
- ARMCM3\_STM32F1\_Nucleo\_F103RB\_IAR/Prog/main.c, [2696](#)
- ARMCM3\_STM32F1\_Nucleo\_F103RB\_Keil/Boot/main.c, [2698](#)
- ARMCM3\_STM32F1\_Nucleo\_F103RB\_Keil/Prog/main.c, [2701](#)
- ARMCM3\_STM32F1\_Olimex\_STM32H103\_GC/C/Boot/main.c, [2703](#)
- ARMCM3\_STM32F1\_Olimex\_STM32H103\_GC/C/Prog/main.c, [2706](#)
- ARMCM3\_STM32F1\_Olimex\_STM32H103\_IAR/R/Boot/main.c, [2708](#)
- ARMCM3\_STM32F1\_Olimex\_STM32H103\_IAR/R/Prog/main.c, [2711](#)
- ARMCM3\_STM32F1\_Olimex\_STM32H103\_Keil/Boot/main.c, [2713](#)
- ARMCM3\_STM32F1\_Olimex\_STM32H103\_Keil/Prog/main.c, [2716](#)
- ARMCM3\_STM32F1\_Olimex\_STM32P103\_GC/C/Boot/main.c, [2718](#)
- ARMCM3\_STM32F1\_Olimex\_STM32P103\_GC/C/Prog/main.c, [2721](#)
- ARMCM3\_STM32F1\_Olimex\_STM32P103\_IAR/R/Boot/main.c, [2723](#)
- ARMCM3\_STM32F1\_Olimex\_STM32P103\_IAR/R/Prog/main.c, [2726](#)
- ARMCM3\_STM32F1\_Olimex\_STM32P103\_Keil/Boot/main.c, [2728](#)
- ARMCM3\_STM32F1\_Olimex\_STM32P103\_Keil/Prog/main.c, [2731](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_GCC/Boot/main.c, [2733](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_GCC/Prog/main.c, [2736](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_IAR/Boot/main.c, [2738](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_IAR/Prog/main.c, [2741](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_Keil/Boot/main.c, [2743](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_Keil/Prog/main.c, [2746](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_GC/C/Boot/main.c, [2748](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_GC/C/Prog/main.c, [2751](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_IAR/R/Boot/main.c, [2753](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_IAR/R/Prog/main.c, [2756](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_Keil/Boot/main.c, [2758](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_Keil/Prog/main.c, [2761](#)
- ARMCM4\_S32K14\_S32K144EVB\_GCC/Boot/main.c, [2763](#)
- ARMCM4\_S32K14\_S32K144EVB\_GCC/Prog/main.c, [2765](#)
- ARMCM4\_S32K14\_S32K144EVB\_IAR/Boot/main.c, [2766](#)
- ARMCM4\_S32K14\_S32K144EVB\_IAR/Prog/main.c, [2768](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_GCC/Boot/main.c, [2770](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_GCC/Prog/main.c, [2773](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_IAR/Boot/main.c, [2776](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_IAR/Prog/main.c, [2778](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_Keil/Boot/main.c, [2781](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_Keil/Prog/main.c, [2783](#)
- ARMCM4\_STM32F3\_Nucleo\_F303K8\_GCC/Boot/main.c, [2786](#)
- ARMCM4\_STM32F3\_Nucleo\_F303K8\_GCC/Prog/main.c, [2788](#)
- ARMCM4\_STM32F3\_Nucleo\_F303K8\_IAR/Boot/main.c, [2791](#)
- ARMCM4\_STM32F3\_Nucleo\_F303K8\_IAR/Prog/main.c, [2793](#)
- ARMCM4\_STM32F3\_Nucleo\_F303K8\_Keil/Boot/main.c, [2796](#)
- ARMCM4\_STM32F3\_Nucleo\_F303K8\_Keil/Prog/main.c, [2798](#)
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_GCC/Boot/main.c, [2801](#)
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_GCC/Prog/main.c, [2803](#)
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_IAR/Boot/main.c, [2806](#)
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_IAR/Prog/main.c, [2808](#)

- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_Keil/↔  
  Boot/main.c, [2811](#)
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_Keil/↔  
  Prog/main.c, [2813](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_GC↔  
  C/Boot/main.c, [2816](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_GC↔  
  C/Prog/main.c, [2818](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_IA↔  
  R/Boot/main.c, [2821](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_IA↔  
  R/Prog/main.c, [2823](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_Keil/↔  
  Boot/main.c, [2826](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_Keil/↔  
  Prog/main.c, [2828](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_GCC/↔  
  Boot/main.c, [2831](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_GCC/↔  
  Prog/main.c, [2833](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_IAR/↔  
  Boot/main.c, [2836](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_IAR/↔  
  Prog/main.c, [2838](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_Keil/↔  
  Boot/main.c, [2841](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_Keil/↔  
  Prog/main.c, [2843](#)
- ARMCM4\_TM4C\_DK\_TM4C123G\_IAR/Boot/main.↔  
  c, [2846](#)
- ARMCM4\_TM4C\_DK\_TM4C123G\_IAR/Prog/main.↔  
  c, [2847](#)
- ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_GCC/↔  
  Boot/main.c, [2848](#)
- ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_GCC/↔  
  Prog/main.c, [2849](#)
- ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_IAR/↔  
  Boot/main.c, [2851](#)
- ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_IAR/↔  
  Prog/main.c, [2852](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_GCC/↔  
  Boot/main.c, [2854](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_GCC/↔  
  Prog/main.c, [2857](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_IAR/↔  
  Boot/main.c, [2859](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_IAR/↔  
  Prog/main.c, [2862](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_Keil/↔  
  Boot/main.c, [2864](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_Keil/↔  
  Prog/main.c, [2867](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_GCC/↔  
  Boot/main.c, [2869](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_GCC/↔  
  Prog/main.c, [2872](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_IAR/↔  
  Boot/main.c, [2874](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_IAR/↔  
  Prog/main.c, [2877](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_Keil/↔  
  Boot/main.c, [2879](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_Keil/↔  
  Prog/main.c, [2882](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_GCC/↔  
  Boot/main.c, [2884](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_GCC/↔  
  Prog/main.c, [2887](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_IAR/↔  
  Boot/main.c, [2889](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_IAR/↔  
  Prog/main.c, [2892](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_Keil/↔  
  Boot/main.c, [2894](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_Keil/↔  
  Prog/main.c, [2897](#)
- HCS12\_DevKit\_S12G128\_CodeWarrior/Boot/main.↔  
  c, [2899](#)
- HCS12\_DevKit\_S12G128\_CodeWarrior/Prog/main.↔  
  c, [2901](#)
- HCS12\_Evbplus\_Dragon12p\_CodeWarrior/↔  
  Boot/main.c, [2902](#)
- HCS12\_Evbplus\_Dragon12p\_CodeWarrior/↔  
  Prog/main.c, [2904](#)
- irq.c, [2132](#)
- IrqInterruptDisable, [2133](#)
- IrqInterruptEnable, [2134](#)
- IrqInterruptRestore, [2134](#)
- irq.h, [2134](#)
- IrqInterruptDisable, [2135](#)
- IrqInterruptEnable, [2135](#)
- IrqInterruptRestore, [2135](#)
- IrqInterruptDisable
- irq.c, [2133](#)
- irq.h, [2135](#)
- IrqInterruptEnable
- irq.c, [2134](#)
- irq.h, [2135](#)
- IrqInterruptRestore
- irq.c, [2134](#)
- irq.h, [2135](#)
- LOAD
- tSysTickRegs, [380](#)
- led.c, [2136](#), [2138](#), [2139](#), [2141](#), [2142](#), [2144](#), [2145](#), [2147](#),  
       [2148](#), [2150](#), [2151](#), [2153](#), [2154](#), [2156](#), [2157](#),  
       [2159](#), [2160](#), [2162](#), [2163](#), [2165](#), [2166](#), [2168](#),  
       [2169](#), [2171](#), [2172](#), [2174](#), [2175](#), [2177](#), [2178](#),  
       [2180](#), [2181](#), [2183](#), [2184](#), [2186](#), [2187](#), [2189](#),  
       [2190](#), [2192](#), [2193](#), [2195–2198](#), [2200](#), [2201](#),  
       [2203](#), [2204](#), [2206](#), [2207](#), [2209](#), [2210](#), [2212](#),  
       [2213](#), [2215](#), [2216](#), [2218](#), [2219](#), [2221](#), [2222](#),  
       [2224](#), [2225](#), [2227](#), [2228](#), [2230](#), [2231](#), [2233](#),  
       [2234](#), [2236](#), [2237](#), [2239](#), [2240](#), [2242](#), [2243](#),  
       [2245](#), [2246](#), [2248](#), [2249](#), [2251](#), [2252](#), [2254](#),  
       [2255](#), [2257](#), [2258](#), [2260](#), [2261](#), [2263](#), [2264](#),  
       [2266](#), [2267](#), [2269](#), [2271](#), [2272](#), [2274](#), [2276](#),

- 2277, 2279, 2281, 2282, 2284, 2286, 2287, 2289, 2290, 2292, 2293, 2295–2297, 2299, 2300, 2302, 2303, 2305, 2306, 2308, 2309, 2311, 2312, 2314, 2315, 2317, 2318, 2320, 2321, 2323–2325, 2327, 2328, 2330, 2331, 2333, 2334, 2336, 2337, 2339, 2340, 2342, 2343, 2345, 2346, 2348, 2349, 2351, 2352, 2354, 2355, 2357, 2358, 2360, 2361, 2363, 2364, 2366, 2367, 2369, 2370
- led.h, 2372–2374, 2376, 2377, 2379, 2380, 2382, 2383, 2385, 2386, 2388, 2389, 2391, 2392, 2394, 2395, 2397, 2398, 2400, 2401, 2403, 2404, 2406, 2407, 2409, 2410, 2412, 2413, 2415, 2416, 2418, 2419, 2421, 2422, 2424, 2425, 2427, 2428, 2430, 2432–2434, 2436–2438, 2440, 2441, 2443, 2444, 2446, 2447, 2449, 2450, 2452, 2453, 2455, 2456, 2458, 2459, 2461, 2462, 2464, 2465, 2467, 2468, 2470, 2471, 2473, 2474, 2476, 2477, 2479, 2480, 2482, 2483, 2485, 2486, 2488, 2489, 2491, 2492, 2494, 2495, 2497, 2498, 2500, 2501, 2503, 2504, 2506, 2507, 2509, 2510, 2512, 2513, 2515, 2516, 2518, 2519, 2521, 2522, 2524, 2525, 2527, 2528, 2530, 2531, 2533, 2535–2537, 2539, 2540, 2542, 2543, 2545, 2546, 2548, 2549, 2551, 2552, 2554, 2555, 2557, 2558, 2560–2562, 2564, 2565, 2567, 2568, 2570, 2571, 2573, 2574, 2576, 2577, 2579, 2580, 2582, 2583, 2585, 2586, 2588, 2589, 2591, 2592, 2594, 2595, 2597, 2598, 2600, 2601, 2603, 2604
- LedBlinkExit
- \_template/Boot/led.c, 2137
  - \_template/Boot/led.h, 2372
  - ARMCM0\_STM32F0\_Discovery\_STM32F051\_↔  
CubeIDE/Boot/App/led.c, 2140
  - ARMCM0\_STM32F0\_Discovery\_STM32F051\_↔  
CubeIDE/Boot/App/led.h, 2375
  - ARMCM0\_STM32F0\_Discovery\_STM32F051\_↔  
GCC/Boot/led.c, 2143
  - ARMCM0\_STM32F0\_Discovery\_STM32F051\_↔  
GCC/Boot/led.h, 2378
  - ARMCM0\_STM32F0\_Discovery\_STM32F051\_I↔  
AR/Boot/led.c, 2146
  - ARMCM0\_STM32F0\_Discovery\_STM32F051\_I↔  
AR/Boot/led.h, 2381
  - ARMCM0\_STM32F0\_Discovery\_STM32F051\_↔  
Keil/Boot/led.c, 2149
  - ARMCM0\_STM32F0\_Discovery\_STM32F051\_↔  
Keil/Boot/led.h, 2384
  - ARMCM0\_STM32F0\_Nucleo\_F091RC\_CubeID↔  
E/Boot/App/led.c, 2152
  - ARMCM0\_STM32F0\_Nucleo\_F091RC\_CubeID↔  
E/Boot/App/led.h, 2387
  - ARMCM0\_STM32F0\_Nucleo\_F091RC\_GCC/↔  
Boot/led.c, 2155
  - ARMCM0\_STM32F0\_Nucleo\_F091RC\_GCC/↔  
Boot/led.h, 2390
  - ARMCM0\_STM32F0\_Nucleo\_F091RC\_IAR/↔  
Boot/led.c, 2158
  - ARMCM0\_STM32F0\_Nucleo\_F091RC\_IAR/↔  
Boot/led.h, 2393
  - ARMCM0\_STM32F0\_Nucleo\_F091RC\_Keil/↔  
Boot/led.c, 2161
  - ARMCM0\_STM32F0\_Nucleo\_F091RC\_Keil/↔  
Boot/led.h, 2396
  - ARMCM0\_STM32G0\_Nucleo\_G071RB\_CubeID↔  
E/Boot/App/led.c, 2164
  - ARMCM0\_STM32G0\_Nucleo\_G071RB\_CubeID↔  
E/Boot/App/led.h, 2399
  - ARMCM0\_STM32G0\_Nucleo\_G071RB\_GCC/↔  
Boot/led.c, 2167
  - ARMCM0\_STM32G0\_Nucleo\_G071RB\_GCC/↔  
Boot/led.h, 2402
  - ARMCM0\_STM32G0\_Nucleo\_G071RB\_IAR/↔  
Boot/led.c, 2170
  - ARMCM0\_STM32G0\_Nucleo\_G071RB\_IAR/↔  
Boot/led.h, 2405
  - ARMCM0\_STM32G0\_Nucleo\_G071RB\_Keil/↔  
Boot/led.c, 2173
  - ARMCM0\_STM32G0\_Nucleo\_G071RB\_Keil/↔  
Boot/led.h, 2408
  - ARMCM0\_XMC1\_XMC1400\_Boot\_Kit\_GCC/↔  
Boot/led.c, 2176
  - ARMCM0\_XMC1\_XMC1400\_Boot\_Kit\_GCC/↔  
Boot/led.h, 2411
  - ARMCM0\_XMC1\_XMC1400\_Boot\_Kit\_IAR/↔  
Boot/led.c, 2179
  - ARMCM0\_XMC1\_XMC1400\_Boot\_Kit\_IAR/↔  
Boot/led.h, 2414
  - ARMCM33\_STM32L5\_Nucleo\_L552ZE\_CubeID↔  
E/Boot/App/led.c, 2182
  - ARMCM33\_STM32L5\_Nucleo\_L552ZE\_CubeID↔  
E/Boot/App/led.h, 2417
  - ARMCM33\_STM32L5\_Nucleo\_L552ZE\_GCC/↔  
Boot/led.c, 2185
  - ARMCM33\_STM32L5\_Nucleo\_L552ZE\_GCC/↔  
Boot/led.h, 2420
  - ARMCM33\_STM32L5\_Nucleo\_L552ZE\_IAR/↔  
Boot/led.c, 2188
  - ARMCM33\_STM32L5\_Nucleo\_L552ZE\_IAR/↔  
Boot/led.h, 2423
  - ARMCM33\_STM32L5\_Nucleo\_L552ZE\_Keil/↔  
Boot/led.c, 2191
  - ARMCM33\_STM32L5\_Nucleo\_L552ZE\_Keil/↔  
Boot/led.h, 2426
  - ARMCM3\_STM32F1\_Nucleo\_F103RB\_CubeID↔  
E/Boot/App/led.c, 2202
  - ARMCM3\_STM32F1\_Nucleo\_F103RB\_CubeID↔  
E/Boot/App/led.h, 2437
  - ARMCM3\_STM32F1\_Nucleo\_F103RB\_GCC/↔  
Boot/led.c, 2205
  - ARMCM3\_STM32F1\_Nucleo\_F103RB\_GCC/↔  
Boot/led.h, 2440
  - ARMCM3\_STM32F1\_Nucleo\_F103RB\_IAR/↔  
Boot/led.c, 2208

- ARMCM3\_STM32F1\_Nucleo\_F103RB\_IAR/↔  
Boot/led.h, [2443](#)
- ARMCM3\_STM32F1\_Nucleo\_F103RB\_Keil/↔  
Boot/led.c, [2211](#)
- ARMCM3\_STM32F1\_Nucleo\_F103RB\_Keil/↔  
Boot/led.h, [2446](#)
- ARMCM3\_STM32F1\_Olimex\_STM32H103\_↔  
CubeIDE/Boot/App/led.c, [2214](#)
- ARMCM3\_STM32F1\_Olimex\_STM32H103\_↔  
CubeIDE/Boot/App/led.h, [2449](#)
- ARMCM3\_STM32F1\_Olimex\_STM32H103\_GC↔  
C/Boot/led.c, [2217](#)
- ARMCM3\_STM32F1\_Olimex\_STM32H103\_GC↔  
C/Boot/led.h, [2452](#)
- ARMCM3\_STM32F1\_Olimex\_STM32H103\_IA↔  
R/Boot/led.c, [2220](#)
- ARMCM3\_STM32F1\_Olimex\_STM32H103\_IA↔  
R/Boot/led.h, [2455](#)
- ARMCM3\_STM32F1\_Olimex\_STM32H103\_↔  
Keil/Boot/led.c, [2223](#)
- ARMCM3\_STM32F1\_Olimex\_STM32H103\_↔  
Keil/Boot/led.h, [2458](#)
- ARMCM3\_STM32F1\_Olimex\_STM32P103\_↔  
CubeIDE/Boot/App/led.c, [2226](#)
- ARMCM3\_STM32F1\_Olimex\_STM32P103\_↔  
CubeIDE/Boot/App/led.h, [2461](#)
- ARMCM3\_STM32F1\_Olimex\_STM32P103\_GC↔  
C/Boot/led.c, [2229](#)
- ARMCM3\_STM32F1\_Olimex\_STM32P103\_GC↔  
C/Boot/led.h, [2464](#)
- ARMCM3\_STM32F1\_Olimex\_STM32P103\_IA↔  
R/Boot/led.c, [2232](#)
- ARMCM3\_STM32F1\_Olimex\_STM32P103\_IA↔  
R/Boot/led.h, [2467](#)
- ARMCM3\_STM32F1\_Olimex\_STM32P103\_Keil/↔  
Boot/led.c, [2235](#)
- ARMCM3\_STM32F1\_Olimex\_STM32P103\_Keil/↔  
Boot/led.h, [2470](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_Cubel↔  
DE/Boot/App/led.c, [2238](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_Cubel↔  
DE/Boot/App/led.h, [2473](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_GCC↔  
Boot/led.c, [2241](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_GCC↔  
Boot/led.h, [2476](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_IAR↔  
Boot/led.c, [2244](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_IAR↔  
Boot/led.h, [2479](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_Keil/↔  
Boot/led.c, [2247](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_Keil/↔  
Boot/led.h, [2482](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_↔  
CubeIDE/Boot/App/led.c, [2250](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_↔  
CubeIDE/Boot/App/led.h, [2485](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_GC↔  
C/Boot/led.c, [2253](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_GC↔  
C/Boot/led.h, [2488](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_IA↔  
R/Boot/led.c, [2256](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_IA↔  
R/Boot/led.h, [2491](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_Keil/↔  
Boot/led.c, [2259](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_Keil/↔  
Boot/led.h, [2494](#)
- ARMCM4\_S32K14\_S32K144EVB\_GCC/Boot/led.↔  
c, [2262](#)
- ARMCM4\_S32K14\_S32K144EVB\_GCC/Boot/led.↔  
h, [2497](#)
- ARMCM4\_S32K14\_S32K144EVB\_IAR/Boot/led.↔  
c, [2265](#)
- ARMCM4\_S32K14\_S32K144EVB\_IAR/Boot/led.↔  
h, [2500](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_Cube↔  
IDE/Boot/App/led.c, [2268](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_Cube↔  
IDE/Boot/App/led.h, [2503](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_GCC↔  
Boot/led.c, [2271](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_GCC↔  
Boot/led.h, [2506](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_IAR↔  
Boot/led.c, [2275](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_IAR↔  
Boot/led.h, [2509](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_Keil↔  
Boot/led.c, [2278](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_Keil↔  
Boot/led.h, [2512](#)
- ARMCM4\_STM32F3\_Nucleo\_F303K8\_CubelID↔  
E/Boot/App/led.c, [2281](#)
- ARMCM4\_STM32F3\_Nucleo\_F303K8\_CubelID↔  
E/Boot/App/led.h, [2515](#)
- ARMCM4\_STM32F3\_Nucleo\_F303K8\_GCC↔  
Boot/led.c, [2285](#)
- ARMCM4\_STM32F3\_Nucleo\_F303K8\_GCC↔  
Boot/led.h, [2518](#)
- ARMCM4\_STM32F3\_Nucleo\_F303K8\_IAR↔  
Boot/led.c, [2288](#)
- ARMCM4\_STM32F3\_Nucleo\_F303K8\_IAR↔  
Boot/led.h, [2521](#)
- ARMCM4\_STM32F3\_Nucleo\_F303K8\_Keil↔  
Boot/led.c, [2291](#)
- ARMCM4\_STM32F3\_Nucleo\_F303K8\_Keil↔  
Boot/led.h, [2524](#)
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_CubelID↔  
E/Boot/App/led.c, [2294](#)
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_CubelID↔  
E/Boot/App/led.h, [2527](#)
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_GCC↔  
Boot/led.h, [2530](#)

- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_IAR/↔  
Boot/led.h, [2532](#)
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_Keil/↔  
Boot/led.h, [2534](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_↔  
CubeIDE/Boot/App/led.c, [2301](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_↔  
CubeIDE/Boot/App/led.h, [2535](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_GC↔  
C/Boot/led.c, [2304](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_GC↔  
C/Boot/led.h, [2538](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_IA↔  
R/Boot/led.c, [2307](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_IA↔  
R/Boot/led.h, [2541](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_Keil/↔  
Boot/led.c, [2310](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_Keil/↔  
Boot/led.h, [2544](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_CubeID↔  
E/Boot/App/led.c, [2313](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_CubeID↔  
E/Boot/App/led.h, [2547](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_GCC/↔  
Boot/led.c, [2316](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_GCC/↔  
Boot/led.h, [2550](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_IAR/↔  
Boot/led.c, [2319](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_IAR/↔  
Boot/led.h, [2553](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_Keil/↔  
Boot/led.c, [2322](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_Keil/↔  
Boot/led.h, [2556](#)
- ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_GCC/↔  
Boot/led.c, [2326](#)
- ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_GCC/↔  
Boot/led.h, [2560](#)
- ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_IAR/↔  
Boot/led.c, [2329](#)
- ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_IAR/↔  
Boot/led.h, [2563](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_CubeID↔  
E/Boot/App/led.c, [2332](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_CubeID↔  
E/Boot/App/led.h, [2566](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_GCC/↔  
Boot/led.c, [2335](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_GCC/↔  
Boot/led.h, [2569](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_IAR/↔  
Boot/led.c, [2338](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_IAR/↔  
Boot/led.h, [2572](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_Keil/↔  
Boot/led.c, [2341](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_Keil/↔  
Boot/led.h, [2575](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_CubeID↔  
E/Boot/App/led.c, [2344](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_CubeID↔  
E/Boot/App/led.h, [2578](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_GCC/↔  
Boot/led.c, [2347](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_GCC/↔  
Boot/led.h, [2581](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_IAR/↔  
Boot/led.c, [2350](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_IAR/↔  
Boot/led.h, [2584](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_Keil/↔  
Boot/led.c, [2353](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_Keil/↔  
Boot/led.h, [2587](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_CubeID↔  
E/Boot/App/led.c, [2356](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_CubeID↔  
E/Boot/App/led.h, [2590](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_GCC/↔  
Boot/led.c, [2359](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_GCC/↔  
Boot/led.h, [2593](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_IAR/↔  
Boot/led.c, [2362](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_IAR/↔  
Boot/led.h, [2596](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_Keil/↔  
Boot/led.c, [2365](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_Keil/↔  
Boot/led.h, [2599](#)
- HCS12\_DevKit\_S12G128\_CodeWarrior/Boot/led.↔  
c, [2368](#)
- HCS12\_DevKit\_S12G128\_CodeWarrior/Boot/led.↔  
h, [2602](#)
- LedBlinkInit
  - \_template/Boot/led.c, [2137](#)
  - \_template/Boot/led.h, [2372](#)
- ARMCM0\_STM32F0\_Discovery\_STM32F051\_↔  
CubeIDE/Boot/App/led.c, [2140](#)
- ARMCM0\_STM32F0\_Discovery\_STM32F051\_↔  
CubeIDE/Boot/App/led.h, [2375](#)
- ARMCM0\_STM32F0\_Discovery\_STM32F051\_↔  
GCC/Boot/led.c, [2143](#)
- ARMCM0\_STM32F0\_Discovery\_STM32F051\_↔  
GCC/Boot/led.h, [2378](#)
- ARMCM0\_STM32F0\_Discovery\_STM32F051\_I↔  
AR/Boot/led.c, [2146](#)
- ARMCM0\_STM32F0\_Discovery\_STM32F051\_I↔  
AR/Boot/led.h, [2381](#)
- ARMCM0\_STM32F0\_Discovery\_STM32F051\_↔  
Keil/Boot/led.c, [2149](#)
- ARMCM0\_STM32F0\_Discovery\_STM32F051\_↔  
Keil/Boot/led.h, [2384](#)
- ARMCM0\_STM32F0\_Nucleo\_F091RC\_CubeID↔



- E/Boot/App/led.c, [2152](#)
- ARMCM0\_STM32F0\_Nucleo\_F091RC\_CubeID↔  
E/Boot/App/led.h, [2387](#)
- ARMCM0\_STM32F0\_Nucleo\_F091RC\_GCC↔  
Boot/led.c, [2155](#)
- ARMCM0\_STM32F0\_Nucleo\_F091RC\_GCC↔  
Boot/led.h, [2390](#)
- ARMCM0\_STM32F0\_Nucleo\_F091RC\_IAR↔  
Boot/led.c, [2158](#)
- ARMCM0\_STM32F0\_Nucleo\_F091RC\_IAR↔  
Boot/led.h, [2393](#)
- ARMCM0\_STM32F0\_Nucleo\_F091RC\_Keil↔  
Boot/led.c, [2161](#)
- ARMCM0\_STM32F0\_Nucleo\_F091RC\_Keil↔  
Boot/led.h, [2396](#)
- ARMCM0\_STM32G0\_Nucleo\_G071RB\_CubeID↔  
E/Boot/App/led.c, [2164](#)
- ARMCM0\_STM32G0\_Nucleo\_G071RB\_CubeID↔  
E/Boot/App/led.h, [2399](#)
- ARMCM0\_STM32G0\_Nucleo\_G071RB\_GCC↔  
Boot/led.c, [2167](#)
- ARMCM0\_STM32G0\_Nucleo\_G071RB\_GCC↔  
Boot/led.h, [2402](#)
- ARMCM0\_STM32G0\_Nucleo\_G071RB\_IAR↔  
Boot/led.c, [2170](#)
- ARMCM0\_STM32G0\_Nucleo\_G071RB\_IAR↔  
Boot/led.h, [2405](#)
- ARMCM0\_STM32G0\_Nucleo\_G071RB\_Keil↔  
Boot/led.c, [2173](#)
- ARMCM0\_STM32G0\_Nucleo\_G071RB\_Keil↔  
Boot/led.h, [2408](#)
- ARMCM0\_XMC1\_XMC1400\_Boot\_Kit\_GCC↔  
Boot/led.c, [2176](#)
- ARMCM0\_XMC1\_XMC1400\_Boot\_Kit\_GCC↔  
Boot/led.h, [2411](#)
- ARMCM0\_XMC1\_XMC1400\_Boot\_Kit\_IAR↔  
Boot/led.c, [2179](#)
- ARMCM0\_XMC1\_XMC1400\_Boot\_Kit\_IAR↔  
Boot/led.h, [2414](#)
- ARMCM33\_STM32L5\_Nucleo\_L552ZE\_CubeID↔  
E/Boot/App/led.c, [2182](#)
- ARMCM33\_STM32L5\_Nucleo\_L552ZE\_CubeID↔  
E/Boot/App/led.h, [2417](#)
- ARMCM33\_STM32L5\_Nucleo\_L552ZE\_GCC↔  
Boot/led.c, [2185](#)
- ARMCM33\_STM32L5\_Nucleo\_L552ZE\_GCC↔  
Boot/led.h, [2420](#)
- ARMCM33\_STM32L5\_Nucleo\_L552ZE\_IAR↔  
Boot/led.c, [2188](#)
- ARMCM33\_STM32L5\_Nucleo\_L552ZE\_IAR↔  
Boot/led.h, [2423](#)
- ARMCM33\_STM32L5\_Nucleo\_L552ZE\_Keil↔  
Boot/led.c, [2191](#)
- ARMCM33\_STM32L5\_Nucleo\_L552ZE\_Keil↔  
Boot/led.h, [2426](#)
- ARMCM3\_STM32F1\_Nucleo\_F103RB\_CubeID↔  
E/Boot/App/led.c, [2202](#)
- ARMCM3\_STM32F1\_Nucleo\_F103RB\_CubeID↔  
E/Boot/App/led.h, [2438](#)
- ARMCM3\_STM32F1\_Nucleo\_F103RB\_GCC↔  
Boot/led.c, [2205](#)
- ARMCM3\_STM32F1\_Nucleo\_F103RB\_GCC↔  
Boot/led.h, [2441](#)
- ARMCM3\_STM32F1\_Nucleo\_F103RB\_IAR↔  
Boot/led.c, [2208](#)
- ARMCM3\_STM32F1\_Nucleo\_F103RB\_IAR↔  
Boot/led.h, [2444](#)
- ARMCM3\_STM32F1\_Nucleo\_F103RB\_Keil↔  
Boot/led.c, [2211](#)
- ARMCM3\_STM32F1\_Nucleo\_F103RB\_Keil↔  
Boot/led.h, [2447](#)
- ARMCM3\_STM32F1\_Olimex\_STM32H103↔  
CubeIDE/Boot/App/led.c, [2214](#)
- ARMCM3\_STM32F1\_Olimex\_STM32H103↔  
CubeIDE/Boot/App/led.h, [2450](#)
- ARMCM3\_STM32F1\_Olimex\_STM32H103\_GC↔  
C/Boot/led.c, [2217](#)
- ARMCM3\_STM32F1\_Olimex\_STM32H103\_GC↔  
C/Boot/led.h, [2453](#)
- ARMCM3\_STM32F1\_Olimex\_STM32H103\_IA↔  
R/Boot/led.c, [2220](#)
- ARMCM3\_STM32F1\_Olimex\_STM32H103\_IA↔  
R/Boot/led.h, [2456](#)
- ARMCM3\_STM32F1\_Olimex\_STM32H103↔  
Keil/Boot/led.c, [2223](#)
- ARMCM3\_STM32F1\_Olimex\_STM32H103↔  
Keil/Boot/led.h, [2459](#)
- ARMCM3\_STM32F1\_Olimex\_STM32P103↔  
CubeIDE/Boot/App/led.c, [2226](#)
- ARMCM3\_STM32F1\_Olimex\_STM32P103↔  
CubeIDE/Boot/App/led.h, [2462](#)
- ARMCM3\_STM32F1\_Olimex\_STM32P103\_GC↔  
C/Boot/led.c, [2229](#)
- ARMCM3\_STM32F1\_Olimex\_STM32P103\_GC↔  
C/Boot/led.h, [2465](#)
- ARMCM3\_STM32F1\_Olimex\_STM32P103\_IA↔  
R/Boot/led.c, [2232](#)
- ARMCM3\_STM32F1\_Olimex\_STM32P103\_IA↔  
R/Boot/led.h, [2468](#)
- ARMCM3\_STM32F1\_Olimex\_STM32P103\_Keil↔  
Boot/led.c, [2235](#)
- ARMCM3\_STM32F1\_Olimex\_STM32P103\_Keil↔  
Boot/led.h, [2471](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_Cube↔  
DE/Boot/App/led.c, [2238](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_Cube↔  
DE/Boot/App/led.h, [2474](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_GCC↔  
Boot/led.c, [2241](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_GCC↔  
Boot/led.h, [2477](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_IAR↔  
Boot/led.c, [2244](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_IAR↔  
Boot/led.h, [2480](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_Keil↔

- Boot/led.c, [2247](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_Keil/↔  
Boot/led.h, [2483](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_↔  
CubeIDE/Boot/App/led.c, [2250](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_↔  
CubeIDE/Boot/App/led.h, [2486](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_GC↔  
C/Boot/led.c, [2253](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_GC↔  
C/Boot/led.h, [2489](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_IA↔  
R/Boot/led.c, [2256](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_IA↔  
R/Boot/led.h, [2492](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_Keil/↔  
Boot/led.c, [2259](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_Keil/↔  
Boot/led.h, [2495](#)
- ARMCM4\_S32K14\_S32K144EVB\_GCC/Boot/led.↔  
c, [2262](#)
- ARMCM4\_S32K14\_S32K144EVB\_GCC/Boot/led.↔  
h, [2498](#)
- ARMCM4\_S32K14\_S32K144EVB\_IAR/Boot/led.↔  
c, [2265](#)
- ARMCM4\_S32K14\_S32K144EVB\_IAR/Boot/led.↔  
h, [2501](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_Cube↔  
IDE/Boot/App/led.c, [2268](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_Cube↔  
IDE/Boot/App/led.h, [2504](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_GCC/↔  
Boot/led.c, [2272](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_GCC/↔  
Boot/led.h, [2507](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_IAR/↔  
Boot/led.c, [2275](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_IAR/↔  
Boot/led.h, [2510](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_Keil/↔  
Boot/led.c, [2278](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_Keil/↔  
Boot/led.h, [2513](#)
- ARMCM4\_STM32F3\_Nucleo\_F303K8\_CubeID↔  
E/Boot/App/led.c, [2282](#)
- ARMCM4\_STM32F3\_Nucleo\_F303K8\_CubeID↔  
E/Boot/App/led.h, [2516](#)
- ARMCM4\_STM32F3\_Nucleo\_F303K8\_GCC/↔  
Boot/led.c, [2285](#)
- ARMCM4\_STM32F3\_Nucleo\_F303K8\_GCC/↔  
Boot/led.h, [2519](#)
- ARMCM4\_STM32F3\_Nucleo\_F303K8\_IAR/↔  
Boot/led.c, [2288](#)
- ARMCM4\_STM32F3\_Nucleo\_F303K8\_IAR/↔  
Boot/led.h, [2522](#)
- ARMCM4\_STM32F3\_Nucleo\_F303K8\_Keil/↔  
Boot/led.c, [2291](#)
- ARMCM4\_STM32F3\_Nucleo\_F303K8\_Keil/↔  
Boot/led.h, [2525](#)
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_CubeID↔  
E/Boot/App/led.c, [2294](#)
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_CubeID↔  
E/Boot/App/led.h, [2528](#)
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_GCC/↔  
Boot/led.h, [2531](#)
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_IAR/↔  
Boot/led.h, [2532](#)
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_Keil/↔  
Boot/led.h, [2534](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_↔  
CubeIDE/Boot/App/led.c, [2301](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_↔  
CubeIDE/Boot/App/led.h, [2535](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_GC↔  
C/Boot/led.c, [2304](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_GC↔  
C/Boot/led.h, [2538](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_IA↔  
R/Boot/led.c, [2307](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_IA↔  
R/Boot/led.h, [2541](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_Keil/↔  
Boot/led.c, [2310](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_Keil/↔  
Boot/led.h, [2544](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_CubeID↔  
E/Boot/App/led.c, [2313](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_CubeID↔  
E/Boot/App/led.h, [2547](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_GCC/↔  
Boot/led.c, [2316](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_GCC/↔  
Boot/led.h, [2550](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_IAR/↔  
Boot/led.c, [2319](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_IAR/↔  
Boot/led.h, [2553](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_Keil/↔  
Boot/led.c, [2322](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_Keil/↔  
Boot/led.h, [2556](#)
- ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_GCC/↔  
Boot/led.c, [2326](#)
- ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_GCC/↔  
Boot/led.h, [2560](#)
- ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_IAR/↔  
Boot/led.c, [2329](#)
- ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_IAR/↔  
Boot/led.h, [2563](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_CubeID↔  
E/Boot/App/led.c, [2332](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_CubeID↔  
E/Boot/App/led.h, [2566](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_GCC/↔  
Boot/led.c, [2335](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_GCC/↔

- Boot/led.h, [2569](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_IAR/↔  
Boot/led.c, [2338](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_IAR/↔  
Boot/led.h, [2572](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_Keil/↔  
Boot/led.c, [2341](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_Keil/↔  
Boot/led.h, [2575](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_CubeID↔  
E/Boot/App/led.c, [2344](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_CubeID↔  
E/Boot/App/led.h, [2578](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_GCC/↔  
Boot/led.c, [2347](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_GCC/↔  
Boot/led.h, [2581](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_IAR/↔  
Boot/led.c, [2350](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_IAR/↔  
Boot/led.h, [2584](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_Keil/↔  
Boot/led.c, [2353](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_Keil/↔  
Boot/led.h, [2587](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_CubeID↔  
E/Boot/App/led.c, [2356](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_CubeID↔  
E/Boot/App/led.h, [2590](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_GCC/↔  
Boot/led.c, [2359](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_GCC/↔  
Boot/led.h, [2593](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_IAR/↔  
Boot/led.c, [2362](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_IAR/↔  
Boot/led.h, [2596](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_Keil/↔  
Boot/led.c, [2365](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_Keil/↔  
Boot/led.h, [2599](#)
- HCS12\_DevKit\_S12G128\_CodeWarrior/Boot/led.↔  
c, [2368](#)
- HCS12\_DevKit\_S12G128\_CodeWarrior/Boot/led.↔  
h, [2602](#)
- LedBlinkTask  
\_template/Boot/led.c, [2137](#)  
\_template/Boot/led.h, [2373](#)
- ARMCM0\_STM32F0\_Discovery\_STM32F051\_↔  
CubeIDE/Boot/App/led.c, [2141](#)
- ARMCM0\_STM32F0\_Discovery\_STM32F051\_↔  
CubeIDE/Boot/App/led.h, [2375](#)
- ARMCM0\_STM32F0\_Discovery\_STM32F051\_↔  
GCC/Boot/led.c, [2144](#)
- ARMCM0\_STM32F0\_Discovery\_STM32F051\_↔  
GCC/Boot/led.h, [2378](#)
- ARMCM0\_STM32F0\_Discovery\_STM32F051\_I↔  
AR/Boot/led.c, [2147](#)
- ARMCM0\_STM32F0\_Discovery\_STM32F051\_I↔  
AR/Boot/led.h, [2381](#)
- ARMCM0\_STM32F0\_Discovery\_STM32F051\_↔  
Keil/Boot/led.c, [2150](#)
- ARMCM0\_STM32F0\_Discovery\_STM32F051\_↔  
Keil/Boot/led.h, [2384](#)
- ARMCM0\_STM32F0\_Nucleo\_F091RC\_CubeID↔  
E/Boot/App/led.c, [2153](#)
- ARMCM0\_STM32F0\_Nucleo\_F091RC\_CubeID↔  
E/Boot/App/led.h, [2387](#)
- ARMCM0\_STM32F0\_Nucleo\_F091RC\_GCC/↔  
Boot/led.c, [2156](#)
- ARMCM0\_STM32F0\_Nucleo\_F091RC\_GCC/↔  
Boot/led.h, [2390](#)
- ARMCM0\_STM32F0\_Nucleo\_F091RC\_IAR/↔  
Boot/led.c, [2159](#)
- ARMCM0\_STM32F0\_Nucleo\_F091RC\_IAR/↔  
Boot/led.h, [2393](#)
- ARMCM0\_STM32F0\_Nucleo\_F091RC\_Keil/↔  
Boot/led.c, [2162](#)
- ARMCM0\_STM32F0\_Nucleo\_F091RC\_Keil/↔  
Boot/led.h, [2396](#)
- ARMCM0\_STM32G0\_Nucleo\_G071RB\_CubeID↔  
E/Boot/App/led.c, [2165](#)
- ARMCM0\_STM32G0\_Nucleo\_G071RB\_CubeID↔  
E/Boot/App/led.h, [2399](#)
- ARMCM0\_STM32G0\_Nucleo\_G071RB\_GCC/↔  
Boot/led.c, [2168](#)
- ARMCM0\_STM32G0\_Nucleo\_G071RB\_GCC/↔  
Boot/led.h, [2402](#)
- ARMCM0\_STM32G0\_Nucleo\_G071RB\_IAR/↔  
Boot/led.c, [2171](#)
- ARMCM0\_STM32G0\_Nucleo\_G071RB\_IAR/↔  
Boot/led.h, [2405](#)
- ARMCM0\_STM32G0\_Nucleo\_G071RB\_Keil/↔  
Boot/led.c, [2174](#)
- ARMCM0\_STM32G0\_Nucleo\_G071RB\_Keil/↔  
Boot/led.h, [2408](#)
- ARMCM0\_XMC1\_XMC1400\_Boot\_Kit\_GCC/↔  
Boot/led.c, [2177](#)
- ARMCM0\_XMC1\_XMC1400\_Boot\_Kit\_GCC/↔  
Boot/led.h, [2411](#)
- ARMCM0\_XMC1\_XMC1400\_Boot\_Kit\_IAR/↔  
Boot/led.c, [2180](#)
- ARMCM0\_XMC1\_XMC1400\_Boot\_Kit\_IAR/↔  
Boot/led.h, [2414](#)
- ARMCM33\_STM32L5\_Nucleo\_L552ZE\_CubeID↔  
E/Boot/App/led.c, [2183](#)
- ARMCM33\_STM32L5\_Nucleo\_L552ZE\_CubeID↔  
E/Boot/App/led.h, [2417](#)
- ARMCM33\_STM32L5\_Nucleo\_L552ZE\_GCC/↔  
Boot/led.c, [2186](#)
- ARMCM33\_STM32L5\_Nucleo\_L552ZE\_GCC/↔  
Boot/led.h, [2420](#)
- ARMCM33\_STM32L5\_Nucleo\_L552ZE\_IAR/↔  
Boot/led.c, [2189](#)
- ARMCM33\_STM32L5\_Nucleo\_L552ZE\_IAR/↔  
Boot/led.h, [2423](#)



- ARMCM33\_STM32L5\_Nucleo\_L552ZE\_Keil/↔  
Boot/led.c, [2192](#)
- ARMCM33\_STM32L5\_Nucleo\_L552ZE\_Keil/↔  
Boot/led.h, [2426](#)
- ARMCM3\_STM32F1\_Nucleo\_F103RB\_CubeID↔  
E/Boot/App/led.c, [2202](#)
- ARMCM3\_STM32F1\_Nucleo\_F103RB\_CubeID↔  
E/Boot/App/led.h, [2438](#)
- ARMCM3\_STM32F1\_Nucleo\_F103RB\_GCC/↔  
Boot/led.c, [2205](#)
- ARMCM3\_STM32F1\_Nucleo\_F103RB\_GCC/↔  
Boot/led.h, [2441](#)
- ARMCM3\_STM32F1\_Nucleo\_F103RB\_IAR/↔  
Boot/led.c, [2208](#)
- ARMCM3\_STM32F1\_Nucleo\_F103RB\_IAR/↔  
Boot/led.h, [2444](#)
- ARMCM3\_STM32F1\_Nucleo\_F103RB\_Keil/↔  
Boot/led.c, [2211](#)
- ARMCM3\_STM32F1\_Nucleo\_F103RB\_Keil/↔  
Boot/led.h, [2447](#)
- ARMCM3\_STM32F1\_Olimex\_STM32H103\_↔  
CubeIDE/Boot/App/led.c, [2214](#)
- ARMCM3\_STM32F1\_Olimex\_STM32H103\_↔  
CubeIDE/Boot/App/led.h, [2450](#)
- ARMCM3\_STM32F1\_Olimex\_STM32H103\_GC↔  
C/Boot/led.c, [2218](#)
- ARMCM3\_STM32F1\_Olimex\_STM32H103\_GC↔  
C/Boot/led.h, [2453](#)
- ARMCM3\_STM32F1\_Olimex\_STM32H103\_IA↔  
R/Boot/led.c, [2221](#)
- ARMCM3\_STM32F1\_Olimex\_STM32H103\_IA↔  
R/Boot/led.h, [2456](#)
- ARMCM3\_STM32F1\_Olimex\_STM32H103\_↔  
Keil/Boot/led.c, [2224](#)
- ARMCM3\_STM32F1\_Olimex\_STM32H103\_↔  
Keil/Boot/led.h, [2459](#)
- ARMCM3\_STM32F1\_Olimex\_STM32P103\_↔  
CubeIDE/Boot/App/led.c, [2227](#)
- ARMCM3\_STM32F1\_Olimex\_STM32P103\_↔  
CubeIDE/Boot/App/led.h, [2462](#)
- ARMCM3\_STM32F1\_Olimex\_STM32P103\_GC↔  
C/Boot/led.c, [2230](#)
- ARMCM3\_STM32F1\_Olimex\_STM32P103\_GC↔  
C/Boot/led.h, [2465](#)
- ARMCM3\_STM32F1\_Olimex\_STM32P103\_IA↔  
R/Boot/led.c, [2233](#)
- ARMCM3\_STM32F1\_Olimex\_STM32P103\_IA↔  
R/Boot/led.h, [2468](#)
- ARMCM3\_STM32F1\_Olimex\_STM32P103\_Keil/↔  
Boot/led.c, [2236](#)
- ARMCM3\_STM32F1\_Olimex\_STM32P103\_Keil/↔  
Boot/led.h, [2471](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_Cube↔  
DE/Boot/App/led.c, [2239](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_Cube↔  
DE/Boot/App/led.h, [2474](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_GCC/↔  
Boot/led.c, [2242](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_GCC/↔  
Boot/led.h, [2477](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_IAR/↔  
Boot/led.c, [2245](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_IAR/↔  
Boot/led.h, [2480](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_Keil/↔  
Boot/led.c, [2248](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_Keil/↔  
Boot/led.h, [2483](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_↔  
CubeIDE/Boot/App/led.c, [2251](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_↔  
CubeIDE/Boot/App/led.h, [2486](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_GC↔  
C/Boot/led.c, [2254](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_GC↔  
C/Boot/led.h, [2489](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_IA↔  
R/Boot/led.c, [2257](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_IA↔  
R/Boot/led.h, [2492](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_Keil/↔  
Boot/led.c, [2260](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_Keil/↔  
Boot/led.h, [2495](#)
- ARMCM4\_S32K14\_S32K144EVB\_GCC/Boot/led.↔  
c, [2263](#)
- ARMCM4\_S32K14\_S32K144EVB\_GCC/Boot/led.↔  
h, [2498](#)
- ARMCM4\_S32K14\_S32K144EVB\_IAR/Boot/led.↔  
c, [2266](#)
- ARMCM4\_S32K14\_S32K144EVB\_IAR/Boot/led.↔  
h, [2501](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_Cube↔  
IDE/Boot/App/led.c, [2269](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_Cube↔  
IDE/Boot/App/led.h, [2504](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_GCC/↔  
Boot/led.c, [2272](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_GCC/↔  
Boot/led.h, [2507](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_IAR/↔  
Boot/led.c, [2275](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_IAR/↔  
Boot/led.h, [2510](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_Keil/↔  
Boot/led.c, [2279](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_Keil/↔  
Boot/led.h, [2513](#)
- ARMCM4\_STM32F3\_Nucleo\_F303K8\_CubeID↔  
E/Boot/App/led.c, [2282](#)
- ARMCM4\_STM32F3\_Nucleo\_F303K8\_CubeID↔  
E/Boot/App/led.h, [2516](#)
- ARMCM4\_STM32F3\_Nucleo\_F303K8\_GCC/↔  
Boot/led.c, [2285](#)
- ARMCM4\_STM32F3\_Nucleo\_F303K8\_GCC/↔  
Boot/led.h, [2519](#)

- ARMCM4\_STM32F3\_Nucleo\_F303K8\_IAR/↔  
Boot/led.c, [2288](#)
- ARMCM4\_STM32F3\_Nucleo\_F303K8\_IAR/↔  
Boot/led.h, [2522](#)
- ARMCM4\_STM32F3\_Nucleo\_F303K8\_Keil/↔  
Boot/led.c, [2291](#)
- ARMCM4\_STM32F3\_Nucleo\_F303K8\_Keil/↔  
Boot/led.h, [2525](#)
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_CubeID↔  
E/Boot/App/led.c, [2294](#)
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_CubeID↔  
E/Boot/App/led.h, [2528](#)
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_GCC/↔  
Boot/led.h, [2531](#)
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_IAR/↔  
Boot/led.h, [2533](#)
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_Keil/↔  
Boot/led.h, [2534](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_↔  
CubeIDE/Boot/App/led.c, [2301](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_↔  
CubeIDE/Boot/App/led.h, [2536](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_GC↔  
C/Boot/led.c, [2304](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_GC↔  
C/Boot/led.h, [2538](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_IA↔  
R/Boot/led.c, [2307](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_IA↔  
R/Boot/led.h, [2541](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_Keil/↔  
Boot/led.c, [2310](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_Keil/↔  
Boot/led.h, [2544](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_CubeID↔  
E/Boot/App/led.c, [2313](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_CubeID↔  
E/Boot/App/led.h, [2547](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_GCC/↔  
Boot/led.c, [2316](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_GCC/↔  
Boot/led.h, [2550](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_IAR/↔  
Boot/led.c, [2319](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_IAR/↔  
Boot/led.h, [2553](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_Keil/↔  
Boot/led.c, [2322](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_Keil/↔  
Boot/led.h, [2556](#)
- ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_GCC/↔  
Boot/led.c, [2327](#)
- ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_GCC/↔  
Boot/led.h, [2561](#)
- ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_IAR/↔  
Boot/led.c, [2330](#)
- ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_IAR/↔  
Boot/led.h, [2563](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_CubeID↔  
E/Boot/App/led.c, [2333](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_CubeID↔  
E/Boot/App/led.h, [2566](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_GCC/↔  
Boot/led.c, [2336](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_GCC/↔  
Boot/led.h, [2569](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_IAR/↔  
Boot/led.c, [2339](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_IAR/↔  
Boot/led.h, [2572](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_Keil/↔  
Boot/led.c, [2342](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_Keil/↔  
Boot/led.h, [2575](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_CubeID↔  
E/Boot/App/led.c, [2345](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_CubeID↔  
E/Boot/App/led.h, [2578](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_GCC/↔  
Boot/led.c, [2348](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_GCC/↔  
Boot/led.h, [2581](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_IAR/↔  
Boot/led.c, [2351](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_IAR/↔  
Boot/led.h, [2584](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_Keil/↔  
Boot/led.c, [2354](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_Keil/↔  
Boot/led.h, [2587](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_CubeID↔  
E/Boot/App/led.c, [2357](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_CubeID↔  
E/Boot/App/led.h, [2590](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_GCC/↔  
Boot/led.c, [2360](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_GCC/↔  
Boot/led.h, [2593](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_IAR/↔  
Boot/led.c, [2363](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_IAR/↔  
Boot/led.h, [2596](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_Keil/↔  
Boot/led.c, [2366](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_Keil/↔  
Boot/led.h, [2599](#)
- HCS12\_DevKit\_S12G128\_CodeWarrior/Boot/led.↔  
c, [2369](#)
- HCS12\_DevKit\_S12G128\_CodeWarrior/Boot/led.↔  
h, [2603](#)
- LedInit  
\_template/Prog/led.c, [2139](#)  
\_template/Prog/led.h, [2374](#)
- ARMCM0\_STM32F0\_Discovery\_STM32F051\_↔  
CubeIDE/Prog/App/led.c, [2142](#)
- ARMCM0\_STM32F0\_Discovery\_STM32F051\_↔

- CubeIDE/Prog/App/led.h, [2376](#)
- ARMCM0\_STM32F0\_Discovery\_STM32F051\_↔  
GCC/Prog/led.c, [2145](#)
- ARMCM0\_STM32F0\_Discovery\_STM32F051\_↔  
GCC/Prog/led.h, [2379](#)
- ARMCM0\_STM32F0\_Discovery\_STM32F051\_↔  
AR/Prog/led.c, [2148](#)
- ARMCM0\_STM32F0\_Discovery\_STM32F051\_↔  
AR/Prog/led.h, [2382](#)
- ARMCM0\_STM32F0\_Discovery\_STM32F051\_↔  
Keil/Prog/led.c, [2151](#)
- ARMCM0\_STM32F0\_Discovery\_STM32F051\_↔  
Keil/Prog/led.h, [2385](#)
- ARMCM0\_STM32F0\_Nucleo\_F091RC\_CubeID↔  
E/Prog/App/led.c, [2154](#)
- ARMCM0\_STM32F0\_Nucleo\_F091RC\_CubeID↔  
E/Prog/App/led.h, [2388](#)
- ARMCM0\_STM32F0\_Nucleo\_F091RC\_GCC/↔  
Prog/led.c, [2157](#)
- ARMCM0\_STM32F0\_Nucleo\_F091RC\_GCC/↔  
Prog/led.h, [2391](#)
- ARMCM0\_STM32F0\_Nucleo\_F091RC\_IAR/↔  
Prog/led.c, [2160](#)
- ARMCM0\_STM32F0\_Nucleo\_F091RC\_IAR/↔  
Prog/led.h, [2394](#)
- ARMCM0\_STM32F0\_Nucleo\_F091RC\_Keil/↔  
Prog/led.c, [2163](#)
- ARMCM0\_STM32F0\_Nucleo\_F091RC\_Keil/↔  
Prog/led.h, [2397](#)
- ARMCM0\_STM32G0\_Nucleo\_G071RB\_CubeID↔  
E/Prog/App/led.c, [2166](#)
- ARMCM0\_STM32G0\_Nucleo\_G071RB\_CubeID↔  
E/Prog/App/led.h, [2400](#)
- ARMCM0\_STM32G0\_Nucleo\_G071RB\_GCC/↔  
Prog/led.c, [2169](#)
- ARMCM0\_STM32G0\_Nucleo\_G071RB\_GCC/↔  
Prog/led.h, [2403](#)
- ARMCM0\_STM32G0\_Nucleo\_G071RB\_IAR/↔  
Prog/led.c, [2172](#)
- ARMCM0\_STM32G0\_Nucleo\_G071RB\_IAR/↔  
Prog/led.h, [2406](#)
- ARMCM0\_STM32G0\_Nucleo\_G071RB\_Keil/↔  
Prog/led.c, [2175](#)
- ARMCM0\_STM32G0\_Nucleo\_G071RB\_Keil/↔  
Prog/led.h, [2409](#)
- ARMCM0\_XMC1\_XMC1400\_Boot\_Kit\_GCC/↔  
Prog/led.c, [2178](#)
- ARMCM0\_XMC1\_XMC1400\_Boot\_Kit\_GCC/↔  
Prog/led.h, [2412](#)
- ARMCM0\_XMC1\_XMC1400\_Boot\_Kit\_IAR/↔  
Prog/led.c, [2181](#)
- ARMCM0\_XMC1\_XMC1400\_Boot\_Kit\_IAR/↔  
Prog/led.h, [2415](#)
- ARMCM33\_STM32L5\_Nucleo\_L552ZE\_CubeID↔  
E/Prog/App/led.c, [2184](#)
- ARMCM33\_STM32L5\_Nucleo\_L552ZE\_CubeID↔  
E/Prog/App/led.h, [2418](#)
- ARMCM33\_STM32L5\_Nucleo\_L552ZE\_GCC/↔  
Prog/led.c, [2187](#)
- ARMCM33\_STM32L5\_Nucleo\_L552ZE\_GCC/↔  
Prog/led.h, [2421](#)
- ARMCM33\_STM32L5\_Nucleo\_L552ZE\_IAR/↔  
Prog/led.c, [2190](#)
- ARMCM33\_STM32L5\_Nucleo\_L552ZE\_IAR/↔  
Prog/led.h, [2424](#)
- ARMCM33\_STM32L5\_Nucleo\_L552ZE\_Keil/↔  
Prog/led.c, [2193](#)
- ARMCM33\_STM32L5\_Nucleo\_L552ZE\_Keil/↔  
Prog/led.h, [2427](#)
- ARMCM3\_EFM32\_Olimex\_EM32G880F128ST↔  
K\_GCC/Prog/led.c, [2194](#)
- ARMCM3\_EFM32\_Olimex\_EM32G880F128ST↔  
K\_GCC/Prog/led.h, [2429](#)
- ARMCM3\_EFM32\_Olimex\_EM32G880F128ST↔  
K\_IAR/Prog/led.c, [2195](#)
- ARMCM3\_EFM32\_Olimex\_EM32G880F128ST↔  
K\_IAR/Prog/led.h, [2431](#)
- ARMCM3\_LM3S\_EK\_LM3S6965\_GCC/Prog/led.↔  
c, [2196](#)
- ARMCM3\_LM3S\_EK\_LM3S6965\_GCC/Prog/led.↔  
h, [2432](#)
- ARMCM3\_LM3S\_EK\_LM3S6965\_IAR/Prog/led.c,  
[2198](#)
- ARMCM3\_LM3S\_EK\_LM3S6965\_IAR/Prog/led.h,  
[2433](#)
- ARMCM3\_LM3S\_EK\_LM3S8962\_GCC/Prog/led.↔  
c, [2199](#)
- ARMCM3\_LM3S\_EK\_LM3S8962\_GCC/Prog/led.↔  
h, [2435](#)
- ARMCM3\_LM3S\_EK\_LM3S8962\_IAR/Prog/led.c,  
[2200](#)
- ARMCM3\_LM3S\_EK\_LM3S8962\_IAR/Prog/led.h,  
[2436](#)
- ARMCM3\_STM32F1\_Nucleo\_F103RB\_CubeID↔  
E/Prog/App/led.c, [2203](#)
- ARMCM3\_STM32F1\_Nucleo\_F103RB\_CubeID↔  
E/Prog/App/led.h, [2439](#)
- ARMCM3\_STM32F1\_Nucleo\_F103RB\_GCC/↔  
Prog/led.c, [2206](#)
- ARMCM3\_STM32F1\_Nucleo\_F103RB\_GCC/↔  
Prog/led.h, [2442](#)
- ARMCM3\_STM32F1\_Nucleo\_F103RB\_IAR/↔  
Prog/led.c, [2209](#)
- ARMCM3\_STM32F1\_Nucleo\_F103RB\_IAR/↔  
Prog/led.h, [2445](#)
- ARMCM3\_STM32F1\_Nucleo\_F103RB\_Keil/↔  
Prog/led.c, [2212](#)
- ARMCM3\_STM32F1\_Nucleo\_F103RB\_Keil/↔  
Prog/led.h, [2448](#)
- ARMCM3\_STM32F1\_Olimex\_STM32H103\_↔  
CubeIDE/Prog/App/led.c, [2215](#)
- ARMCM3\_STM32F1\_Olimex\_STM32H103\_↔  
CubeIDE/Prog/App/led.h, [2451](#)
- ARMCM3\_STM32F1\_Olimex\_STM32H103\_GC↔  
C/Prog/led.c, [2219](#)
- ARMCM3\_STM32F1\_Olimex\_STM32H103\_GC↔

- C/Prog/led.h, [2454](#)
- ARMCM3\_STM32F1\_Olimex\_STM32H103\_IA↔  
R/Prog/led.c, [2222](#)
- ARMCM3\_STM32F1\_Olimex\_STM32H103\_IA↔  
R/Prog/led.h, [2457](#)
- ARMCM3\_STM32F1\_Olimex\_STM32H103\_↔  
Keil/Prog/led.c, [2225](#)
- ARMCM3\_STM32F1\_Olimex\_STM32H103\_↔  
Keil/Prog/led.h, [2460](#)
- ARMCM3\_STM32F1\_Olimex\_STM32P103\_↔  
CubeIDE/Prog/App/led.c, [2228](#)
- ARMCM3\_STM32F1\_Olimex\_STM32P103\_↔  
CubeIDE/Prog/App/led.h, [2463](#)
- ARMCM3\_STM32F1\_Olimex\_STM32P103\_GC↔  
C/Prog/led.c, [2231](#)
- ARMCM3\_STM32F1\_Olimex\_STM32P103\_GC↔  
C/Prog/led.h, [2466](#)
- ARMCM3\_STM32F1\_Olimex\_STM32P103\_IA↔  
R/Prog/led.c, [2234](#)
- ARMCM3\_STM32F1\_Olimex\_STM32P103\_IA↔  
R/Prog/led.h, [2469](#)
- ARMCM3\_STM32F1\_Olimex\_STM32P103\_Keil/↔  
Prog/led.c, [2237](#)
- ARMCM3\_STM32F1\_Olimex\_STM32P103\_Keil/↔  
Prog/led.h, [2472](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_CubeID↔  
DE/Prog/App/led.c, [2240](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_CubeID↔  
DE/Prog/App/led.h, [2475](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_GCC/↔  
Prog/led.c, [2243](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_GCC/↔  
Prog/led.h, [2478](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_IAR/↔  
Prog/led.c, [2246](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_IAR/↔  
Prog/led.h, [2481](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_Keil/↔  
Prog/led.c, [2249](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_Keil/↔  
Prog/led.h, [2484](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_↔  
CubeIDE/Prog/App/led.c, [2252](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_↔  
CubeIDE/Prog/App/led.h, [2487](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_GC↔  
C/Prog/led.c, [2255](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_GC↔  
C/Prog/led.h, [2490](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_IA↔  
R/Prog/led.c, [2258](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_IA↔  
R/Prog/led.h, [2493](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_Keil/↔  
Prog/led.c, [2261](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_Keil/↔  
Prog/led.h, [2496](#)
- ARMCM4\_S32K14\_S32K144EVB\_GCC/Prog/led.↔  
c, [2264](#)
- ARMCM4\_S32K14\_S32K144EVB\_GCC/Prog/led.↔  
h, [2499](#)
- ARMCM4\_S32K14\_S32K144EVB\_IAR/Prog/led.↔  
c, [2267](#)
- ARMCM4\_S32K14\_S32K144EVB\_IAR/Prog/led.↔  
h, [2502](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_Cube↔  
IDE/Prog/App/led.c, [2270](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_Cube↔  
IDE/Prog/App/led.h, [2505](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_GCC/↔  
Prog/led.c, [2273](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_GCC/↔  
Prog/led.h, [2508](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_IAR/↔  
Prog/led.c, [2276](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_IAR/↔  
Prog/led.h, [2511](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_Keil/↔  
Prog/led.c, [2280](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_Keil/↔  
Prog/led.h, [2514](#)
- ARMCM4\_STM32F3\_Nucleo\_F303K8\_CubeID↔  
E/Prog/App/led.c, [2283](#)
- ARMCM4\_STM32F3\_Nucleo\_F303K8\_CubeID↔  
E/Prog/App/led.h, [2517](#)
- ARMCM4\_STM32F3\_Nucleo\_F303K8\_GCC/↔  
Prog/led.c, [2286](#)
- ARMCM4\_STM32F3\_Nucleo\_F303K8\_GCC/↔  
Prog/led.h, [2520](#)
- ARMCM4\_STM32F3\_Nucleo\_F303K8\_IAR/↔  
Prog/led.c, [2289](#)
- ARMCM4\_STM32F3\_Nucleo\_F303K8\_IAR/↔  
Prog/led.h, [2523](#)
- ARMCM4\_STM32F3\_Nucleo\_F303K8\_Keil/↔  
Prog/led.c, [2292](#)
- ARMCM4\_STM32F3\_Nucleo\_F303K8\_Keil/↔  
Prog/led.h, [2526](#)
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_CubeID↔  
E/Prog/App/led.c, [2295](#)
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_CubeID↔  
E/Prog/App/led.h, [2529](#)
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_GCC/↔  
Prog/led.c, [2297](#)
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_IAR/↔  
Prog/led.c, [2298](#)
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_Keil/↔  
Prog/led.c, [2299](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_↔  
CubeIDE/Prog/App/led.c, [2302](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_↔  
CubeIDE/Prog/App/led.h, [2537](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_GC↔  
C/Prog/led.c, [2305](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_GC↔  
C/Prog/led.h, [2539](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_IA↔

- R/Prog/led.c, [2308](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_IAR/↔  
R/Prog/led.h, [2542](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_Keil/↔  
Prog/led.c, [2311](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_Keil/↔  
Prog/led.h, [2545](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_CubeID↔  
E/Prog/App/led.c, [2314](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_CubeID↔  
E/Prog/App/led.h, [2548](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_GCC/↔  
Prog/led.c, [2317](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_GCC/↔  
Prog/led.h, [2551](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_IAR/↔  
Prog/led.c, [2320](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_IAR/↔  
Prog/led.h, [2554](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_Keil/↔  
Prog/led.c, [2323](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_Keil/↔  
Prog/led.h, [2557](#)
- ARMCM4\_TM4C\_DK\_TM4C123G\_IAR/Prog/led.↔  
c, [2324](#)
- ARMCM4\_TM4C\_DK\_TM4C123G\_IAR/Prog/led.↔  
h, [2559](#)
- ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_GCC/↔  
Prog/led.c, [2328](#)
- ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_GCC/↔  
Prog/led.h, [2562](#)
- ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_IAR/↔  
Prog/led.c, [2331](#)
- ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_IAR/↔  
Prog/led.h, [2564](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_CubeID↔  
E/Prog/App/led.c, [2334](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_CubeID↔  
E/Prog/App/led.h, [2567](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_GCC/↔  
Prog/led.c, [2337](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_GCC/↔  
Prog/led.h, [2570](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_IAR/↔  
Prog/led.c, [2340](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_IAR/↔  
Prog/led.h, [2573](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_Keil/↔  
Prog/led.c, [2343](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_Keil/↔  
Prog/led.h, [2576](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_CubeID↔  
E/Prog/App/led.c, [2346](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_CubeID↔  
E/Prog/App/led.h, [2579](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_GCC/↔  
Prog/led.c, [2349](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_GCC/↔  
Prog/led.h, [2582](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_IAR/↔  
Prog/led.c, [2352](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_IAR/↔  
Prog/led.h, [2585](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_Keil/↔  
Prog/led.c, [2355](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_Keil/↔  
Prog/led.h, [2588](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_CubeID↔  
E/Prog/App/led.c, [2358](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_CubeID↔  
E/Prog/App/led.h, [2591](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_GCC/↔  
Prog/led.c, [2361](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_GCC/↔  
Prog/led.h, [2594](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_IAR/↔  
Prog/led.c, [2364](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_IAR/↔  
Prog/led.h, [2597](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_Keil/↔  
Prog/led.c, [2367](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_Keil/↔  
Prog/led.h, [2600](#)
- HCS12\_DevKit\_S12G128\_CodeWarrior/Prog/led.↔  
c, [2370](#)
- HCS12\_DevKit\_S12G128\_CodeWarrior/Prog/led.↔  
h, [2604](#)
- HCS12\_Evbplus\_Dragon12p\_CodeWarrior/↔  
Prog/led.c, [2371](#)
- HCS12\_Evbplus\_Dragon12p\_CodeWarrior/↔  
Prog/led.h, [2605](#)
- LedToggle
  - \_template/Prog/led.c, [2139](#)
  - \_template/Prog/led.h, [2374](#)
  - ARMCM0\_STM32F0\_Discovery\_STM32F051\_↔  
CubeIDE/Prog/App/led.c, [2142](#)
  - ARMCM0\_STM32F0\_Discovery\_STM32F051\_↔  
CubeIDE/Prog/App/led.h, [2377](#)
  - ARMCM0\_STM32F0\_Discovery\_STM32F051\_↔  
GCC/Prog/led.c, [2145](#)
  - ARMCM0\_STM32F0\_Discovery\_STM32F051\_↔  
GCC/Prog/led.h, [2380](#)
  - ARMCM0\_STM32F0\_Discovery\_STM32F051\_I↔  
AR/Prog/led.c, [2148](#)
  - ARMCM0\_STM32F0\_Discovery\_STM32F051\_I↔  
AR/Prog/led.h, [2383](#)
  - ARMCM0\_STM32F0\_Discovery\_STM32F051\_↔  
Keil/Prog/led.c, [2151](#)
  - ARMCM0\_STM32F0\_Discovery\_STM32F051\_↔  
Keil/Prog/led.h, [2386](#)
  - ARMCM0\_STM32F0\_Nucleo\_F091RC\_CubeID↔  
E/Prog/App/led.c, [2154](#)
  - ARMCM0\_STM32F0\_Nucleo\_F091RC\_CubeID↔  
E/Prog/App/led.h, [2389](#)
  - ARMCM0\_STM32F0\_Nucleo\_F091RC\_GCC/↔  
Prog/led.c, [2157](#)



- ARMCM0\_STM32F0\_Nucleo\_F091RC\_GCC/↔  
Prog/led.h, [2392](#)
- ARMCM0\_STM32F0\_Nucleo\_F091RC\_IAR/↔  
Prog/led.c, [2160](#)
- ARMCM0\_STM32F0\_Nucleo\_F091RC\_IAR/↔  
Prog/led.h, [2395](#)
- ARMCM0\_STM32F0\_Nucleo\_F091RC\_Keil/↔  
Prog/led.c, [2163](#)
- ARMCM0\_STM32F0\_Nucleo\_F091RC\_Keil/↔  
Prog/led.h, [2398](#)
- ARMCM0\_STM32G0\_Nucleo\_G071RB\_CubeID↔  
E/Prog/App/led.c, [2166](#)
- ARMCM0\_STM32G0\_Nucleo\_G071RB\_CubeID↔  
E/Prog/App/led.h, [2401](#)
- ARMCM0\_STM32G0\_Nucleo\_G071RB\_GCC/↔  
Prog/led.c, [2169](#)
- ARMCM0\_STM32G0\_Nucleo\_G071RB\_GCC/↔  
Prog/led.h, [2404](#)
- ARMCM0\_STM32G0\_Nucleo\_G071RB\_IAR/↔  
Prog/led.c, [2172](#)
- ARMCM0\_STM32G0\_Nucleo\_G071RB\_IAR/↔  
Prog/led.h, [2407](#)
- ARMCM0\_STM32G0\_Nucleo\_G071RB\_Keil/↔  
Prog/led.c, [2175](#)
- ARMCM0\_STM32G0\_Nucleo\_G071RB\_Keil/↔  
Prog/led.h, [2410](#)
- ARMCM0\_XMC1\_XMC1400\_Boot\_Kit\_GCC/↔  
Prog/led.c, [2178](#)
- ARMCM0\_XMC1\_XMC1400\_Boot\_Kit\_GCC/↔  
Prog/led.h, [2413](#)
- ARMCM0\_XMC1\_XMC1400\_Boot\_Kit\_IAR/↔  
Prog/led.c, [2181](#)
- ARMCM0\_XMC1\_XMC1400\_Boot\_Kit\_IAR/↔  
Prog/led.h, [2416](#)
- ARMCM33\_STM32L5\_Nucleo\_L552ZE\_CubeID↔  
E/Prog/App/led.c, [2184](#)
- ARMCM33\_STM32L5\_Nucleo\_L552ZE\_CubeID↔  
E/Prog/App/led.h, [2419](#)
- ARMCM33\_STM32L5\_Nucleo\_L552ZE\_GCC/↔  
Prog/led.c, [2187](#)
- ARMCM33\_STM32L5\_Nucleo\_L552ZE\_GCC/↔  
Prog/led.h, [2422](#)
- ARMCM33\_STM32L5\_Nucleo\_L552ZE\_IAR/↔  
Prog/led.c, [2190](#)
- ARMCM33\_STM32L5\_Nucleo\_L552ZE\_IAR/↔  
Prog/led.h, [2425](#)
- ARMCM33\_STM32L5\_Nucleo\_L552ZE\_Keil/↔  
Prog/led.c, [2193](#)
- ARMCM33\_STM32L5\_Nucleo\_L552ZE\_Keil/↔  
Prog/led.h, [2428](#)
- ARMCM3\_EFM32\_Olimex\_EM32G880F128ST↔  
K\_GCC/Prog/led.c, [2194](#)
- ARMCM3\_EFM32\_Olimex\_EM32G880F128ST↔  
K\_GCC/Prog/led.h, [2429](#)
- ARMCM3\_EFM32\_Olimex\_EM32G880F128ST↔  
K\_IAR/Prog/led.c, [2195](#)
- ARMCM3\_EFM32\_Olimex\_EM32G880F128ST↔  
K\_IAR/Prog/led.h, [2431](#)
- ARMCM3\_LM3S\_EK\_LM3S6965\_GCC/Prog/led.↔  
c, [2197](#)
- ARMCM3\_LM3S\_EK\_LM3S6965\_GCC/Prog/led.↔  
h, [2432](#)
- ARMCM3\_LM3S\_EK\_LM3S6965\_IAR/Prog/led.c,  
[2198](#)
- ARMCM3\_LM3S\_EK\_LM3S6965\_IAR/Prog/led.h,  
[2434](#)
- ARMCM3\_LM3S\_EK\_LM3S8962\_GCC/Prog/led.↔  
c, [2199](#)
- ARMCM3\_LM3S\_EK\_LM3S8962\_GCC/Prog/led.↔  
h, [2435](#)
- ARMCM3\_LM3S\_EK\_LM3S8962\_IAR/Prog/led.c,  
[2200](#)
- ARMCM3\_LM3S\_EK\_LM3S8962\_IAR/Prog/led.h,  
[2436](#)
- ARMCM3\_STM32F1\_Nucleo\_F103RB\_CubeID↔  
E/Prog/App/led.c, [2203](#)
- ARMCM3\_STM32F1\_Nucleo\_F103RB\_CubeID↔  
E/Prog/App/led.h, [2439](#)
- ARMCM3\_STM32F1\_Nucleo\_F103RB\_GCC/↔  
Prog/led.c, [2206](#)
- ARMCM3\_STM32F1\_Nucleo\_F103RB\_GCC/↔  
Prog/led.h, [2442](#)
- ARMCM3\_STM32F1\_Nucleo\_F103RB\_IAR/↔  
Prog/led.c, [2209](#)
- ARMCM3\_STM32F1\_Nucleo\_F103RB\_IAR/↔  
Prog/led.h, [2445](#)
- ARMCM3\_STM32F1\_Nucleo\_F103RB\_Keil/↔  
Prog/led.c, [2212](#)
- ARMCM3\_STM32F1\_Nucleo\_F103RB\_Keil/↔  
Prog/led.h, [2448](#)
- ARMCM3\_STM32F1\_Olimex\_STM32H103\_↔  
CubeIDE/Prog/App/led.c, [2216](#)
- ARMCM3\_STM32F1\_Olimex\_STM32H103\_↔  
CubeIDE/Prog/App/led.h, [2451](#)
- ARMCM3\_STM32F1\_Olimex\_STM32H103\_GC↔  
C/Prog/led.c, [2219](#)
- ARMCM3\_STM32F1\_Olimex\_STM32H103\_GC↔  
C/Prog/led.h, [2454](#)
- ARMCM3\_STM32F1\_Olimex\_STM32H103\_IA↔  
R/Prog/led.c, [2222](#)
- ARMCM3\_STM32F1\_Olimex\_STM32H103\_IA↔  
R/Prog/led.h, [2457](#)
- ARMCM3\_STM32F1\_Olimex\_STM32H103\_↔  
Keil/Prog/led.c, [2225](#)
- ARMCM3\_STM32F1\_Olimex\_STM32H103\_↔  
Keil/Prog/led.h, [2460](#)
- ARMCM3\_STM32F1\_Olimex\_STM32P103\_↔  
CubeIDE/Prog/App/led.c, [2228](#)
- ARMCM3\_STM32F1\_Olimex\_STM32P103\_↔  
CubeIDE/Prog/App/led.h, [2463](#)
- ARMCM3\_STM32F1\_Olimex\_STM32P103\_GC↔  
C/Prog/led.c, [2231](#)
- ARMCM3\_STM32F1\_Olimex\_STM32P103\_GC↔  
C/Prog/led.h, [2466](#)
- ARMCM3\_STM32F1\_Olimex\_STM32P103\_IA↔  
R/Prog/led.c, [2234](#)

- ARMCM3\_STM32F1\_Olimex\_STM32P103\_IA↔  
R/Prog/led.h, [2469](#)
- ARMCM3\_STM32F1\_Olimex\_STM32P103\_Keil/↔  
Prog/led.c, [2237](#)
- ARMCM3\_STM32F1\_Olimex\_STM32P103\_Keil/↔  
Prog/led.h, [2472](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_CubeI↔  
DE/Prog/App/led.c, [2240](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_CubeI↔  
DE/Prog/App/led.h, [2475](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_GCC/↔  
Prog/led.c, [2243](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_GCC/↔  
Prog/led.h, [2478](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_IAR/↔  
Prog/led.c, [2246](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_IAR/↔  
Prog/led.h, [2481](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_Keil/↔  
Prog/led.c, [2249](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_Keil/↔  
Prog/led.h, [2484](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_↔  
CubeIDE/Prog/App/led.c, [2252](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_↔  
CubeIDE/Prog/App/led.h, [2487](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_GC↔  
C/Prog/led.c, [2255](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_GC↔  
C/Prog/led.h, [2490](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_IA↔  
R/Prog/led.c, [2258](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_IA↔  
R/Prog/led.h, [2493](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_Keil/↔  
Prog/led.c, [2261](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_Keil/↔  
Prog/led.h, [2496](#)
- ARMCM4\_S32K14\_S32K144EVB\_GCC/Prog/led.↔  
c, [2264](#)
- ARMCM4\_S32K14\_S32K144EVB\_GCC/Prog/led.↔  
h, [2499](#)
- ARMCM4\_S32K14\_S32K144EVB\_IAR/Prog/led.↔  
c, [2267](#)
- ARMCM4\_S32K14\_S32K144EVB\_IAR/Prog/led.↔  
h, [2502](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_Cube↔  
IDE/Prog/App/led.c, [2270](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_Cube↔  
IDE/Prog/App/led.h, [2505](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_GCC/↔  
Prog/led.c, [2273](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_GCC/↔  
Prog/led.h, [2508](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_IAR/↔  
Prog/led.c, [2277](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_IAR/↔  
Prog/led.h, [2511](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_Keil/↔  
Prog/led.c, [2280](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_Keil/↔  
Prog/led.h, [2514](#)
- ARMCM4\_STM32F3\_Nucleo\_F303K8\_CubeID↔  
E/Prog/App/led.c, [2283](#)
- ARMCM4\_STM32F3\_Nucleo\_F303K8\_CubeID↔  
E/Prog/App/led.h, [2517](#)
- ARMCM4\_STM32F3\_Nucleo\_F303K8\_GCC/↔  
Prog/led.c, [2286](#)
- ARMCM4\_STM32F3\_Nucleo\_F303K8\_GCC/↔  
Prog/led.h, [2520](#)
- ARMCM4\_STM32F3\_Nucleo\_F303K8\_IAR/↔  
Prog/led.c, [2289](#)
- ARMCM4\_STM32F3\_Nucleo\_F303K8\_IAR/↔  
Prog/led.h, [2523](#)
- ARMCM4\_STM32F3\_Nucleo\_F303K8\_Keil/↔  
Prog/led.c, [2292](#)
- ARMCM4\_STM32F3\_Nucleo\_F303K8\_Keil/↔  
Prog/led.h, [2526](#)
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_CubeID↔  
E/Prog/App/led.c, [2295](#)
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_CubeID↔  
E/Prog/App/led.h, [2529](#)
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_GCC/↔  
Prog/led.c, [2297](#)
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_IAR/↔  
Prog/led.c, [2298](#)
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_Keil/↔  
Prog/led.c, [2299](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_↔  
CubeIDE/Prog/App/led.c, [2302](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_↔  
CubeIDE/Prog/App/led.h, [2537](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_GC↔  
C/Prog/led.c, [2305](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_GC↔  
C/Prog/led.h, [2540](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_IA↔  
R/Prog/led.c, [2308](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_IA↔  
R/Prog/led.h, [2543](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_Keil/↔  
Prog/led.c, [2311](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_Keil/↔  
Prog/led.h, [2546](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_CubeID↔  
E/Prog/App/led.c, [2314](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_CubeID↔  
E/Prog/App/led.h, [2549](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_GCC/↔  
Prog/led.c, [2317](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_GCC/↔  
Prog/led.h, [2552](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_IAR/↔  
Prog/led.c, [2320](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_IAR/↔  
Prog/led.h, [2555](#)

- ARMCM4\_STM32L4\_Nucleo\_L476RG\_Keil/↔  
Prog/led.c, [2323](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_Keil/↔  
Prog/led.h, [2558](#)
- ARMCM4\_TM4C\_DK\_TM4C123G\_IAR/Prog/led.↔  
c, [2325](#)
- ARMCM4\_TM4C\_DK\_TM4C123G\_IAR/Prog/led.↔  
h, [2559](#)
- ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_GCC/↔  
Prog/led.c, [2328](#)
- ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_GCC/↔  
Prog/led.h, [2562](#)
- ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_IAR/↔  
Prog/led.c, [2331](#)
- ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_IAR/↔  
Prog/led.h, [2565](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_CubeID↔  
E/Prog/App/led.c, [2334](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_CubeID↔  
E/Prog/App/led.h, [2568](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_GCC/↔  
Prog/led.c, [2337](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_GCC/↔  
Prog/led.h, [2571](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_IAR/↔  
Prog/led.c, [2340](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_IAR/↔  
Prog/led.h, [2574](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_Keil/↔  
Prog/led.c, [2343](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_Keil/↔  
Prog/led.h, [2577](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_CubeID↔  
E/Prog/App/led.c, [2346](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_CubeID↔  
E/Prog/App/led.h, [2580](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_GCC/↔  
Prog/led.c, [2349](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_GCC/↔  
Prog/led.h, [2583](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_IAR/↔  
Prog/led.c, [2352](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_IAR/↔  
Prog/led.h, [2586](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_Keil/↔  
Prog/led.c, [2355](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_Keil/↔  
Prog/led.h, [2589](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_CubeID↔  
E/Prog/App/led.c, [2358](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_CubeID↔  
E/Prog/App/led.h, [2592](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_GCC/↔  
Prog/led.c, [2361](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_GCC/↔  
Prog/led.h, [2595](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_IAR/↔  
Prog/led.c, [2364](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_IAR/↔  
Prog/led.h, [2598](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_Keil/↔  
Prog/led.c, [2367](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_Keil/↔  
Prog/led.h, [2601](#)
- HCS12\_DevKit\_S12G128\_CodeWarrior/Prog/led.↔  
c, [2370](#)
- HCS12\_DevKit\_S12G128\_CodeWarrior/Prog/led.↔  
h, [2604](#)
- HCS12\_Evbplus\_Dragon12p\_CodeWarrior/↔  
Prog/led.c, [2371](#)
- HCS12\_Evbplus\_Dragon12p\_CodeWarrior/↔  
Prog/led.h, [2605](#)
- length  
tFifoCtrl, [367](#)
- line  
tSrecLineParseObject, [379](#)
- main  
\_template/Boot/main.c, [2606](#)  
\_template/Prog/main.c, [2608](#)  
ARMCM0\_STM32F0\_Discovery\_STM32F051\_↔  
GCC/Boot/main.c, [2610](#)  
ARMCM0\_STM32F0\_Discovery\_STM32F051\_↔  
GCC/Prog/main.c, [2612](#)  
ARMCM0\_STM32F0\_Discovery\_STM32F051\_I↔  
AR/Boot/main.c, [2614](#)  
ARMCM0\_STM32F0\_Discovery\_STM32F051\_I↔  
AR/Prog/main.c, [2617](#)  
ARMCM0\_STM32F0\_Discovery\_STM32F051\_↔  
Keil/Boot/main.c, [2619](#)  
ARMCM0\_STM32F0\_Discovery\_STM32F051\_↔  
Keil/Prog/main.c, [2621](#)  
ARMCM0\_STM32F0\_Nucleo\_F091RC\_GCC/↔  
Boot/main.c, [2624](#)  
ARMCM0\_STM32F0\_Nucleo\_F091RC\_GCC/↔  
Prog/main.c, [2626](#)  
ARMCM0\_STM32F0\_Nucleo\_F091RC\_IAR/↔  
Boot/main.c, [2628](#)  
ARMCM0\_STM32F0\_Nucleo\_F091RC\_IAR/↔  
Prog/main.c, [2631](#)  
ARMCM0\_STM32F0\_Nucleo\_F091RC\_Keil/↔  
Boot/main.c, [2633](#)  
ARMCM0\_STM32F0\_Nucleo\_F091RC\_Keil/↔  
Prog/main.c, [2635](#)  
ARMCM0\_STM32G0\_Nucleo\_G071RB\_GCC/↔  
Boot/main.c, [2638](#)  
ARMCM0\_STM32G0\_Nucleo\_G071RB\_GCC/↔  
Prog/main.c, [2640](#)  
ARMCM0\_STM32G0\_Nucleo\_G071RB\_IAR/↔  
Boot/main.c, [2643](#)  
ARMCM0\_STM32G0\_Nucleo\_G071RB\_IAR/↔  
Prog/main.c, [2645](#)  
ARMCM0\_STM32G0\_Nucleo\_G071RB\_Keil/↔  
Boot/main.c, [2648](#)  
ARMCM0\_STM32G0\_Nucleo\_G071RB\_Keil/↔  
Prog/main.c, [2650](#)



- ARMCM0\_XMC1\_XMC1400\_Boot\_Kit\_GCC/↔  
Boot/main.c, [2652](#)
- ARMCM0\_XMC1\_XMC1400\_Boot\_Kit\_GCC/↔  
Prog/main.c, [2653](#)
- ARMCM0\_XMC1\_XMC1400\_Boot\_Kit\_IAR/↔  
Boot/main.c, [2655](#)
- ARMCM0\_XMC1\_XMC1400\_Boot\_Kit\_IAR/↔  
Prog/main.c, [2656](#)
- ARMCM33\_STM32L5\_Nucleo\_L552ZE\_GCC/↔  
Boot/main.c, [2658](#)
- ARMCM33\_STM32L5\_Nucleo\_L552ZE\_GCC/↔  
Prog/main.c, [2661](#)
- ARMCM33\_STM32L5\_Nucleo\_L552ZE\_IAR/↔  
Boot/main.c, [2663](#)
- ARMCM33\_STM32L5\_Nucleo\_L552ZE\_IAR/↔  
Prog/main.c, [2666](#)
- ARMCM33\_STM32L5\_Nucleo\_L552ZE\_Keil/↔  
Boot/main.c, [2668](#)
- ARMCM33\_STM32L5\_Nucleo\_L552ZE\_Keil/↔  
Prog/main.c, [2671](#)
- ARMCM3\_EFM32\_Olimex\_EM32G880F128ST↔  
K\_GCC/Boot/cstart.c, [1247](#)
- ARMCM3\_EFM32\_Olimex\_EM32G880F128ST↔  
K\_GCC/Boot/main.c, [2673](#)
- ARMCM3\_EFM32\_Olimex\_EM32G880F128ST↔  
K\_GCC/Prog/cstart.c, [1249](#)
- ARMCM3\_EFM32\_Olimex\_EM32G880F128ST↔  
K\_GCC/Prog/main.c, [2674](#)
- ARMCM3\_EFM32\_Olimex\_EM32G880F128ST↔  
K\_IAR/Boot/main.c, [2675](#)
- ARMCM3\_EFM32\_Olimex\_EM32G880F128ST↔  
K\_IAR/Prog/main.c, [2677](#)
- ARMCM3\_LM3S\_EK\_LM3S6965\_GCC/Boot/cstart.↔  
c, [1250](#)
- ARMCM3\_LM3S\_EK\_LM3S6965\_GCC/Boot/main.↔  
c, [2678](#)
- ARMCM3\_LM3S\_EK\_LM3S6965\_GCC/Prog/cstart.↔  
c, [1251](#)
- ARMCM3\_LM3S\_EK\_LM3S6965\_GCC/Prog/main.↔  
c, [2679](#)
- ARMCM3\_LM3S\_EK\_LM3S6965\_IAR/Boot/main.↔  
c, [2681](#)
- ARMCM3\_LM3S\_EK\_LM3S6965\_IAR/Prog/main.↔  
c, [2682](#)
- ARMCM3\_LM3S\_EK\_LM3S8962\_GCC/Boot/cstart.↔  
c, [1252](#)
- ARMCM3\_LM3S\_EK\_LM3S8962\_GCC/Boot/main.↔  
c, [2683](#)
- ARMCM3\_LM3S\_EK\_LM3S8962\_GCC/Prog/cstart.↔  
c, [1254](#)
- ARMCM3\_LM3S\_EK\_LM3S8962\_GCC/Prog/main.↔  
c, [2684](#)
- ARMCM3\_LM3S\_EK\_LM3S8962\_IAR/Boot/main.↔  
c, [2686](#)
- ARMCM3\_LM3S\_EK\_LM3S8962\_IAR/Prog/main.↔  
c, [2687](#)
- ARMCM3\_STM32F1\_Nucleo\_F103RB\_GCC/↔  
Boot/main.c, [2689](#)
- ARMCM3\_STM32F1\_Nucleo\_F103RB\_GCC/↔  
Prog/main.c, [2691](#)
- ARMCM3\_STM32F1\_Nucleo\_F103RB\_IAR/↔  
Boot/main.c, [2694](#)
- ARMCM3\_STM32F1\_Nucleo\_F103RB\_IAR/↔  
Prog/main.c, [2696](#)
- ARMCM3\_STM32F1\_Nucleo\_F103RB\_Keil/↔  
Boot/main.c, [2699](#)
- ARMCM3\_STM32F1\_Nucleo\_F103RB\_Keil/↔  
Prog/main.c, [2701](#)
- ARMCM3\_STM32F1\_Olimex\_STM32H103\_GC↔  
C/Boot/main.c, [2704](#)
- ARMCM3\_STM32F1\_Olimex\_STM32H103\_GC↔  
C/Prog/main.c, [2706](#)
- ARMCM3\_STM32F1\_Olimex\_STM32H103\_IA↔  
R/Boot/main.c, [2709](#)
- ARMCM3\_STM32F1\_Olimex\_STM32H103\_IA↔  
R/Prog/main.c, [2711](#)
- ARMCM3\_STM32F1\_Olimex\_STM32H103\_↔  
Keil/Boot/main.c, [2714](#)
- ARMCM3\_STM32F1\_Olimex\_STM32H103\_↔  
Keil/Prog/main.c, [2716](#)
- ARMCM3\_STM32F1\_Olimex\_STM32P103\_GC↔  
C/Boot/main.c, [2719](#)
- ARMCM3\_STM32F1\_Olimex\_STM32P103\_GC↔  
C/Prog/main.c, [2721](#)
- ARMCM3\_STM32F1\_Olimex\_STM32P103\_IA↔  
R/Boot/main.c, [2724](#)
- ARMCM3\_STM32F1\_Olimex\_STM32P103\_IA↔  
R/Prog/main.c, [2726](#)
- ARMCM3\_STM32F1\_Olimex\_STM32P103\_Keil/↔  
Boot/main.c, [2729](#)
- ARMCM3\_STM32F1\_Olimex\_STM32P103\_Keil/↔  
Prog/main.c, [2731](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_GCC/↔  
Boot/main.c, [2734](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_GCC/↔  
Prog/main.c, [2736](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_IAR/↔  
Boot/main.c, [2739](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_IAR/↔  
Prog/main.c, [2741](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_Keil/↔  
Boot/main.c, [2744](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_Keil/↔  
Prog/main.c, [2746](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_GC↔  
C/Boot/main.c, [2749](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_GC↔  
C/Prog/main.c, [2751](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_IA↔  
R/Boot/main.c, [2754](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_IA↔  
R/Prog/main.c, [2756](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_Keil/↔  
Boot/main.c, [2759](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_Keil/↔  
Prog/main.c, [2761](#)

- ARMCM4\_S32K14\_S32K144EVB\_GCC/Boot/main.c, [2763](#)
- ARMCM4\_S32K14\_S32K144EVB\_GCC/Prog/main.c, [2765](#)
- ARMCM4\_S32K14\_S32K144EVB\_IAR/Boot/main.c, [2766](#)
- ARMCM4\_S32K14\_S32K144EVB\_IAR/Prog/main.c, [2768](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_GCC/Boot/main.c, [2771](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_GCC/Prog/main.c, [2773](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_IAR/Boot/main.c, [2776](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_IAR/Prog/main.c, [2779](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_Keil/Boot/main.c, [2781](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_Keil/Prog/main.c, [2784](#)
- ARMCM4\_STM32F3\_Nucleo\_F303K8\_GCC/Boot/main.c, [2786](#)
- ARMCM4\_STM32F3\_Nucleo\_F303K8\_GCC/Prog/main.c, [2789](#)
- ARMCM4\_STM32F3\_Nucleo\_F303K8\_IAR/Boot/main.c, [2791](#)
- ARMCM4\_STM32F3\_Nucleo\_F303K8\_IAR/Prog/main.c, [2794](#)
- ARMCM4\_STM32F3\_Nucleo\_F303K8\_Keil/Boot/main.c, [2796](#)
- ARMCM4\_STM32F3\_Nucleo\_F303K8\_Keil/Prog/main.c, [2799](#)
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_GCC/Boot/main.c, [2801](#)
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_GCC/Prog/main.c, [2804](#)
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_IAR/Boot/main.c, [2806](#)
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_IAR/Prog/main.c, [2809](#)
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_Keil/Boot/main.c, [2811](#)
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_Keil/Prog/main.c, [2814](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_GC/Boot/main.c, [2816](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_GC/Prog/main.c, [2819](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_IAR/Boot/main.c, [2821](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_IAR/Prog/main.c, [2824](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_Keil/Boot/main.c, [2826](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_Keil/Prog/main.c, [2829](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_GCC/Boot/main.c, [2831](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_GCC/Prog/main.c, [2834](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_IAR/Boot/main.c, [2836](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_IAR/Prog/main.c, [2839](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_Keil/Boot/main.c, [2841](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_Keil/Prog/main.c, [2844](#)
- ARMCM4\_TM4C\_DK\_TM4C123G\_IAR/Boot/main.c, [2846](#)
- ARMCM4\_TM4C\_DK\_TM4C123G\_IAR/Prog/main.c, [2847](#)
- ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_GCC/Boot/main.c, [2848](#)
- ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_GCC/Prog/main.c, [2850](#)
- ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_IAR/Boot/main.c, [2851](#)
- ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_IAR/Prog/main.c, [2853](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_GCC/Boot/main.c, [2855](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_GCC/Prog/main.c, [2857](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_IAR/Boot/main.c, [2860](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_IAR/Prog/main.c, [2862](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_Keil/Boot/main.c, [2865](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_Keil/Prog/main.c, [2867](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_GCC/Boot/main.c, [2870](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_GCC/Prog/main.c, [2872](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_IAR/Boot/main.c, [2875](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_IAR/Prog/main.c, [2877](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_Keil/Boot/main.c, [2880](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_Keil/Prog/main.c, [2882](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_GCC/Boot/main.c, [2885](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_GCC/Prog/main.c, [2887](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_IAR/Boot/main.c, [2890](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_IAR/Prog/main.c, [2892](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_Keil/Boot/main.c, [2895](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_Keil/Prog/main.c, [2897](#)

- HCS12\_DevKit\_S12G128\_CodeWarrior/Boot/main.c, [2899](#)
- HCS12\_DevKit\_S12G128\_CodeWarrior/Prog/main.c, [2901](#)
- HCS12\_Evbplus\_Dragon12p\_CodeWarrior/Boot/main.c, [2902](#)
- HCS12\_Evbplus\_Dragon12p\_CodeWarrior/Boot/vectors.c, [3669](#)
- HCS12\_Evbplus\_Dragon12p\_CodeWarrior/Prog/main.c, [2904](#)
- main.c, [2606–2608](#), [2611](#), [2613](#), [2615](#), [2618](#), [2620](#), [2622](#), [2625](#), [2627](#), [2629](#), [2632](#), [2634](#), [2636](#), [2639](#), [2641](#), [2644](#), [2646](#), [2649](#), [2651](#), [2653](#), [2654](#), [2656](#), [2657](#), [2659](#), [2662](#), [2664](#), [2667](#), [2669](#), [2672](#), [2673](#), [2675–2677](#), [2679–2682](#), [2684–2687](#), [2690](#), [2692](#), [2695](#), [2697](#), [2700](#), [2702](#), [2705](#), [2707](#), [2710](#), [2712](#), [2715](#), [2717](#), [2720](#), [2722](#), [2725](#), [2727](#), [2730](#), [2732](#), [2735](#), [2737](#), [2740](#), [2742](#), [2745](#), [2747](#), [2750](#), [2752](#), [2755](#), [2757](#), [2760](#), [2762](#), [2764](#), [2766](#), [2767](#), [2769](#), [2772](#), [2775](#), [2777](#), [2780](#), [2782](#), [2785](#), [2787](#), [2790](#), [2792](#), [2795](#), [2797](#), [2800](#), [2802](#), [2805](#), [2807](#), [2810](#), [2812](#), [2815](#), [2817](#), [2820](#), [2822](#), [2825](#), [2827](#), [2830](#), [2832](#), [2835](#), [2837](#), [2840](#), [2842](#), [2845–2847](#), [2849](#), [2850](#), [2852](#), [2853](#), [2856](#), [2858](#), [2861](#), [2863](#), [2866](#), [2868](#), [2871](#), [2873](#), [2876](#), [2878](#), [2881](#), [2883](#), [2886](#), [2888](#), [2891](#), [2893](#), [2896](#), [2898](#), [2900](#), [2902](#), [2903](#)
- mta
- txcplInfo, [385](#)
- net.c, [2905](#), [2907](#), [2909](#), [2911](#), [2913](#), [2915](#), [2917](#), [2919](#), [2921](#), [2923](#), [2925](#), [2927](#), [2929](#)
- net.h, [2932](#), [2934](#), [2936](#), [2938](#), [2940](#), [2942](#), [2944](#), [2946](#), [2948](#), [2950](#), [2952](#), [2954](#), [2956](#)
- NetApp
- Demo/ARMCM3\_LM3S\_EK\_LM3S6965\_GCC/Prog/net.c, [2906](#)
- Demo/ARMCM3\_LM3S\_EK\_LM3S6965\_GCC/Prog/net.h, [2932](#)
- Demo/ARMCM3\_LM3S\_EK\_LM3S6965\_IAR/Prog/net.c, [2908](#)
- Demo/ARMCM3\_LM3S\_EK\_LM3S6965\_IAR/Prog/net.h, [2934](#)
- Demo/ARMCM4\_STM32F4\_Nucleo\_F429ZI\_CubeIDE/Prog/App/net.c, [2910](#)
- Demo/ARMCM4\_STM32F4\_Nucleo\_F429ZI\_CubeIDE/Prog/App/net.h, [2936](#)
- Demo/ARMCM4\_STM32F4\_Nucleo\_F429ZI\_GCC/Prog/net.c, [2912](#)
- Demo/ARMCM4\_STM32F4\_Nucleo\_F429ZI\_GCC/Prog/net.h, [2938](#)
- Demo/ARMCM4\_STM32F4\_Nucleo\_F429ZI\_IA\_R/Prog/net.c, [2914](#)
- Demo/ARMCM4\_STM32F4\_Nucleo\_F429ZI\_IA\_R/Prog/net.h, [2940](#)
- Demo/ARMCM4\_STM32F4\_Nucleo\_F429ZI\_Keil/Prog/net.c, [2916](#)
- Demo/ARMCM4\_STM32F4\_Nucleo\_F429ZI\_Keil/Prog/net.h, [2942](#)
- Demo/ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_GCC/Prog/net.c, [2918](#)
- Demo/ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_GCC/Prog/net.h, [2944](#)
- Demo/ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_IAR/Prog/net.c, [2920](#)
- Demo/ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_IAR/Prog/net.h, [2946](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F767ZI\_CubeIDE/Prog/App/net.c, [2922](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F767ZI\_CubeIDE/Prog/App/net.h, [2948](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F767ZI\_GCC/Prog/net.c, [2924](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F767ZI\_GCC/Prog/net.h, [2950](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F767ZI\_IAR/Prog/net.c, [2926](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F767ZI\_IAR/Prog/net.h, [2952](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F767ZI\_Keil/Prog/net.c, [2928](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F767ZI\_Keil/Prog/net.h, [2954](#)
- Source/net.c, [2930](#)
- Source/net.h, [2956](#)
- NetInit
- Demo/ARMCM3\_LM3S\_EK\_LM3S6965\_GCC/Prog/net.c, [2906](#)
- Demo/ARMCM3\_LM3S\_EK\_LM3S6965\_GCC/Prog/net.h, [2933](#)
- Demo/ARMCM3\_LM3S\_EK\_LM3S6965\_IAR/Prog/net.c, [2908](#)
- Demo/ARMCM3\_LM3S\_EK\_LM3S6965\_IAR/Prog/net.h, [2935](#)
- Demo/ARMCM4\_STM32F4\_Nucleo\_F429ZI\_CubeIDE/Prog/App/net.c, [2910](#)
- Demo/ARMCM4\_STM32F4\_Nucleo\_F429ZI\_CubeIDE/Prog/App/net.h, [2937](#)
- Demo/ARMCM4\_STM32F4\_Nucleo\_F429ZI\_GCC/Prog/net.c, [2912](#)
- Demo/ARMCM4\_STM32F4\_Nucleo\_F429ZI\_GCC/Prog/net.h, [2939](#)
- Demo/ARMCM4\_STM32F4\_Nucleo\_F429ZI\_IAR/Prog/net.c, [2914](#)
- Demo/ARMCM4\_STM32F4\_Nucleo\_F429ZI\_IAR/Prog/net.h, [2941](#)
- Demo/ARMCM4\_STM32F4\_Nucleo\_F429ZI\_Keil/Prog/net.c, [2916](#)
- Demo/ARMCM4\_STM32F4\_Nucleo\_F429ZI\_Keil/Prog/net.h, [2943](#)
- Demo/ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_GCC/Prog/net.c, [2918](#)
- Demo/ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_GCC/Prog/net.h, [2945](#)
- Demo/ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_IAR/Prog/net.c, [2920](#)
- Demo/ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_IAR/Prog/net.h, [2946](#)

- AR/Prog/net.c, [2920](#)
- Demo/ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_I↔  
AR/Prog/net.h, [2947](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F767ZI\_↔  
CubeIDE/Prog/App/net.c, [2922](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F767ZI\_↔  
CubeIDE/Prog/App/net.h, [2949](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F767ZI\_G↔  
CC/Prog/net.c, [2924](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F767ZI\_G↔  
CC/Prog/net.h, [2951](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F767ZI\_IA↔  
R/Prog/net.c, [2926](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F767ZI\_IA↔  
R/Prog/net.h, [2953](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F767ZI\_↔  
Keil/Prog/net.c, [2928](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F767ZI\_↔  
Keil/Prog/net.h, [2955](#)
- Source/net.c, [2930](#)
- Source/net.h, [2957](#)
- NetReceivePacket
  - Source/net.c, [2930](#)
  - Source/net.h, [2957](#)
- NetServerTask
  - Source/net.c, [2931](#)
- NetTask
  - Demo/ARMCM3\_LM3S\_EK\_LM3S6965\_GCC/↔  
Prog/net.c, [2906](#)
  - Demo/ARMCM3\_LM3S\_EK\_LM3S6965\_GCC/↔  
Prog/net.h, [2933](#)
  - Demo/ARMCM3\_LM3S\_EK\_LM3S6965\_IAR/↔  
Prog/net.c, [2908](#)
  - Demo/ARMCM3\_LM3S\_EK\_LM3S6965\_IAR/↔  
Prog/net.h, [2935](#)
  - Demo/ARMCM4\_STM32F4\_Nucleo\_F429ZI\_↔  
CubeIDE/Prog/App/net.c, [2910](#)
  - Demo/ARMCM4\_STM32F4\_Nucleo\_F429ZI\_↔  
CubeIDE/Prog/App/net.h, [2937](#)
  - Demo/ARMCM4\_STM32F4\_Nucleo\_F429ZI\_G↔  
CC/Prog/net.c, [2912](#)
  - Demo/ARMCM4\_STM32F4\_Nucleo\_F429ZI\_G↔  
CC/Prog/net.h, [2939](#)
  - Demo/ARMCM4\_STM32F4\_Nucleo\_F429ZI\_IA↔  
R/Prog/net.c, [2914](#)
  - Demo/ARMCM4\_STM32F4\_Nucleo\_F429ZI\_IA↔  
R/Prog/net.h, [2941](#)
  - Demo/ARMCM4\_STM32F4\_Nucleo\_F429ZI\_↔  
Keil/Prog/net.c, [2916](#)
  - Demo/ARMCM4\_STM32F4\_Nucleo\_F429ZI\_↔  
Keil/Prog/net.h, [2943](#)
  - Demo/ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_↔  
GCC/Prog/net.c, [2918](#)
  - Demo/ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_↔  
GCC/Prog/net.h, [2945](#)
  - Demo/ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_I↔  
AR/Prog/net.c, [2920](#)
  - Demo/ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_I↔
  - AR/Prog/net.h, [2947](#)
  - Demo/ARMCM7\_STM32F7\_Nucleo\_F767ZI\_↔  
CubeIDE/Prog/App/net.c, [2922](#)
  - Demo/ARMCM7\_STM32F7\_Nucleo\_F767ZI\_↔  
CubeIDE/Prog/App/net.h, [2949](#)
  - Demo/ARMCM7\_STM32F7\_Nucleo\_F767ZI\_G↔  
CC/Prog/net.c, [2924](#)
  - Demo/ARMCM7\_STM32F7\_Nucleo\_F767ZI\_G↔  
CC/Prog/net.h, [2951](#)
  - Demo/ARMCM7\_STM32F7\_Nucleo\_F767ZI\_IA↔  
R/Prog/net.c, [2926](#)
  - Demo/ARMCM7\_STM32F7\_Nucleo\_F767ZI\_IA↔  
R/Prog/net.h, [2953](#)
  - Demo/ARMCM7\_STM32F7\_Nucleo\_F767ZI\_↔  
Keil/Prog/net.c, [2928](#)
  - Demo/ARMCM7\_STM32F7\_Nucleo\_F767ZI\_↔  
Keil/Prog/net.h, [2955](#)
  - NetTransmitPacket
    - Source/net.c, [2931](#)
    - Source/net.h, [2958](#)
  - Non-volatile memory driver of a port, [327](#)
  - nvm.c, [2958](#), [2964](#), [2969](#), [2975](#), [2980](#), [2985](#), [2990](#),  
[2995](#), [3000](#), [3005](#), [3010](#), [3015](#), [3020](#), [3025](#),  
[3030](#), [3035](#), [3040](#), [3045](#), [3050](#)
  - nvm.h, [3055](#)
    - NvmDone, [3056](#)
    - NvmErase, [3056](#)
    - NvmGetUserProgBaseAddress, [3056](#)
    - NvmInit, [3057](#)
    - NvmReinit, [3057](#)
    - NvmVerifyChecksum, [3057](#)
    - NvmWrite, [3057](#)
  - NvmDone
    - \_template/nvm.c, [2959](#)
    - ARMCM0\_S32K11/nvm.c, [2965](#)
    - ARMCM0\_STM32F0/nvm.c, [2971](#)
    - ARMCM0\_STM32G0/nvm.c, [2976](#)
    - ARMCM0\_XMC1/nvm.c, [2981](#)
    - ARMCM33\_STM32L5/nvm.c, [2986](#)
    - ARMCM3\_EFM32/nvm.c, [2991](#)
    - ARMCM3\_LM3S/nvm.c, [2996](#)
    - ARMCM3\_STM32F1/nvm.c, [3001](#)
    - ARMCM3\_STM32F2/nvm.c, [3006](#)
    - ARMCM4\_S32K14/nvm.c, [3011](#)
    - ARMCM4\_STM32F3/nvm.c, [3016](#)
    - ARMCM4\_STM32F4/nvm.c, [3021](#)
    - ARMCM4\_STM32L4/nvm.c, [3026](#)
    - ARMCM4\_TM4C/nvm.c, [3031](#)
    - ARMCM4\_XMC4/nvm.c, [3036](#)
    - ARMCM7\_STM32F7/nvm.c, [3041](#)
    - ARMCM7\_STM32H7/nvm.c, [3046](#)
    - HCS12/nvm.c, [3051](#)
    - nvm.h, [3056](#)
  - NvmDoneHook
    - \_template/Boot/hooks.c, [1622](#)
    - \_template/nvm.c, [2960](#)
    - ARMCM0\_S32K11/nvm.c, [2965](#)
    - ARMCM0\_STM32F0/nvm.c, [2971](#)

- ARMCM0\_STM32F0\_Discovery\_STM32F051\_↔  
CubeIDE/Boot/App/hooks.c, [1627](#)
- ARMCM0\_STM32F0\_Discovery\_STM32F051\_↔  
GCC/Boot/hooks.c, [1632](#)
- ARMCM0\_STM32F0\_Discovery\_STM32F051\_I↔  
AR/Boot/hooks.c, [1637](#)
- ARMCM0\_STM32F0\_Discovery\_STM32F051\_↔  
Keil/Boot/hooks.c, [1642](#)
- ARMCM0\_STM32F0\_Nucleo\_F091RC\_CubeID↔  
E/Boot/App/hooks.c, [1647](#)
- ARMCM0\_STM32F0\_Nucleo\_F091RC\_GCC/↔  
Boot/hooks.c, [1652](#)
- ARMCM0\_STM32F0\_Nucleo\_F091RC\_IAR/↔  
Boot/hooks.c, [1657](#)
- ARMCM0\_STM32F0\_Nucleo\_F091RC\_Keil/↔  
Boot/hooks.c, [1662](#)
- ARMCM0\_STM32G0/nvm.c, [2976](#)
- ARMCM0\_STM32G0\_Nucleo\_G071RB\_CubeID↔  
E/Boot/App/hooks.c, [1667](#)
- ARMCM0\_STM32G0\_Nucleo\_G071RB\_GCC/↔  
Boot/hooks.c, [1672](#)
- ARMCM0\_STM32G0\_Nucleo\_G071RB\_IAR/↔  
Boot/hooks.c, [1677](#)
- ARMCM0\_STM32G0\_Nucleo\_G071RB\_Keil/↔  
Boot/hooks.c, [1682](#)
- ARMCM0\_XMC1/nvm.c, [2981](#)
- ARMCM0\_XMC1\_XMC1400\_Boot\_Kit\_GCC/↔  
Boot/hooks.c, [1687](#)
- ARMCM0\_XMC1\_XMC1400\_Boot\_Kit\_IAR/↔  
Boot/hooks.c, [1692](#)
- ARMCM33\_STM32L5/nvm.c, [2986](#)
- ARMCM33\_STM32L5\_Nucleo\_L552ZE\_CubeID↔  
E/Boot/App/hooks.c, [1697](#)
- ARMCM33\_STM32L5\_Nucleo\_L552ZE\_GCC/↔  
Boot/hooks.c, [1703](#)
- ARMCM33\_STM32L5\_Nucleo\_L552ZE\_IAR/↔  
Boot/hooks.c, [1709](#)
- ARMCM33\_STM32L5\_Nucleo\_L552ZE\_Keil/↔  
Boot/hooks.c, [1715](#)
- ARMCM3\_EFM32/nvm.c, [2991](#)
- ARMCM3\_EFM32\_Olimex\_EM32G880F128ST↔  
K\_GCC/Boot/hooks.c, [1721](#)
- ARMCM3\_EFM32\_Olimex\_EM32G880F128ST↔  
K\_IAR/Boot/hooks.c, [1726](#)
- ARMCM3\_LM3S/nvm.c, [2996](#)
- ARMCM3\_LM3S\_EK\_LM3S6965\_GCC/Boot/hooks.↔  
c, [1733](#)
- ARMCM3\_LM3S\_EK\_LM3S6965\_IAR/Boot/hooks.↔  
c, [1741](#)
- ARMCM3\_LM3S\_EK\_LM3S8962\_GCC/Boot/hooks.↔  
c, [1747](#)
- ARMCM3\_LM3S\_EK\_LM3S8962\_IAR/Boot/hooks.↔  
c, [1752](#)
- ARMCM3\_STM32F1/nvm.c, [3001](#)
- ARMCM3\_STM32F1\_Nucleo\_F103RB\_CubeID↔  
E/Boot/App/hooks.c, [1757](#)
- ARMCM3\_STM32F1\_Nucleo\_F103RB\_GCC/↔  
Boot/hooks.c, [1762](#)
- ARMCM3\_STM32F1\_Nucleo\_F103RB\_IAR/↔  
Boot/hooks.c, [1767](#)
- ARMCM3\_STM32F1\_Nucleo\_F103RB\_Keil/↔  
Boot/hooks.c, [1772](#)
- ARMCM3\_STM32F1\_Olimex\_STM32H103\_↔  
CubeIDE/Boot/App/hooks.c, [1777](#)
- ARMCM3\_STM32F1\_Olimex\_STM32H103\_GC↔  
C/Boot/hooks.c, [1783](#)
- ARMCM3\_STM32F1\_Olimex\_STM32H103\_IA↔  
R/Boot/hooks.c, [1789](#)
- ARMCM3\_STM32F1\_Olimex\_STM32H103\_↔  
Keil/Boot/hooks.c, [1795](#)
- ARMCM3\_STM32F1\_Olimex\_STM32P103\_↔  
CubeIDE/Boot/App/hooks.c, [1803](#)
- ARMCM3\_STM32F1\_Olimex\_STM32P103\_GC↔  
C/Boot/hooks.c, [1811](#)
- ARMCM3\_STM32F1\_Olimex\_STM32P103\_IA↔  
R/Boot/hooks.c, [1818](#)
- ARMCM3\_STM32F1\_Olimex\_STM32P103\_Keil/↔  
Boot/hooks.c, [1826](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_Cube↔  
DE/Boot/App/hooks.c, [1834](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_GCC/↔  
Boot/hooks.c, [1842](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_IAR/↔  
Boot/hooks.c, [1851](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_Keil/↔  
Boot/hooks.c, [1860](#)
- ARMCM3\_STM32F2/nvm.c, [3006](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_↔  
CubeIDE/Boot/App/hooks.c, [1869](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_GC↔  
C/Boot/hooks.c, [1877](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_IA↔  
R/Boot/hooks.c, [1884](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_Keil/↔  
Boot/hooks.c, [1892](#)
- ARMCM4\_S32K14/nvm.c, [3011](#)
- ARMCM4\_S32K14\_S32K144EVB\_GCC/Boot/hooks.↔  
c, [1898](#)
- ARMCM4\_S32K14\_S32K144EVB\_IAR/Boot/hooks.↔  
c, [1903](#)
- ARMCM4\_STM32F3/nvm.c, [3016](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_Cube↔  
IDE/Boot/App/hooks.c, [1908](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_GCC/↔  
Boot/hooks.c, [1914](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_IAR/↔  
Boot/hooks.c, [1920](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_Keil/↔  
Boot/hooks.c, [1926](#)
- ARMCM4\_STM32F3\_Nucleo\_F303K8\_CubeID↔  
E/Boot/App/hooks.c, [1932](#)
- ARMCM4\_STM32F3\_Nucleo\_F303K8\_GCC/↔  
Boot/hooks.c, [1937](#)
- ARMCM4\_STM32F3\_Nucleo\_F303K8\_IAR/↔  
Boot/hooks.c, [1942](#)
- ARMCM4\_STM32F3\_Nucleo\_F303K8\_Keil/↔



- Boot/hooks.c, [1947](#)
- ARMCM4\_STM32F4\_nvmm.c, [3021](#)
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_CubeID↔  
E/Boot/App/hooks.c, [1952](#)
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_GCC/↔  
Boot/hooks.c, [1958](#)
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_IAR/↔  
Boot/hooks.c, [1964](#)
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_Keil/↔  
Boot/hooks.c, [1970](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_↔  
CubeIDE/Boot/App/hooks.c, [1978](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_GC↔  
C/Boot/hooks.c, [1986](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_IA↔  
R/Boot/hooks.c, [1995](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_Keil/↔  
Boot/hooks.c, [2004](#)
- ARMCM4\_STM32L4\_nvmm.c, [3026](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_CubeID↔  
E/Boot/App/hooks.c, [2011](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_GCC/↔  
Boot/hooks.c, [2016](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_IAR/↔  
Boot/hooks.c, [2021](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_Keil/↔  
Boot/hooks.c, [2026](#)
- ARMCM4\_TM4C\_nvmm.c, [3031](#)
- ARMCM4\_TM4C\_DK\_TM4C123G\_IAR/Boot/hooks.↔  
c, [2033](#)
- ARMCM4\_XMC4\_nvmm.c, [3036](#)
- ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_GCC/↔  
Boot/hooks.c, [2042](#)
- ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_IAR/↔  
Boot/hooks.c, [2050](#)
- ARMCM7\_STM32F7\_nvmm.c, [3041](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_CubeID↔  
E/Boot/App/hooks.c, [2056](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_GCC/↔  
Boot/hooks.c, [2062](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_IAR/↔  
Boot/hooks.c, [2068](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_Keil/↔  
Boot/hooks.c, [2074](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_CubeID↔  
E/Boot/App/hooks.c, [2080](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_GCC/↔  
Boot/hooks.c, [2085](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_IAR/↔  
Boot/hooks.c, [2090](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_Keil/↔  
Boot/hooks.c, [2095](#)
- ARMCM7\_STM32H7\_nvmm.c, [3046](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_CubeID↔  
E/Boot/App/hooks.c, [2100](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_GCC/↔  
Boot/hooks.c, [2106](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_IAR/↔  
Boot/hooks.c, [2112](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_Keil/↔  
Boot/hooks.c, [2118](#)
- HCS12\_nvmm.c, [3051](#)
- HCS12\_DevKit\_S12G128\_CodeWarrior/Boot/hooks.↔  
c, [2124](#)
- HCS12\_Evbplus\_Dragon12p\_CodeWarrior/↔  
Boot/hooks.c, [2129](#)
- NvmErase  
\_template/nvmm.c, [2960](#)
- ARMCM0\_S32K11\_nvmm.c, [2966](#)
- ARMCM0\_STM32F0\_nvmm.c, [2971](#)
- ARMCM0\_STM32G0\_nvmm.c, [2976](#)
- ARMCM0\_XMC1\_nvmm.c, [2981](#)
- ARMCM33\_STM32L5\_nvmm.c, [2986](#)
- ARMCM3\_EFM32\_nvmm.c, [2991](#)
- ARMCM3\_LM3S\_nvmm.c, [2996](#)
- ARMCM3\_STM32F1\_nvmm.c, [3001](#)
- ARMCM3\_STM32F2\_nvmm.c, [3006](#)
- ARMCM4\_S32K14\_nvmm.c, [3011](#)
- ARMCM4\_STM32F3\_nvmm.c, [3016](#)
- ARMCM4\_STM32F4\_nvmm.c, [3021](#)
- ARMCM4\_STM32L4\_nvmm.c, [3026](#)
- ARMCM4\_TM4C\_nvmm.c, [3031](#)
- ARMCM4\_XMC4\_nvmm.c, [3036](#)
- ARMCM7\_STM32F7\_nvmm.c, [3041](#)
- ARMCM7\_STM32H7\_nvmm.c, [3046](#)
- HCS12\_nvmm.c, [3051](#)
- nvmm.h, [3056](#)
- NvmEraseHook  
\_template/Boot/hooks.c, [1622](#)
- \_template/nvmm.c, [2960](#)
- ARMCM0\_S32K11\_nvmm.c, [2966](#)
- ARMCM0\_STM32F0\_nvmm.c, [2972](#)
- ARMCM0\_STM32F0\_Discovery\_STM32F051\_↔  
CubeIDE/Boot/App/hooks.c, [1627](#)
- ARMCM0\_STM32F0\_Discovery\_STM32F051\_↔  
GCC/Boot/hooks.c, [1632](#)
- ARMCM0\_STM32F0\_Discovery\_STM32F051\_I↔  
AR/Boot/hooks.c, [1637](#)
- ARMCM0\_STM32F0\_Discovery\_STM32F051\_↔  
Keil/Boot/hooks.c, [1642](#)
- ARMCM0\_STM32F0\_Nucleo\_F091RC\_CubeID↔  
E/Boot/App/hooks.c, [1647](#)
- ARMCM0\_STM32F0\_Nucleo\_F091RC\_GCC/↔  
Boot/hooks.c, [1652](#)
- ARMCM0\_STM32F0\_Nucleo\_F091RC\_IAR/↔  
Boot/hooks.c, [1657](#)
- ARMCM0\_STM32F0\_Nucleo\_F091RC\_Keil/↔  
Boot/hooks.c, [1662](#)
- ARMCM0\_STM32G0\_nvmm.c, [2977](#)
- ARMCM0\_STM32G0\_Nucleo\_G071RB\_CubeID↔  
E/Boot/App/hooks.c, [1667](#)
- ARMCM0\_STM32G0\_Nucleo\_G071RB\_GCC/↔  
Boot/hooks.c, [1672](#)
- ARMCM0\_STM32G0\_Nucleo\_G071RB\_IAR/↔  
Boot/hooks.c, [1677](#)
- ARMCM0\_STM32G0\_Nucleo\_G071RB\_Keil/↔

- Boot/hooks.c, [1682](#)
- ARMCM0\_XMC1/nvm.c, [2982](#)
- ARMCM0\_XMC1\_XMC1400\_Boot\_Kit\_GCC/↔  
Boot/hooks.c, [1687](#)
- ARMCM0\_XMC1\_XMC1400\_Boot\_Kit\_IAR/↔  
Boot/hooks.c, [1692](#)
- ARMCM33\_STM32L5/nvm.c, [2987](#)
- ARMCM33\_STM32L5\_Nucleo\_L552ZE\_CubeID↔  
E/Boot/App/hooks.c, [1697](#)
- ARMCM33\_STM32L5\_Nucleo\_L552ZE\_GCC/↔  
Boot/hooks.c, [1703](#)
- ARMCM33\_STM32L5\_Nucleo\_L552ZE\_IAR/↔  
Boot/hooks.c, [1709](#)
- ARMCM33\_STM32L5\_Nucleo\_L552ZE\_Keil/↔  
Boot/hooks.c, [1715](#)
- ARMCM3\_EFM32/nvm.c, [2992](#)
- ARMCM3\_EFM32\_Olimex\_EM32G880F128ST↔  
K\_GCC/Boot/hooks.c, [1721](#)
- ARMCM3\_EFM32\_Olimex\_EM32G880F128ST↔  
K\_IAR/Boot/hooks.c, [1726](#)
- ARMCM3\_LM3S/nvm.c, [2997](#)
- ARMCM3\_LM3S\_EK\_LM3S6965\_GCC/Boot/hooks.↔  
c, [1734](#)
- ARMCM3\_LM3S\_EK\_LM3S6965\_IAR/Boot/hooks.↔  
c, [1741](#)
- ARMCM3\_LM3S\_EK\_LM3S8962\_GCC/Boot/hooks.↔  
c, [1747](#)
- ARMCM3\_LM3S\_EK\_LM3S8962\_IAR/Boot/hooks.↔  
c, [1752](#)
- ARMCM3\_STM32F1/nvm.c, [3002](#)
- ARMCM3\_STM32F1\_Nucleo\_F103RB\_CubeID↔  
E/Boot/App/hooks.c, [1757](#)
- ARMCM3\_STM32F1\_Nucleo\_F103RB\_GCC/↔  
Boot/hooks.c, [1762](#)
- ARMCM3\_STM32F1\_Nucleo\_F103RB\_IAR/↔  
Boot/hooks.c, [1767](#)
- ARMCM3\_STM32F1\_Nucleo\_F103RB\_Keil/↔  
Boot/hooks.c, [1772](#)
- ARMCM3\_STM32F1\_Olimex\_STM32H103↔  
CubeIDE/Boot/App/hooks.c, [1777](#)
- ARMCM3\_STM32F1\_Olimex\_STM32H103\_GC↔  
C/Boot/hooks.c, [1783](#)
- ARMCM3\_STM32F1\_Olimex\_STM32H103\_IA↔  
R/Boot/hooks.c, [1789](#)
- ARMCM3\_STM32F1\_Olimex\_STM32H103↔  
Keil/Boot/hooks.c, [1795](#)
- ARMCM3\_STM32F1\_Olimex\_STM32P103↔  
CubeIDE/Boot/App/hooks.c, [1803](#)
- ARMCM3\_STM32F1\_Olimex\_STM32P103\_GC↔  
C/Boot/hooks.c, [1811](#)
- ARMCM3\_STM32F1\_Olimex\_STM32P103\_IA↔  
R/Boot/hooks.c, [1819](#)
- ARMCM3\_STM32F1\_Olimex\_STM32P103\_Keil/↔  
Boot/hooks.c, [1826](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_Cube↔  
DE/Boot/App/hooks.c, [1834](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_GCC/↔  
Boot/hooks.c, [1843](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_IAR/↔  
Boot/hooks.c, [1852](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_Keil/↔  
Boot/hooks.c, [1861](#)
- ARMCM3\_STM32F2/nvm.c, [3007](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207↔  
CubeIDE/Boot/App/hooks.c, [1869](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_GC↔  
C/Boot/hooks.c, [1877](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_IA↔  
R/Boot/hooks.c, [1885](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_Keil/↔  
Boot/hooks.c, [1892](#)
- ARMCM4\_S32K14/nvm.c, [3012](#)
- ARMCM4\_S32K14\_S32K144EVB\_GCC/Boot/hooks.↔  
c, [1898](#)
- ARMCM4\_S32K14\_S32K144EVB\_IAR/Boot/hooks.↔  
c, [1903](#)
- ARMCM4\_STM32F3/nvm.c, [3017](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_Cube↔  
IDE/Boot/App/hooks.c, [1908](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_GCC/↔  
Boot/hooks.c, [1914](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_IAR/↔  
Boot/hooks.c, [1920](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_Keil/↔  
Boot/hooks.c, [1926](#)
- ARMCM4\_STM32F3\_Nucleo\_F303K8\_CubeID↔  
E/Boot/App/hooks.c, [1932](#)
- ARMCM4\_STM32F3\_Nucleo\_F303K8\_GCC/↔  
Boot/hooks.c, [1937](#)
- ARMCM4\_STM32F3\_Nucleo\_F303K8\_IAR/↔  
Boot/hooks.c, [1942](#)
- ARMCM4\_STM32F3\_Nucleo\_F303K8\_Keil/↔  
Boot/hooks.c, [1947](#)
- ARMCM4\_STM32F4/nvm.c, [3022](#)
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_CubeID↔  
E/Boot/App/hooks.c, [1952](#)
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_GCC/↔  
Boot/hooks.c, [1958](#)
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_IAR/↔  
Boot/hooks.c, [1964](#)
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_Keil/↔  
Boot/hooks.c, [1970](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405↔  
CubeIDE/Boot/App/hooks.c, [1978](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_GC↔  
C/Boot/hooks.c, [1987](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_IA↔  
R/Boot/hooks.c, [1996](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_Keil/↔  
Boot/hooks.c, [2005](#)
- ARMCM4\_STM32L4/nvm.c, [3027](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_CubeID↔  
E/Boot/App/hooks.c, [2011](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_GCC/↔  
Boot/hooks.c, [2016](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_IAR/↔

- Boot/hooks.c, [2021](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_Keil/↔
  - Boot/hooks.c, [2026](#)
- ARMCM4\_TM4C/nvm.c, [3032](#)
- ARMCM4\_TM4C\_DK\_TM4C123G\_IAR/Boot/hooks.↔
  - c, [2034](#)
- ARMCM4\_XMC4/nvm.c, [3037](#)
- ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_GCC/↔
  - Boot/hooks.c, [2042](#)
- ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_IAR/↔
  - Boot/hooks.c, [2050](#)
- ARMCM7\_STM32F7/nvm.c, [3042](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_CubeID↔
  - E/Boot/App/hooks.c, [2056](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_GCC/↔
  - Boot/hooks.c, [2062](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_IAR/↔
  - Boot/hooks.c, [2068](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_Keil/↔
  - Boot/hooks.c, [2074](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_CubeID↔
  - E/Boot/App/hooks.c, [2080](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_GCC/↔
  - Boot/hooks.c, [2085](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_IAR/↔
  - Boot/hooks.c, [2090](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_Keil/↔
  - Boot/hooks.c, [2095](#)
- ARMCM7\_STM32H7/nvm.c, [3047](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_CubeID↔
  - E/Boot/App/hooks.c, [2100](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_GCC/↔
  - Boot/hooks.c, [2106](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_IAR/↔
  - Boot/hooks.c, [2112](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_Keil/↔
  - Boot/hooks.c, [2118](#)
- HCS12/nvm.c, [3052](#)
- HCS12\_DevKit\_S12G128\_CodeWarrior/Boot/hooks.↔
  - c, [2124](#)
- HCS12\_Evbplus\_Dragon12p\_CodeWarrior/↔
  - Boot/hooks.c, [2130](#)
- NvmGetUserProgBaseAddress
  - \_template/nvm.c, [2961](#)
  - ARMCM0\_S32K11/nvm.c, [2967](#)
  - ARMCM0\_STM32F0/nvm.c, [2972](#)
  - ARMCM0\_STM32G0/nvm.c, [2977](#)
  - ARMCM0\_XMC1/nvm.c, [2982](#)
  - ARMCM33\_STM32L5/nvm.c, [2987](#)
  - ARMCM3\_EFM32/nvm.c, [2992](#)
  - ARMCM3\_LM3S/nvm.c, [2997](#)
  - ARMCM3\_STM32F1/nvm.c, [3002](#)
  - ARMCM3\_STM32F2/nvm.c, [3007](#)
  - ARMCM4\_S32K14/nvm.c, [3012](#)
  - ARMCM4\_STM32F3/nvm.c, [3017](#)
  - ARMCM4\_STM32F4/nvm.c, [3022](#)
  - ARMCM4\_STM32L4/nvm.c, [3027](#)
  - ARMCM4\_TM4C/nvm.c, [3032](#)
  - ARMCM4\_XMC4/nvm.c, [3037](#)
  - ARMCM7\_STM32F7/nvm.c, [3042](#)
  - ARMCM7\_STM32H7/nvm.c, [3047](#)
  - HCS12/nvm.c, [3052](#)
  - nvm.h, [3056](#)
- NvmInit
  - \_template/nvm.c, [2961](#)
  - ARMCM0\_S32K11/nvm.c, [2967](#)
  - ARMCM0\_STM32F0/nvm.c, [2972](#)
  - ARMCM0\_STM32G0/nvm.c, [2977](#)
  - ARMCM0\_XMC1/nvm.c, [2982](#)
  - ARMCM33\_STM32L5/nvm.c, [2987](#)
  - ARMCM3\_EFM32/nvm.c, [2992](#)
  - ARMCM3\_LM3S/nvm.c, [2997](#)
  - ARMCM3\_STM32F1/nvm.c, [3002](#)
  - ARMCM3\_STM32F2/nvm.c, [3007](#)
  - ARMCM4\_S32K14/nvm.c, [3012](#)
  - ARMCM4\_STM32F3/nvm.c, [3017](#)
  - ARMCM4\_STM32F4/nvm.c, [3022](#)
  - ARMCM4\_STM32L4/nvm.c, [3027](#)
  - ARMCM4\_TM4C/nvm.c, [3032](#)
  - ARMCM4\_XMC4/nvm.c, [3037](#)
  - ARMCM7\_STM32F7/nvm.c, [3042](#)
  - ARMCM7\_STM32H7/nvm.c, [3047](#)
  - HCS12/nvm.c, [3052](#)
  - nvm.h, [3057](#)
- NvmInitHook
  - \_template/Boot/hooks.c, [1623](#)
  - \_template/nvm.c, [2961](#)
  - ARMCM0\_S32K11/nvm.c, [2967](#)
  - ARMCM0\_STM32F0/nvm.c, [2973](#)
  - ARMCM0\_STM32F0\_Discovery\_STM32F051\_↔
    - CubeIDE/Boot/App/hooks.c, [1628](#)
  - ARMCM0\_STM32F0\_Discovery\_STM32F051\_↔
    - GCC/Boot/hooks.c, [1633](#)
  - ARMCM0\_STM32F0\_Discovery\_STM32F051\_I↔
    - AR/Boot/hooks.c, [1638](#)
  - ARMCM0\_STM32F0\_Discovery\_STM32F051\_↔
    - Keil/Boot/hooks.c, [1643](#)
  - ARMCM0\_STM32F0\_Nucleo\_F091RC\_CubeID↔
    - E/Boot/App/hooks.c, [1648](#)
  - ARMCM0\_STM32F0\_Nucleo\_F091RC\_GCC/↔
    - Boot/hooks.c, [1653](#)
  - ARMCM0\_STM32F0\_Nucleo\_F091RC\_IAR/↔
    - Boot/hooks.c, [1658](#)
  - ARMCM0\_STM32F0\_Nucleo\_F091RC\_Keil/↔
    - Boot/hooks.c, [1663](#)
  - ARMCM0\_STM32G0/nvm.c, [2978](#)
  - ARMCM0\_STM32G0\_Nucleo\_G071RB\_CubeID↔
    - E/Boot/App/hooks.c, [1668](#)
  - ARMCM0\_STM32G0\_Nucleo\_G071RB\_GCC/↔
    - Boot/hooks.c, [1673](#)
  - ARMCM0\_STM32G0\_Nucleo\_G071RB\_IAR/↔
    - Boot/hooks.c, [1678](#)
  - ARMCM0\_STM32G0\_Nucleo\_G071RB\_Keil/↔
    - Boot/hooks.c, [1683](#)
  - ARMCM0\_XMC1/nvm.c, [2983](#)



- ARMCM0\_XMC1\_XMC1400\_Boot\_Kit\_GCC/↔  
Boot/hooks.c, [1688](#)
- ARMCM0\_XMC1\_XMC1400\_Boot\_Kit\_IAR/↔  
Boot/hooks.c, [1693](#)
- ARMCM33\_STM32L5/nvm.c, [2988](#)
- ARMCM33\_STM32L5\_Nucleo\_L552ZE\_CubeID↔  
E/Boot/App/hooks.c, [1698](#)
- ARMCM33\_STM32L5\_Nucleo\_L552ZE\_GCC/↔  
Boot/hooks.c, [1704](#)
- ARMCM33\_STM32L5\_Nucleo\_L552ZE\_IAR/↔  
Boot/hooks.c, [1710](#)
- ARMCM33\_STM32L5\_Nucleo\_L552ZE\_Keil/↔  
Boot/hooks.c, [1716](#)
- ARMCM3\_EFM32/nvm.c, [2993](#)
- ARMCM3\_EFM32\_Olimex\_EM32G880F128ST↔  
K\_GCC/Boot/hooks.c, [1722](#)
- ARMCM3\_EFM32\_Olimex\_EM32G880F128ST↔  
K\_IAR/Boot/hooks.c, [1727](#)
- ARMCM3\_LM3S/nvm.c, [2998](#)
- ARMCM3\_LM3S\_EK\_LM3S6965\_GCC/Boot/hooks.↔  
c, [1734](#)
- ARMCM3\_LM3S\_EK\_LM3S6965\_IAR/Boot/hooks.↔  
c, [1742](#)
- ARMCM3\_LM3S\_EK\_LM3S8962\_GCC/Boot/hooks.↔  
c, [1747](#)
- ARMCM3\_LM3S\_EK\_LM3S8962\_IAR/Boot/hooks.↔  
c, [1753](#)
- ARMCM3\_STM32F1/nvm.c, [3003](#)
- ARMCM3\_STM32F1\_Nucleo\_F103RB\_CubeID↔  
E/Boot/App/hooks.c, [1758](#)
- ARMCM3\_STM32F1\_Nucleo\_F103RB\_GCC/↔  
Boot/hooks.c, [1763](#)
- ARMCM3\_STM32F1\_Nucleo\_F103RB\_IAR/↔  
Boot/hooks.c, [1768](#)
- ARMCM3\_STM32F1\_Nucleo\_F103RB\_Keil/↔  
Boot/hooks.c, [1773](#)
- ARMCM3\_STM32F1\_Olimex\_STM32H103↔  
CubeIDE/Boot/App/hooks.c, [1778](#)
- ARMCM3\_STM32F1\_Olimex\_STM32H103\_GC↔  
C/Boot/hooks.c, [1784](#)
- ARMCM3\_STM32F1\_Olimex\_STM32H103\_IA↔  
R/Boot/hooks.c, [1790](#)
- ARMCM3\_STM32F1\_Olimex\_STM32H103↔  
Keil/Boot/hooks.c, [1796](#)
- ARMCM3\_STM32F1\_Olimex\_STM32P103↔  
CubeIDE/Boot/App/hooks.c, [1804](#)
- ARMCM3\_STM32F1\_Olimex\_STM32P103\_GC↔  
C/Boot/hooks.c, [1811](#)
- ARMCM3\_STM32F1\_Olimex\_STM32P103\_IA↔  
R/Boot/hooks.c, [1819](#)
- ARMCM3\_STM32F1\_Olimex\_STM32P103\_Keil/↔  
Boot/hooks.c, [1827](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_Cube↔  
DE/Boot/App/hooks.c, [1834](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_GCC/↔  
Boot/hooks.c, [1843](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_IAR/↔  
Boot/hooks.c, [1852](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_Keil/↔  
Boot/hooks.c, [1861](#)
- ARMCM3\_STM32F2/nvm.c, [3008](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207↔  
CubeIDE/Boot/App/hooks.c, [1870](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_GC↔  
C/Boot/hooks.c, [1877](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_IA↔  
R/Boot/hooks.c, [1885](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_Keil/↔  
Boot/hooks.c, [1893](#)
- ARMCM4\_S32K14/nvm.c, [3013](#)
- ARMCM4\_S32K14\_S32K144EVB\_GCC/Boot/hooks.↔  
c, [1898](#)
- ARMCM4\_S32K14\_S32K144EVB\_IAR/Boot/hooks.↔  
c, [1904](#)
- ARMCM4\_STM32F3/nvm.c, [3018](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_Cube↔  
IDE/Boot/App/hooks.c, [1909](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_GCC/↔  
Boot/hooks.c, [1915](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_IAR/↔  
Boot/hooks.c, [1921](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_Keil/↔  
Boot/hooks.c, [1927](#)
- ARMCM4\_STM32F3\_Nucleo\_F303K8\_CubeID↔  
E/Boot/App/hooks.c, [1933](#)
- ARMCM4\_STM32F3\_Nucleo\_F303K8\_GCC/↔  
Boot/hooks.c, [1938](#)
- ARMCM4\_STM32F3\_Nucleo\_F303K8\_IAR/↔  
Boot/hooks.c, [1943](#)
- ARMCM4\_STM32F3\_Nucleo\_F303K8\_Keil/↔  
Boot/hooks.c, [1948](#)
- ARMCM4\_STM32F4/nvm.c, [3023](#)
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_CubeID↔  
E/Boot/App/hooks.c, [1953](#)
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_GCC/↔  
Boot/hooks.c, [1959](#)
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_IAR/↔  
Boot/hooks.c, [1965](#)
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_Keil/↔  
Boot/hooks.c, [1971](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405↔  
CubeIDE/Boot/App/hooks.c, [1979](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_GC↔  
C/Boot/hooks.c, [1987](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_IA↔  
R/Boot/hooks.c, [1996](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_Keil/↔  
Boot/hooks.c, [2005](#)
- ARMCM4\_STM32L4/nvm.c, [3028](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_CubeID↔  
E/Boot/App/hooks.c, [2012](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_GCC/↔  
Boot/hooks.c, [2017](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_IAR/↔  
Boot/hooks.c, [2022](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_Keil/↔

- Boot/hooks.c, [2027](#)
- ARMCM4\_TM4C/nvm.c, [3033](#)
- ARMCM4\_TM4C\_DK\_TM4C123G\_IAR/Boot/hooks.c, [2034](#)
- ARMCM4\_XMC4/nvm.c, [3038](#)
- ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_GCC/↔  
Boot/hooks.c, [2043](#)
- ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_IAR/↔  
Boot/hooks.c, [2051](#)
- ARMCM7\_STM32F7/nvm.c, [3043](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_CubeID↔  
E/Boot/App/hooks.c, [2056](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_GCC/↔  
Boot/hooks.c, [2063](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_IAR/↔  
Boot/hooks.c, [2069](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_Keil/↔  
Boot/hooks.c, [2075](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_CubeID↔  
E/Boot/App/hooks.c, [2081](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_GCC/↔  
Boot/hooks.c, [2086](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_IAR/↔  
Boot/hooks.c, [2091](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_Keil/↔  
Boot/hooks.c, [2096](#)
- ARMCM7\_STM32H7/nvm.c, [3048](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_CubeID↔  
E/Boot/App/hooks.c, [2101](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_GCC/↔  
Boot/hooks.c, [2107](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_IAR/↔  
Boot/hooks.c, [2113](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_Keil/↔  
Boot/hooks.c, [2119](#)
- HCS12/nvm.c, [3053](#)
- HCS12\_DevKit\_S12G128\_CodeWarrior/Boot/hooks.c, [2125](#)
- HCS12\_Evbplus\_Dragon12p\_CodeWarrior/↔  
Boot/hooks.c, [2130](#)
- NvmReinit
  - \_template/nvm.c, [2962](#)
  - ARMCM0\_S32K11/nvm.c, [2967](#)
  - ARMCM0\_STM32F0/nvm.c, [2973](#)
  - ARMCM0\_STM32G0/nvm.c, [2978](#)
  - ARMCM0\_XMC1/nvm.c, [2983](#)
  - ARMCM33\_STM32L5/nvm.c, [2988](#)
  - ARMCM3\_EFM32/nvm.c, [2993](#)
  - ARMCM3\_LM3S/nvm.c, [2998](#)
  - ARMCM3\_STM32F1/nvm.c, [3003](#)
  - ARMCM3\_STM32F2/nvm.c, [3008](#)
  - ARMCM4\_S32K14/nvm.c, [3013](#)
  - ARMCM4\_STM32F3/nvm.c, [3018](#)
  - ARMCM4\_STM32F4/nvm.c, [3023](#)
  - ARMCM4\_STM32L4/nvm.c, [3028](#)
  - ARMCM4\_TM4C/nvm.c, [3033](#)
  - ARMCM4\_XMC4/nvm.c, [3038](#)
  - ARMCM7\_STM32F7/nvm.c, [3043](#)
  - ARMCM7\_STM32H7/nvm.c, [3048](#)
  - HCS12/nvm.c, [3053](#)
  - nvm.h, [3057](#)
  - NvmReinitHook
    - \_template/Boot/hooks.c, [1623](#)
    - \_template/nvm.c, [2962](#)
    - ARMCM0\_S32K11/nvm.c, [2968](#)
    - ARMCM0\_STM32F0/nvm.c, [2973](#)
    - ARMCM0\_STM32F0\_Discovery\_STM32F051\_↔  
CubeIDE/Boot/App/hooks.c, [1628](#)
    - ARMCM0\_STM32F0\_Discovery\_STM32F051\_↔  
GCC/Boot/hooks.c, [1633](#)
    - ARMCM0\_STM32F0\_Discovery\_STM32F051\_I↔  
AR/Boot/hooks.c, [1638](#)
    - ARMCM0\_STM32F0\_Discovery\_STM32F051\_↔  
Keil/Boot/hooks.c, [1643](#)
    - ARMCM0\_STM32F0\_Nucleo\_F091RC\_CubeID↔  
E/Boot/App/hooks.c, [1648](#)
    - ARMCM0\_STM32F0\_Nucleo\_F091RC\_GCC/↔  
Boot/hooks.c, [1653](#)
    - ARMCM0\_STM32F0\_Nucleo\_F091RC\_IAR/↔  
Boot/hooks.c, [1658](#)
    - ARMCM0\_STM32F0\_Nucleo\_F091RC\_Keil/↔  
Boot/hooks.c, [1663](#)
    - ARMCM0\_STM32G0/nvm.c, [2978](#)
    - ARMCM0\_STM32G0\_Nucleo\_G071RB\_CubeID↔  
E/Boot/App/hooks.c, [1668](#)
    - ARMCM0\_STM32G0\_Nucleo\_G071RB\_GCC/↔  
Boot/hooks.c, [1673](#)
    - ARMCM0\_STM32G0\_Nucleo\_G071RB\_IAR/↔  
Boot/hooks.c, [1678](#)
    - ARMCM0\_STM32G0\_Nucleo\_G071RB\_Keil/↔  
Boot/hooks.c, [1683](#)
    - ARMCM0\_XMC1/nvm.c, [2983](#)
    - ARMCM0\_XMC1\_XMC1400\_Boot\_Kit\_GCC/↔  
Boot/hooks.c, [1688](#)
    - ARMCM0\_XMC1\_XMC1400\_Boot\_Kit\_IAR/↔  
Boot/hooks.c, [1693](#)
    - ARMCM33\_STM32L5/nvm.c, [2988](#)
    - ARMCM33\_STM32L5\_Nucleo\_L552ZE\_CubeID↔  
E/Boot/App/hooks.c, [1698](#)
    - ARMCM33\_STM32L5\_Nucleo\_L552ZE\_GCC/↔  
Boot/hooks.c, [1704](#)
    - ARMCM33\_STM32L5\_Nucleo\_L552ZE\_IAR/↔  
Boot/hooks.c, [1710](#)
    - ARMCM33\_STM32L5\_Nucleo\_L552ZE\_Keil/↔  
Boot/hooks.c, [1716](#)
    - ARMCM3\_EFM32/nvm.c, [2993](#)
    - ARMCM3\_EFM32\_Olimex\_EM32G880F128ST↔  
K\_GCC/Boot/hooks.c, [1722](#)
    - ARMCM3\_EFM32\_Olimex\_EM32G880F128ST↔  
K\_IAR/Boot/hooks.c, [1727](#)
    - ARMCM3\_LM3S/nvm.c, [2998](#)
    - ARMCM3\_LM3S\_EK\_LM3S6965\_GCC/Boot/hooks.c, [1734](#)
    - ARMCM3\_LM3S\_EK\_LM3S6965\_IAR/Boot/hooks.c, [1742](#)
    - ARMCM3\_LM3S\_EK\_LM3S8962\_GCC/Boot/hooks.c, ↔

- c, [1748](#)
- ARMCM3\_LM3S\_EK\_LM3S8962\_IAR/Boot/hooks.↔  
c, [1753](#)
- ARMCM3\_STM32F1/nvm.c, [3003](#)
- ARMCM3\_STM32F1\_Nucleo\_F103RB\_CubeID↔  
E/Boot/App/hooks.c, [1758](#)
- ARMCM3\_STM32F1\_Nucleo\_F103RB\_GCC/↔  
Boot/hooks.c, [1763](#)
- ARMCM3\_STM32F1\_Nucleo\_F103RB\_IAR/↔  
Boot/hooks.c, [1768](#)
- ARMCM3\_STM32F1\_Nucleo\_F103RB\_Keil/↔  
Boot/hooks.c, [1773](#)
- ARMCM3\_STM32F1\_Olimex\_STM32H103\_↔  
CubeIDE/Boot/App/hooks.c, [1778](#)
- ARMCM3\_STM32F1\_Olimex\_STM32H103\_GC↔  
C/Boot/hooks.c, [1784](#)
- ARMCM3\_STM32F1\_Olimex\_STM32H103\_IA↔  
R/Boot/hooks.c, [1790](#)
- ARMCM3\_STM32F1\_Olimex\_STM32H103\_↔  
Keil/Boot/hooks.c, [1796](#)
- ARMCM3\_STM32F1\_Olimex\_STM32P103\_↔  
CubeIDE/Boot/App/hooks.c, [1804](#)
- ARMCM3\_STM32F1\_Olimex\_STM32P103\_GC↔  
C/Boot/hooks.c, [1812](#)
- ARMCM3\_STM32F1\_Olimex\_STM32P103\_IA↔  
R/Boot/hooks.c, [1819](#)
- ARMCM3\_STM32F1\_Olimex\_STM32P103\_Keil/↔  
Boot/hooks.c, [1827](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_CubeID↔  
DE/Boot/App/hooks.c, [1835](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_GCC/↔  
Boot/hooks.c, [1843](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_IAR/↔  
Boot/hooks.c, [1852](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_Keil/↔  
Boot/hooks.c, [1861](#)
- ARMCM3\_STM32F2/nvm.c, [3008](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_↔  
CubeIDE/Boot/App/hooks.c, [1870](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_GC↔  
C/Boot/hooks.c, [1878](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_IA↔  
R/Boot/hooks.c, [1885](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_Keil/↔  
Boot/hooks.c, [1893](#)
- ARMCM4\_S32K14/nvm.c, [3013](#)
- ARMCM4\_S32K14\_S32K144EVB\_GCC/Boot/hooks.↔  
c, [1899](#)
- ARMCM4\_S32K14\_S32K144EVB\_IAR/Boot/hooks.↔  
c, [1904](#)
- ARMCM4\_STM32F3/nvm.c, [3018](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_Cube↔  
IDE/Boot/App/hooks.c, [1909](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_GCC/↔  
Boot/hooks.c, [1915](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_IAR/↔  
Boot/hooks.c, [1921](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_Keil/↔  
Boot/hooks.c, [1927](#)
- ARMCM4\_STM32F3\_Nucleo\_F303K8\_CubeID↔  
E/Boot/App/hooks.c, [1933](#)
- ARMCM4\_STM32F3\_Nucleo\_F303K8\_GCC/↔  
Boot/hooks.c, [1938](#)
- ARMCM4\_STM32F3\_Nucleo\_F303K8\_IAR/↔  
Boot/hooks.c, [1943](#)
- ARMCM4\_STM32F3\_Nucleo\_F303K8\_Keil/↔  
Boot/hooks.c, [1948](#)
- ARMCM4\_STM32F4/nvm.c, [3023](#)
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_CubeID↔  
E/Boot/App/hooks.c, [1953](#)
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_GCC/↔  
Boot/hooks.c, [1959](#)
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_IAR/↔  
Boot/hooks.c, [1965](#)
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_Keil/↔  
Boot/hooks.c, [1971](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_↔  
CubeIDE/Boot/App/hooks.c, [1979](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_GC↔  
C/Boot/hooks.c, [1987](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_IA↔  
R/Boot/hooks.c, [1996](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_Keil/↔  
Boot/hooks.c, [2005](#)
- ARMCM4\_STM32L4/nvm.c, [3028](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_CubeID↔  
E/Boot/App/hooks.c, [2012](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_GCC/↔  
Boot/hooks.c, [2017](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_IAR/↔  
Boot/hooks.c, [2022](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_Keil/↔  
Boot/hooks.c, [2027](#)
- ARMCM4\_TM4C/nvm.c, [3033](#)
- ARMCM4\_TM4C\_DK\_TM4C123G\_IAR/Boot/hooks.↔  
c, [2034](#)
- ARMCM4\_XMC4/nvm.c, [3038](#)
- ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_GCC/↔  
Boot/hooks.c, [2043](#)
- ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_IAR/↔  
Boot/hooks.c, [2051](#)
- ARMCM7\_STM32F7/nvm.c, [3043](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_CubeID↔  
E/Boot/App/hooks.c, [2057](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_GCC/↔  
Boot/hooks.c, [2063](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_IAR/↔  
Boot/hooks.c, [2069](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_Keil/↔  
Boot/hooks.c, [2075](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_CubeID↔  
E/Boot/App/hooks.c, [2081](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_GCC/↔  
Boot/hooks.c, [2086](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_IAR/↔  
Boot/hooks.c, [2091](#)

- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_Keil/↵  
  Boot/hooks.c, [2096](#)
- ARMCM7\_STM32H7/nvm.c, [3048](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_CubeID↵  
  E/Boot/App/hooks.c, [2101](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_GCC/↵  
  Boot/hooks.c, [2107](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_IAR/↵  
  Boot/hooks.c, [2113](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_Keil/↵  
  Boot/hooks.c, [2119](#)
- HCS12/nvm.c, [3053](#)
- HCS12\_DevKit\_S12G128\_CodeWarrior/Boot/hooks.↵  
  c, [2125](#)
- HCS12\_Evbplus\_Dragon12p\_CodeWarrior/↵  
  Boot/hooks.c, [2130](#)
- NvmVerifyChecksum  
  \_template/nvm.c, [2962](#)
- ARMCM0\_S32K11/nvm.c, [2968](#)
- ARMCM0\_STM32F0/nvm.c, [2973](#)
- ARMCM0\_STM32G0/nvm.c, [2978](#)
- ARMCM0\_XMC1/nvm.c, [2983](#)
- ARMCM33\_STM32L5/nvm.c, [2988](#)
- ARMCM3\_EFM32/nvm.c, [2993](#)
- ARMCM3\_LM3S/nvm.c, [2998](#)
- ARMCM3\_STM32F1/nvm.c, [3003](#)
- ARMCM3\_STM32F2/nvm.c, [3008](#)
- ARMCM4\_S32K14/nvm.c, [3013](#)
- ARMCM4\_STM32F3/nvm.c, [3018](#)
- ARMCM4\_STM32F4/nvm.c, [3023](#)
- ARMCM4\_STM32L4/nvm.c, [3028](#)
- ARMCM4\_TM4C/nvm.c, [3033](#)
- ARMCM4\_XMC4/nvm.c, [3038](#)
- ARMCM7\_STM32F7/nvm.c, [3043](#)
- ARMCM7\_STM32H7/nvm.c, [3048](#)
- HCS12/nvm.c, [3053](#)
- nvm.h, [3057](#)
- NvmWrite  
  \_template/nvm.c, [2963](#)
- ARMCM0\_S32K11/nvm.c, [2968](#)
- ARMCM0\_STM32F0/nvm.c, [2974](#)
- ARMCM0\_STM32G0/nvm.c, [2979](#)
- ARMCM0\_XMC1/nvm.c, [2984](#)
- ARMCM33\_STM32L5/nvm.c, [2989](#)
- ARMCM3\_EFM32/nvm.c, [2994](#)
- ARMCM3\_LM3S/nvm.c, [2999](#)
- ARMCM3\_STM32F1/nvm.c, [3004](#)
- ARMCM3\_STM32F2/nvm.c, [3009](#)
- ARMCM4\_S32K14/nvm.c, [3014](#)
- ARMCM4\_STM32F3/nvm.c, [3019](#)
- ARMCM4\_STM32F4/nvm.c, [3024](#)
- ARMCM4\_STM32L4/nvm.c, [3029](#)
- ARMCM4\_TM4C/nvm.c, [3034](#)
- ARMCM4\_XMC4/nvm.c, [3039](#)
- ARMCM7\_STM32F7/nvm.c, [3044](#)
- ARMCM7\_STM32H7/nvm.c, [3049](#)
- HCS12/nvm.c, [3054](#)
- nvm.h, [3057](#)
- NvmWriteHook  
  \_template/Boot/hooks.c, [1623](#)
- \_template/nvm.c, [2963](#)
- ARMCM0\_S32K11/nvm.c, [2969](#)
- ARMCM0\_STM32F0/nvm.c, [2974](#)
- ARMCM0\_STM32F0\_Discovery\_STM32F051\_↵  
  CubeIDE/Boot/App/hooks.c, [1628](#)
- ARMCM0\_STM32F0\_Discovery\_STM32F051\_↵  
  GCC/Boot/hooks.c, [1633](#)
- ARMCM0\_STM32F0\_Discovery\_STM32F051\_I↵  
  AR/Boot/hooks.c, [1638](#)
- ARMCM0\_STM32F0\_Discovery\_STM32F051\_↵  
  Keil/Boot/hooks.c, [1643](#)
- ARMCM0\_STM32F0\_Nucleo\_F091RC\_CubeID↵  
  E/Boot/App/hooks.c, [1648](#)
- ARMCM0\_STM32F0\_Nucleo\_F091RC\_GCC/↵  
  Boot/hooks.c, [1653](#)
- ARMCM0\_STM32F0\_Nucleo\_F091RC\_IAR/↵  
  Boot/hooks.c, [1658](#)
- ARMCM0\_STM32F0\_Nucleo\_F091RC\_Keil/↵  
  Boot/hooks.c, [1663](#)
- ARMCM0\_STM32G0/nvm.c, [2979](#)
- ARMCM0\_STM32G0\_Nucleo\_G071RB\_CubeID↵  
  E/Boot/App/hooks.c, [1668](#)
- ARMCM0\_STM32G0\_Nucleo\_G071RB\_GCC/↵  
  Boot/hooks.c, [1673](#)
- ARMCM0\_STM32G0\_Nucleo\_G071RB\_IAR/↵  
  Boot/hooks.c, [1678](#)
- ARMCM0\_STM32G0\_Nucleo\_G071RB\_Keil/↵  
  Boot/hooks.c, [1683](#)
- ARMCM0\_XMC1/nvm.c, [2984](#)
- ARMCM0\_XMC1\_XMC1400\_Boot\_Kit\_GCC/↵  
  Boot/hooks.c, [1688](#)
- ARMCM0\_XMC1\_XMC1400\_Boot\_Kit\_IAR/↵  
  Boot/hooks.c, [1693](#)
- ARMCM33\_STM32L5/nvm.c, [2989](#)
- ARMCM33\_STM32L5\_Nucleo\_L552ZE\_CubeID↵  
  E/Boot/App/hooks.c, [1698](#)
- ARMCM33\_STM32L5\_Nucleo\_L552ZE\_GCC/↵  
  Boot/hooks.c, [1704](#)
- ARMCM33\_STM32L5\_Nucleo\_L552ZE\_IAR/↵  
  Boot/hooks.c, [1710](#)
- ARMCM33\_STM32L5\_Nucleo\_L552ZE\_Keil/↵  
  Boot/hooks.c, [1716](#)
- ARMCM3\_EFM32/nvm.c, [2994](#)
- ARMCM3\_EFM32\_Olimex\_EM32G880F128ST↵  
  K\_GCC/Boot/hooks.c, [1722](#)
- ARMCM3\_EFM32\_Olimex\_EM32G880F128ST↵  
  K\_IAR/Boot/hooks.c, [1727](#)
- ARMCM3\_LM3S/nvm.c, [2999](#)
- ARMCM3\_LM3S\_EK\_LM3S6965\_GCC/Boot/hooks.↵  
  c, [1735](#)
- ARMCM3\_LM3S\_EK\_LM3S6965\_IAR/Boot/hooks.↵  
  c, [1742](#)
- ARMCM3\_LM3S\_EK\_LM3S8962\_GCC/Boot/hooks.↵  
  c, [1748](#)
- ARMCM3\_LM3S\_EK\_LM3S8962\_IAR/Boot/hooks.↵  
  c, [1753](#)

- ARMCM3\_STM32F1/nvm.c, [3004](#)
- ARMCM3\_STM32F1\_Nucleo\_F103RB\_CubeID↔  
E/Boot/App/hooks.c, [1758](#)
- ARMCM3\_STM32F1\_Nucleo\_F103RB\_GCC/↔  
Boot/hooks.c, [1763](#)
- ARMCM3\_STM32F1\_Nucleo\_F103RB\_IAR/↔  
Boot/hooks.c, [1768](#)
- ARMCM3\_STM32F1\_Nucleo\_F103RB\_Keil/↔  
Boot/hooks.c, [1773](#)
- ARMCM3\_STM32F1\_Olimex\_STM32H103\_↔  
CubeIDE/Boot/App/hooks.c, [1778](#)
- ARMCM3\_STM32F1\_Olimex\_STM32H103\_GC↔  
C/Boot/hooks.c, [1784](#)
- ARMCM3\_STM32F1\_Olimex\_STM32H103\_IA↔  
R/Boot/hooks.c, [1790](#)
- ARMCM3\_STM32F1\_Olimex\_STM32H103\_↔  
Keil/Boot/hooks.c, [1796](#)
- ARMCM3\_STM32F1\_Olimex\_STM32P103\_↔  
CubeIDE/Boot/App/hooks.c, [1804](#)
- ARMCM3\_STM32F1\_Olimex\_STM32P103\_GC↔  
C/Boot/hooks.c, [1812](#)
- ARMCM3\_STM32F1\_Olimex\_STM32P103\_IA↔  
R/Boot/hooks.c, [1820](#)
- ARMCM3\_STM32F1\_Olimex\_STM32P103\_Keil/↔  
Boot/hooks.c, [1827](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_CubeID↔  
DE/Boot/App/hooks.c, [1835](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_GCC/↔  
Boot/hooks.c, [1844](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_IAR/↔  
Boot/hooks.c, [1853](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_Keil/↔  
Boot/hooks.c, [1862](#)
- ARMCM3\_STM32F2/nvm.c, [3009](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_↔  
CubeIDE/Boot/App/hooks.c, [1870](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_GC↔  
C/Boot/hooks.c, [1878](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_IA↔  
R/Boot/hooks.c, [1886](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_Keil/↔  
Boot/hooks.c, [1893](#)
- ARMCM4\_S32K14/nvm.c, [3014](#)
- ARMCM4\_S32K14\_S32K144EVB\_GCC/Boot/hooks.↔  
c, [1899](#)
- ARMCM4\_S32K14\_S32K144EVB\_IAR/Boot/hooks.↔  
c, [1904](#)
- ARMCM4\_STM32F3/nvm.c, [3019](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_Cube↔  
IDE/Boot/App/hooks.c, [1909](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_GCC/↔  
Boot/hooks.c, [1915](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_IAR/↔  
Boot/hooks.c, [1921](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_Keil/↔  
Boot/hooks.c, [1927](#)
- ARMCM4\_STM32F3\_Nucleo\_F303K8\_CubeID↔  
E/Boot/App/hooks.c, [1933](#)
- ARMCM4\_STM32F3\_Nucleo\_F303K8\_GCC/↔  
Boot/hooks.c, [1938](#)
- ARMCM4\_STM32F3\_Nucleo\_F303K8\_IAR/↔  
Boot/hooks.c, [1943](#)
- ARMCM4\_STM32F3\_Nucleo\_F303K8\_Keil/↔  
Boot/hooks.c, [1948](#)
- ARMCM4\_STM32F4/nvm.c, [3024](#)
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_CubeID↔  
E/Boot/App/hooks.c, [1953](#)
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_GCC/↔  
Boot/hooks.c, [1959](#)
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_IAR/↔  
Boot/hooks.c, [1965](#)
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_Keil/↔  
Boot/hooks.c, [1971](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_↔  
CubeIDE/Boot/App/hooks.c, [1979](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_GC↔  
C/Boot/hooks.c, [1988](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_IA↔  
R/Boot/hooks.c, [1997](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_Keil/↔  
Boot/hooks.c, [2006](#)
- ARMCM4\_STM32L4/nvm.c, [3029](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_CubeID↔  
E/Boot/App/hooks.c, [2012](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_GCC/↔  
Boot/hooks.c, [2017](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_IAR/↔  
Boot/hooks.c, [2022](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_Keil/↔  
Boot/hooks.c, [2027](#)
- ARMCM4\_TM4C/nvm.c, [3034](#)
- ARMCM4\_TM4C\_DK\_TM4C123G\_IAR/Boot/hooks.↔  
c, [2035](#)
- ARMCM4\_XMC4/nvm.c, [3039](#)
- ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_GCC/↔  
Boot/hooks.c, [2043](#)
- ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_IAR/↔  
Boot/hooks.c, [2051](#)
- ARMCM7\_STM32F7/nvm.c, [3044](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_CubeID↔  
E/Boot/App/hooks.c, [2057](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_GCC/↔  
Boot/hooks.c, [2063](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_IAR/↔  
Boot/hooks.c, [2069](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_Keil/↔  
Boot/hooks.c, [2075](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_CubeID↔  
E/Boot/App/hooks.c, [2081](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_GCC/↔  
Boot/hooks.c, [2086](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_IAR/↔  
Boot/hooks.c, [2091](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_Keil/↔  
Boot/hooks.c, [2096](#)
- ARMCM7\_STM32H7/nvm.c, [3049](#)



- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_CubeID↔  
E/Boot/App/hooks.c, [2101](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_GCC/↔  
Boot/hooks.c, [2107](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_IAR/↔  
Boot/hooks.c, [2113](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_Keil/↔  
Boot/hooks.c, [2119](#)
- HCS12/nvm.c, [3054](#)
- HCS12\_DevKit\_S12G128\_CodeWarrior/Boot/hooks.↔  
c, [2125](#)
- HCS12\_Evbplus\_Dragon12p\_CodeWarrior/↔  
Boot/hooks.c, [2131](#)
- oc7d
  - tTimerRegs, [381](#)
- oc7m
  - tTimerRegs, [381](#)
- phaseSeg1
  - tCanBusTiming, [355](#)
- phaseSeg2
  - tCanBusTiming, [356](#)
- plausibility.h, [3058](#)
- PostInit
  - ARMCM0\_XMC1\_XMC1400\_Boot\_Kit\_GCC/↔  
Boot/main.c, [2652](#)
  - ARMCM0\_XMC1\_XMC1400\_Boot\_Kit\_IAR/↔  
Boot/main.c, [2655](#)
  - ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_GCC/↔  
Boot/main.c, [2848](#)
  - ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_IAR/↔  
Boot/main.c, [2851](#)
- prescaler
  - tFlashPrescalerSysclockMapping, [371](#)
- propSeg
  - tCanBusTiming, [356](#)
- protection
  - tXcpInfo, [385](#)
- ptr
  - tlsrFunc, [377](#), [378](#)
- RS232 UART driver of a port, [328](#)
- readptr
  - tFifoCtrl, [368](#)
- reset\_handler
  - ARMCM3\_EFM32\_Olimex\_EM32G880F128ST↔  
K\_GCC/Boot/cstart.c, [1248](#)
  - ARMCM3\_EFM32\_Olimex\_EM32G880F128ST↔  
K\_GCC/Boot/vectors.c, [3623](#)
  - ARMCM3\_EFM32\_Olimex\_EM32G880F128ST↔  
K\_GCC/Prog/cstart.c, [1249](#)
  - ARMCM3\_EFM32\_Olimex\_EM32G880F128ST↔  
K\_GCC/Prog/vectors.c, [3625](#)
  - ARMCM3\_EFM32\_Olimex\_EM32G880F128ST↔  
K\_IAR/Boot/vectors.c, [3627](#)
  - ARMCM3\_LM3S\_EK\_LM3S6965\_GCC/Boot/cstart.↔  
c, [1250](#)
  - ARMCM3\_LM3S\_EK\_LM3S6965\_GCC/Boot/vectors.↔  
c, [3629](#)
  - ARMCM3\_LM3S\_EK\_LM3S6965\_GCC/Prog/cstart.↔  
c, [1251](#)
  - ARMCM3\_LM3S\_EK\_LM3S6965\_GCC/Prog/vectors.↔  
c, [3631](#)
  - ARMCM3\_LM3S\_EK\_LM3S6965\_IAR/Boot/vectors.↔  
c, [3632](#)
  - ARMCM3\_LM3S\_EK\_LM3S8962\_GCC/Boot/cstart.↔  
c, [1253](#)
  - ARMCM3\_LM3S\_EK\_LM3S8962\_GCC/Boot/vectors.↔  
c, [3635](#)
  - ARMCM3\_LM3S\_EK\_LM3S8962\_GCC/Prog/cstart.↔  
c, [1254](#)
  - ARMCM3\_LM3S\_EK\_LM3S8962\_GCC/Prog/vectors.↔  
c, [3636](#)
  - ARMCM3\_LM3S\_EK\_LM3S8962\_IAR/Boot/vectors.↔  
c, [3638](#)
  - ARMCM4\_TM4C\_DK\_TM4C123G\_IAR/Boot/vectors.↔  
c, [3640](#)
  - HCS12\_Evbplus\_Dragon12p\_CodeWarrior/↔  
Boot/vectors.c, [3670](#)
- ResetISR
  - ARMCM4\_TM4C\_DK\_TM4C123G\_IAR/Prog/vectors.↔  
c, [3642](#)
- rs232.c, [3058–3066](#)
- rs232.h, [3067](#)
- Rs232ReceiveByte
  - Demo/\_template/Prog/boot.c, [618](#)
  - Demo/ARMCM0\_STM32F0\_Discovery\_STM32↔  
F051\_CubeIDE/Prog/App/boot.c, [621](#)
  - Demo/ARMCM0\_STM32F0\_Discovery\_STM32↔  
F051\_GCC/Prog/boot.c, [624](#)
  - Demo/ARMCM0\_STM32F0\_Discovery\_STM32↔  
F051\_IAR/Prog/boot.c, [627](#)
  - Demo/ARMCM0\_STM32F0\_Discovery\_STM32↔  
F051\_Keil/Prog/boot.c, [630](#)
  - Demo/ARMCM0\_STM32F0\_Nucleo\_F091RC↔  
CubeIDE/Prog/App/boot.c, [634](#)
  - Demo/ARMCM0\_STM32F0\_Nucleo\_F091RC\_G↔  
CC/Prog/boot.c, [639](#)
  - Demo/ARMCM0\_STM32F0\_Nucleo\_F091RC\_I↔  
AR/Prog/boot.c, [644](#)
  - Demo/ARMCM0\_STM32F0\_Nucleo\_F091RC↔  
Keil/Prog/boot.c, [649](#)
  - Demo/ARMCM0\_STM32G0\_Nucleo\_G071RB↔  
CubeIDE/Prog/App/boot.c, [653](#)
  - Demo/ARMCM0\_STM32G0\_Nucleo\_G071RB↔  
GCC/Prog/boot.c, [656](#)
  - Demo/ARMCM0\_STM32G0\_Nucleo\_G071RB\_I↔  
AR/Prog/boot.c, [659](#)
  - Demo/ARMCM0\_STM32G0\_Nucleo\_G071RB↔  
Keil/Prog/boot.c, [662](#)
  - Demo/ARMCM0\_XMC1\_XMC1400\_Boot\_Kit\_G↔  
CC/Prog/boot.c, [666](#)
  - Demo/ARMCM0\_XMC1\_XMC1400\_Boot\_Kit\_IA↔  
R/Prog/boot.c, [669](#)
  - Demo/ARMCM33\_STM32L5\_Nucleo\_L552ZE↔

- CubeIDE/Prog/App/boot.c, [674](#)  
 Demo/ARMCM33\_STM32L5\_Nucleo\_L552ZE\_↔  
 GCC/Prog/boot.c, [679](#)  
 Demo/ARMCM33\_STM32L5\_Nucleo\_L552ZE\_I↔  
 AR/Prog/boot.c, [684](#)  
 Demo/ARMCM33\_STM32L5\_Nucleo\_L552ZE\_↔  
 Keil/Prog/boot.c, [689](#)  
 Demo/ARMCM3\_EFM32\_Olimex\_EM32G880\_↔  
 F128STK\_GCC/Prog/boot.c, [692](#)  
 Demo/ARMCM3\_EFM32\_Olimex\_EM32G880\_↔  
 F128STK\_IAR/Prog/boot.c, [695](#)  
 Demo/ARMCM3\_LM3S\_EK\_LM3S6965\_GCC/↔  
 Prog/boot.c, [698](#)  
 Demo/ARMCM3\_LM3S\_EK\_LM3S6965\_IAR/↔  
 Prog/boot.c, [700](#)  
 Demo/ARMCM3\_LM3S\_EK\_LM3S8962\_GCC/↔  
 Prog/boot.c, [704](#)  
 Demo/ARMCM3\_LM3S\_EK\_LM3S8962\_IAR/↔  
 Prog/boot.c, [708](#)  
 Demo/ARMCM3\_STM32F1\_Nucleo\_F103RB\_↔  
 CubeIDE/Prog/App/boot.c, [711](#)  
 Demo/ARMCM3\_STM32F1\_Nucleo\_F103RB\_G↔  
 CC/Prog/boot.c, [714](#)  
 Demo/ARMCM3\_STM32F1\_Nucleo\_F103RB\_I↔  
 AR/Prog/boot.c, [717](#)  
 Demo/ARMCM3\_STM32F1\_Nucleo\_F103RB\_↔  
 Keil/Prog/boot.c, [720](#)  
 Demo/ARMCM3\_STM32F1\_Olimex\_STM32\_↔  
 P103\_CubeIDE/Prog/App/boot.c, [724](#)  
 Demo/ARMCM3\_STM32F1\_Olimex\_STM32\_↔  
 P103\_GCC/Prog/boot.c, [729](#)  
 Demo/ARMCM3\_STM32F1\_Olimex\_STM32\_↔  
 P103\_IAR/Prog/boot.c, [734](#)  
 Demo/ARMCM3\_STM32F1\_Olimex\_STM32\_↔  
 P103\_Keil/Prog/boot.c, [739](#)  
 Demo/ARMCM3\_STM32F2\_Olimex\_STM32\_↔  
 P207\_CubeIDE/Prog/App/boot.c, [760](#)  
 Demo/ARMCM3\_STM32F2\_Olimex\_STM32\_↔  
 P207\_GCC/Prog/boot.c, [765](#)  
 Demo/ARMCM3\_STM32F2\_Olimex\_STM32\_↔  
 P207\_IAR/Prog/boot.c, [770](#)  
 Demo/ARMCM3\_STM32F2\_Olimex\_STM32\_↔  
 P207\_Keil/Prog/boot.c, [775](#)  
 Demo/ARMCM4\_S32K14\_S32K144EVB\_GCC/↔  
 Prog/boot.c, [782](#)  
 Demo/ARMCM4\_S32K14\_S32K144EVB\_IAR/↔  
 Prog/boot.c, [788](#)  
 Demo/ARMCM4\_STM32F3\_Nucleo\_F303K8\_↔  
 CubeIDE/Prog/App/boot.c, [793](#)  
 Demo/ARMCM4\_STM32F3\_Nucleo\_F303K8\_G↔  
 CC/Prog/boot.c, [798](#)  
 Demo/ARMCM4\_STM32F3\_Nucleo\_F303K8\_IA↔  
 R/Prog/boot.c, [803](#)  
 Demo/ARMCM4\_STM32F3\_Nucleo\_F303K8\_↔  
 Keil/Prog/boot.c, [808](#)  
 Demo/ARMCM4\_STM32F4\_Nucleo\_F429ZI\_↔  
 CubeIDE/Prog/App/boot.c, [813](#)  
 Demo/ARMCM4\_STM32F4\_Nucleo\_F429ZI\_G↔  
 CC/Prog/boot.c, [818](#)  
 Demo/ARMCM4\_STM32F4\_Nucleo\_F429ZI\_IA↔  
 R/Prog/boot.c, [823](#)  
 Demo/ARMCM4\_STM32F4\_Nucleo\_F429ZI\_↔  
 Keil/Prog/boot.c, [828](#)  
 Demo/ARMCM4\_STM32F4\_Olimex\_STM32\_↔  
 P405\_CubeIDE/Prog/App/boot.c, [833](#)  
 Demo/ARMCM4\_STM32F4\_Olimex\_STM32\_↔  
 P405\_GCC/Prog/boot.c, [838](#)  
 Demo/ARMCM4\_STM32F4\_Olimex\_STM32\_↔  
 P405\_IAR/Prog/boot.c, [843](#)  
 Demo/ARMCM4\_STM32F4\_Olimex\_STM32\_↔  
 P405\_Keil/Prog/boot.c, [848](#)  
 Demo/ARMCM4\_STM32L4\_Nucleo\_L476RG\_↔  
 CubeIDE/Prog/App/boot.c, [853](#)  
 Demo/ARMCM4\_STM32L4\_Nucleo\_L476RG\_G↔  
 CC/Prog/boot.c, [858](#)  
 Demo/ARMCM4\_STM32L4\_Nucleo\_L476RG\_I↔  
 AR/Prog/boot.c, [863](#)  
 Demo/ARMCM4\_STM32L4\_Nucleo\_L476RG\_↔  
 Keil/Prog/boot.c, [868](#)  
 Demo/ARMCM4\_TM4C\_DK\_TM4C123G\_IAR/↔  
 Prog/boot.c, [871](#)  
 Demo/ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_↔  
 GCC/Prog/boot.c, [875](#)  
 Demo/ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_I↔  
 AR/Prog/boot.c, [878](#)  
 Demo/ARMCM7\_STM32F7\_Nucleo\_F746ZG\_↔  
 CubeIDE/Prog/App/boot.c, [883](#)  
 Demo/ARMCM7\_STM32F7\_Nucleo\_F746ZG\_G↔  
 CC/Prog/boot.c, [888](#)  
 Demo/ARMCM7\_STM32F7\_Nucleo\_F746ZG\_I↔  
 AR/Prog/boot.c, [893](#)  
 Demo/ARMCM7\_STM32F7\_Nucleo\_F746ZG\_↔  
 Keil/Prog/boot.c, [898](#)  
 Demo/ARMCM7\_STM32F7\_Nucleo\_F767ZI\_↔  
 CubeIDE/Prog/App/boot.c, [901](#)  
 Demo/ARMCM7\_STM32F7\_Nucleo\_F767ZI\_G↔  
 CC/Prog/boot.c, [904](#)  
 Demo/ARMCM7\_STM32F7\_Nucleo\_F767ZI\_IA↔  
 R/Prog/boot.c, [907](#)  
 Demo/ARMCM7\_STM32F7\_Nucleo\_F767ZI\_↔  
 Keil/Prog/boot.c, [910](#)  
 Demo/ARMCM7\_STM32H7\_Nucleo\_H743ZI\_↔  
 CubeIDE/Prog/App/boot.c, [915](#)  
 Demo/ARMCM7\_STM32H7\_Nucleo\_H743ZI\_G↔  
 CC/Prog/boot.c, [920](#)  
 Demo/ARMCM7\_STM32H7\_Nucleo\_H743ZI\_IA↔  
 R/Prog/boot.c, [925](#)  
 Demo/ARMCM7\_STM32H7\_Nucleo\_H743ZI\_↔  
 Keil/Prog/boot.c, [930](#)  
 Demo/HCS12\_Evbplus\_Dragon12p\_Code↔  
 Warrior/Prog/boot.c, [935](#)  
 rxSlot  
 tCanRegs, [363](#)  
 s\_n\_k\_resource  
 tXcplInfo, [385](#)  
 sector\_num

tFlashSector, [376](#)  
 sector\_size  
   tFlashSector, [376](#)  
 sector\_start  
   tFlashSector, [376](#)  
 shared  
   ARMCM3\_LM3S\_EK\_LM3S6965\_IAR/Boot/shared↔  
     \_params.c, [3079](#)  
   ARMCM3\_LM3S\_EK\_LM3S6965\_IAR/Prog/shared↔  
     \_params.c, [3083](#)  
   ARMCM4\_STM32F4\_Nucleo\_F429ZI\_IAR/↔  
     Boot/shared\_params.c, [3105](#)  
   ARMCM4\_STM32F4\_Nucleo\_F429ZI\_IAR/↔  
     Prog/shared\_params.c, [3109](#)  
   ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_IAR/↔  
     Boot/shared\_params.c, [3131](#)  
   ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_IAR/↔  
     Prog/shared\_params.c, [3135](#)  
   ARMCM7\_STM32F7\_Nucleo\_F767ZI\_IAR/↔  
     Boot/shared\_params.c, [3157](#)  
   ARMCM7\_STM32F7\_Nucleo\_F767ZI\_IAR/↔  
     Prog/shared\_params.c, [3161](#)  
 shared\_params.c, [3067](#), [3071](#), [3076](#), [3080](#), [3084](#), [3088](#),  
   [3092](#), [3097](#), [3101](#), [3105](#), [3110](#), [3114](#), [3118](#),  
   [3123](#), [3127](#), [3131](#), [3136](#), [3140](#), [3144](#), [3149](#),  
   [3153](#), [3157](#), [3162](#), [3166](#)  
 shared\_params.h, [3170](#), [3172](#), [3175](#), [3177](#), [3179](#), [3182](#),  
   [3184](#), [3186](#), [3189](#), [3191](#), [3193](#), [3196](#), [3198](#),  
   [3200](#), [3203](#), [3205](#), [3207](#), [3210](#), [3212](#), [3214](#),  
   [3217](#)  
 SharedParamsCalculateChecksum  
   ARMCM3\_LM3S\_EK\_LM3S6965\_GCC/Boot/shared↔  
     \_params.c, [3069](#)  
   ARMCM3\_LM3S\_EK\_LM3S6965\_GCC/Prog/shared↔  
     \_params.c, [3073](#)  
   ARMCM3\_LM3S\_EK\_LM3S6965\_IAR/Boot/shared↔  
     \_params.c, [3077](#)  
   ARMCM3\_LM3S\_EK\_LM3S6965\_IAR/Prog/shared↔  
     \_params.c, [3081](#)  
   ARMCM4\_STM32F4\_Nucleo\_F429ZI\_CubelD↔  
     E/Boot/App/shared\_params.c, [3085](#)  
   ARMCM4\_STM32F4\_Nucleo\_F429ZI\_CubelD↔  
     E/Prog/App/shared\_params.c, [3090](#)  
   ARMCM4\_STM32F4\_Nucleo\_F429ZI\_GCC/↔  
     Boot/shared\_params.c, [3094](#)  
   ARMCM4\_STM32F4\_Nucleo\_F429ZI\_GCC/↔  
     Prog/shared\_params.c, [3098](#)  
   ARMCM4\_STM32F4\_Nucleo\_F429ZI\_IAR/↔  
     Boot/shared\_params.c, [3102](#)  
   ARMCM4\_STM32F4\_Nucleo\_F429ZI\_IAR/↔  
     Prog/shared\_params.c, [3107](#)  
   ARMCM4\_STM32F4\_Nucleo\_F429ZI\_Keil/↔  
     Boot/shared\_params.c, [3111](#)  
   ARMCM4\_STM32F4\_Nucleo\_F429ZI\_Keil/↔  
     Prog/shared\_params.c, [3116](#)  
   ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_GCC/↔  
     Boot/shared\_params.c, [3120](#)  
   ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_GCC/↔  
     Prog/shared\_params.c, [3124](#)  
   ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_IAR/↔  
     Boot/shared\_params.c, [3128](#)  
   ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_IAR/↔  
     Prog/shared\_params.c, [3133](#)  
   ARMCM7\_STM32F7\_Nucleo\_F767ZI\_CubelD↔  
     E/Boot/App/shared\_params.c, [3137](#)  
   ARMCM7\_STM32F7\_Nucleo\_F767ZI\_CubelD↔  
     E/Prog/App/shared\_params.c, [3142](#)  
   ARMCM7\_STM32F7\_Nucleo\_F767ZI\_GCC/↔  
     Boot/shared\_params.c, [3146](#)  
   ARMCM7\_STM32F7\_Nucleo\_F767ZI\_GCC/↔  
     Prog/shared\_params.c, [3150](#)  
   ARMCM7\_STM32F7\_Nucleo\_F767ZI\_IAR/↔  
     Boot/shared\_params.c, [3154](#)  
   ARMCM7\_STM32F7\_Nucleo\_F767ZI\_IAR/↔  
     Prog/shared\_params.c, [3159](#)  
   ARMCM7\_STM32F7\_Nucleo\_F767ZI\_Keil/↔  
     Boot/shared\_params.c, [3163](#)  
   ARMCM7\_STM32F7\_Nucleo\_F767ZI\_Keil/↔  
     Prog/shared\_params.c, [3168](#)  
 SharedParamsInit  
   ARMCM3\_LM3S\_EK\_LM3S6965\_GCC/Boot/shared↔  
     \_params.c, [3069](#)  
   ARMCM3\_LM3S\_EK\_LM3S6965\_GCC/Boot/shared↔  
     \_params.h, [3171](#)  
   ARMCM3\_LM3S\_EK\_LM3S6965\_GCC/Prog/shared↔  
     \_params.c, [3073](#)  
   ARMCM3\_LM3S\_EK\_LM3S6965\_GCC/Prog/shared↔  
     \_params.h, [3174](#)  
   ARMCM3\_LM3S\_EK\_LM3S6965\_IAR/Boot/shared↔  
     \_params.c, [3077](#)  
   ARMCM3\_LM3S\_EK\_LM3S6965\_IAR/Boot/shared↔  
     \_params.h, [3176](#)  
   ARMCM3\_LM3S\_EK\_LM3S6965\_IAR/Prog/shared↔  
     \_params.c, [3081](#)  
   ARMCM3\_LM3S\_EK\_LM3S6965\_IAR/Prog/shared↔  
     \_params.h, [3178](#)  
   ARMCM4\_STM32F4\_Nucleo\_F429ZI\_CubelD↔  
     E/Boot/App/shared\_params.c, [3086](#)  
   ARMCM4\_STM32F4\_Nucleo\_F429ZI\_CubelD↔  
     E/Boot/App/shared\_params.h, [3181](#)  
   ARMCM4\_STM32F4\_Nucleo\_F429ZI\_CubelD↔  
     E/Prog/App/shared\_params.c, [3090](#)  
   ARMCM4\_STM32F4\_Nucleo\_F429ZI\_CubelD↔  
     E/Prog/App/shared\_params.h, [3183](#)  
   ARMCM4\_STM32F4\_Nucleo\_F429ZI\_GCC/↔  
     Boot/shared\_params.c, [3094](#)  
   ARMCM4\_STM32F4\_Nucleo\_F429ZI\_GCC/↔  
     Prog/shared\_params.c, [3099](#)  
   ARMCM4\_STM32F4\_Nucleo\_F429ZI\_GCC/↔  
     Prog/shared\_params.h, [3185](#)  
   ARMCM4\_STM32F4\_Nucleo\_F429ZI\_IAR/↔  
     Boot/shared\_params.c, [3102](#)  
   ARMCM4\_STM32F4\_Nucleo\_F429ZI\_IAR/↔  
     Prog/shared\_params.c, [3107](#)  
   ARMCM4\_STM32F4\_Nucleo\_F429ZI\_IAR/↔  
     Prog/shared\_params.h, [3188](#)



- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_Keil/↔  
  Boot/shared\_params.c, [3112](#)
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_Keil/↔  
  Prog/shared\_params.c, [3116](#)
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_Keil/↔  
  Prog/shared\_params.h, [3190](#)
- ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_GCC/↔  
  Boot/shared\_params.c, [3120](#)
- ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_GCC/↔  
  Boot/shared\_params.h, [3192](#)
- ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_GCC/↔  
  Prog/shared\_params.c, [3125](#)
- ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_GCC/↔  
  Prog/shared\_params.h, [3195](#)
- ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_IAR/↔  
  Boot/shared\_params.c, [3128](#)
- ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_IAR/↔  
  Boot/shared\_params.h, [3197](#)
- ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_IAR/↔  
  Prog/shared\_params.c, [3133](#)
- ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_IAR/↔  
  Prog/shared\_params.h, [3199](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_CubelD↔  
  E/Boot/App/shared\_params.c, [3138](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_CubelD↔  
  E/Boot/App/shared\_params.h, [3202](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_CubelD↔  
  E/Prog/App/shared\_params.c, [3142](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_CubelD↔  
  E/Prog/App/shared\_params.h, [3204](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_GCC/↔  
  Boot/shared\_params.c, [3146](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_GCC/↔  
  Boot/shared\_params.h, [3206](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_GCC/↔  
  Prog/shared\_params.c, [3151](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_GCC/↔  
  Prog/shared\_params.h, [3209](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_IAR/↔  
  Boot/shared\_params.c, [3154](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_IAR/↔  
  Boot/shared\_params.h, [3211](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_IAR/↔  
  Prog/shared\_params.c, [3159](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_IAR/↔  
  Prog/shared\_params.h, [3213](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_Keil/↔  
  Boot/shared\_params.c, [3164](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_Keil/↔  
  Boot/shared\_params.h, [3216](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_Keil/↔  
  Prog/shared\_params.c, [3168](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_Keil/↔  
  Prog/shared\_params.h, [3218](#)
- SharedParamsReadByIndex
  - ARMCM3\_LM3S\_EK\_LM3S6965\_GCC/Boot/shared↔  
  \_params.c, [3069](#)
  - ARMCM3\_LM3S\_EK\_LM3S6965\_GCC/Boot/shared↔  
  \_params.h, [3171](#)
  - ARMCM3\_LM3S\_EK\_LM3S6965\_GCC/Prog/shared↔  
  \_params.c, [3074](#)
  - ARMCM3\_LM3S\_EK\_LM3S6965\_GCC/Prog/shared↔  
  \_params.h, [3174](#)
  - ARMCM3\_LM3S\_EK\_LM3S6965\_IAR/Boot/shared↔  
  \_params.c, [3077](#)
  - ARMCM3\_LM3S\_EK\_LM3S6965\_IAR/Boot/shared↔  
  \_params.h, [3176](#)
  - ARMCM3\_LM3S\_EK\_LM3S6965\_IAR/Prog/shared↔  
  \_params.c, [3081](#)
  - ARMCM3\_LM3S\_EK\_LM3S6965\_IAR/Prog/shared↔  
  \_params.h, [3178](#)
  - ARMCM4\_STM32F4\_Nucleo\_F429ZI\_CubelD↔  
  E/Boot/App/shared\_params.c, [3086](#)
  - ARMCM4\_STM32F4\_Nucleo\_F429ZI\_CubelD↔  
  E/Boot/App/shared\_params.h, [3181](#)
  - ARMCM4\_STM32F4\_Nucleo\_F429ZI\_CubelD↔  
  E/Prog/App/shared\_params.c, [3090](#)
  - ARMCM4\_STM32F4\_Nucleo\_F429ZI\_CubelD↔  
  E/Prog/App/shared\_params.h, [3183](#)
  - ARMCM4\_STM32F4\_Nucleo\_F429ZI\_GCC/↔  
  Boot/shared\_params.c, [3095](#)
  - ARMCM4\_STM32F4\_Nucleo\_F429ZI\_GCC/↔  
  Prog/shared\_params.c, [3099](#)
  - ARMCM4\_STM32F4\_Nucleo\_F429ZI\_GCC/↔  
  Prog/shared\_params.h, [3185](#)
  - ARMCM4\_STM32F4\_Nucleo\_F429ZI\_IAR/↔  
  Boot/shared\_params.c, [3103](#)
  - ARMCM4\_STM32F4\_Nucleo\_F429ZI\_IAR/↔  
  Prog/shared\_params.c, [3107](#)
  - ARMCM4\_STM32F4\_Nucleo\_F429ZI\_IAR/↔  
  Prog/shared\_params.h, [3188](#)
  - ARMCM4\_STM32F4\_Nucleo\_F429ZI\_Keil/↔  
  Boot/shared\_params.c, [3112](#)
  - ARMCM4\_STM32F4\_Nucleo\_F429ZI\_Keil/↔  
  Prog/shared\_params.c, [3116](#)
  - ARMCM4\_STM32F4\_Nucleo\_F429ZI\_Keil/↔  
  Prog/shared\_params.h, [3190](#)
  - ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_GCC/↔  
  Boot/shared\_params.c, [3121](#)
  - ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_GCC/↔  
  Boot/shared\_params.h, [3192](#)
  - ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_GCC/↔  
  Prog/shared\_params.c, [3125](#)
  - ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_GCC/↔  
  Prog/shared\_params.h, [3195](#)
  - ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_IAR/↔  
  Boot/shared\_params.c, [3129](#)
  - ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_IAR/↔  
  Boot/shared\_params.h, [3197](#)
  - ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_IAR/↔  
  Prog/shared\_params.c, [3133](#)
  - ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_IAR/↔  
  Prog/shared\_params.h, [3199](#)
  - ARMCM7\_STM32F7\_Nucleo\_F767ZI\_CubelD↔  
  E/Boot/App/shared\_params.c, [3138](#)
  - ARMCM7\_STM32F7\_Nucleo\_F767ZI\_CubelD↔

- E/Boot/App/shared\_params.h, [3202](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_CubeID↔
  - E/Prog/App/shared\_params.c, [3142](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_CubeID↔
  - E/Prog/App/shared\_params.h, [3204](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_GCC↔
  - Boot/shared\_params.c, [3147](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_GCC↔
  - Boot/shared\_params.h, [3206](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_GCC↔
  - Prog/shared\_params.c, [3151](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_GCC↔
  - Prog/shared\_params.h, [3209](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_IAR↔
  - Boot/shared\_params.c, [3155](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_IAR↔
  - Boot/shared\_params.h, [3211](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_IAR↔
  - Prog/shared\_params.c, [3159](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_IAR↔
  - Prog/shared\_params.h, [3213](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_Keil↔
  - Boot/shared\_params.c, [3164](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_Keil↔
  - Boot/shared\_params.h, [3216](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_Keil↔
  - Prog/shared\_params.c, [3168](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_Keil↔
  - Prog/shared\_params.h, [3218](#)
- SharedParamsValidateBuffer
  - ARMCM3\_LM3S\_EK\_LM3S6965\_GCC/Boot/shared↔
    - \_params.c, [3070](#)
  - ARMCM3\_LM3S\_EK\_LM3S6965\_GCC/Prog/shared↔
    - \_params.c, [3074](#)
  - ARMCM3\_LM3S\_EK\_LM3S6965\_IAR/Boot/shared↔
    - \_params.c, [3078](#)
  - ARMCM3\_LM3S\_EK\_LM3S6965\_IAR/Prog/shared↔
    - \_params.c, [3082](#)
  - ARMCM4\_STM32F4\_Nucleo\_F429ZI\_CubeID↔
    - E/Boot/App/shared\_params.c, [3086](#)
  - ARMCM4\_STM32F4\_Nucleo\_F429ZI\_CubeID↔
    - E/Prog/App/shared\_params.c, [3091](#)
  - ARMCM4\_STM32F4\_Nucleo\_F429ZI\_GCC↔
    - Boot/shared\_params.c, [3095](#)
  - ARMCM4\_STM32F4\_Nucleo\_F429ZI\_GCC↔
    - Prog/shared\_params.c, [3099](#)
  - ARMCM4\_STM32F4\_Nucleo\_F429ZI\_IAR↔
    - Boot/shared\_params.c, [3103](#)
  - ARMCM4\_STM32F4\_Nucleo\_F429ZI\_IAR↔
    - Prog/shared\_params.c, [3108](#)
  - ARMCM4\_STM32F4\_Nucleo\_F429ZI\_Keil↔
    - Boot/shared\_params.c, [3112](#)
  - ARMCM4\_STM32F4\_Nucleo\_F429ZI\_Keil↔
    - Prog/shared\_params.c, [3117](#)
  - ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_GCC↔
    - Boot/shared\_params.c, [3121](#)
  - ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_GCC↔
    - Prog/shared\_params.c, [3125](#)
  - ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_IAR↔
    - Boot/shared\_params.c, [3129](#)
  - ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_IAR↔
    - Prog/shared\_params.c, [3134](#)
  - ARMCM7\_STM32F7\_Nucleo\_F767ZI\_CubeID↔
    - E/Boot/App/shared\_params.c, [3139](#)
  - ARMCM7\_STM32F7\_Nucleo\_F767ZI\_CubeID↔
    - E/Prog/App/shared\_params.c, [3143](#)
  - ARMCM7\_STM32F7\_Nucleo\_F767ZI\_GCC↔
    - Boot/shared\_params.c, [3147](#)
  - ARMCM7\_STM32F7\_Nucleo\_F767ZI\_GCC↔
    - Prog/shared\_params.c, [3151](#)
  - ARMCM7\_STM32F7\_Nucleo\_F767ZI\_IAR↔
    - Boot/shared\_params.c, [3155](#)
  - ARMCM7\_STM32F7\_Nucleo\_F767ZI\_IAR↔
    - Prog/shared\_params.c, [3160](#)
  - ARMCM7\_STM32F7\_Nucleo\_F767ZI\_Keil↔
    - Boot/shared\_params.c, [3164](#)
  - ARMCM7\_STM32F7\_Nucleo\_F767ZI\_Keil↔
    - Prog/shared\_params.c, [3169](#)
- SharedParamsVerifyChecksum
  - ARMCM3\_LM3S\_EK\_LM3S6965\_GCC/Boot/shared↔
    - \_params.c, [3070](#)
  - ARMCM3\_LM3S\_EK\_LM3S6965\_GCC/Prog/shared↔
    - \_params.c, [3074](#)
  - ARMCM3\_LM3S\_EK\_LM3S6965\_IAR/Boot/shared↔
    - \_params.c, [3078](#)
  - ARMCM3\_LM3S\_EK\_LM3S6965\_IAR/Prog/shared↔
    - \_params.c, [3082](#)
  - ARMCM4\_STM32F4\_Nucleo\_F429ZI\_CubeID↔
    - E/Boot/App/shared\_params.c, [3087](#)
  - ARMCM4\_STM32F4\_Nucleo\_F429ZI\_CubeID↔
    - E/Prog/App/shared\_params.c, [3091](#)
  - ARMCM4\_STM32F4\_Nucleo\_F429ZI\_GCC↔
    - Boot/shared\_params.c, [3095](#)
  - ARMCM4\_STM32F4\_Nucleo\_F429ZI\_GCC↔
    - Prog/shared\_params.c, [3100](#)
  - ARMCM4\_STM32F4\_Nucleo\_F429ZI\_IAR↔
    - Boot/shared\_params.c, [3103](#)
  - ARMCM4\_STM32F4\_Nucleo\_F429ZI\_IAR↔
    - Prog/shared\_params.c, [3108](#)
  - ARMCM4\_STM32F4\_Nucleo\_F429ZI\_Keil↔
    - Boot/shared\_params.c, [3113](#)
  - ARMCM4\_STM32F4\_Nucleo\_F429ZI\_Keil↔
    - Prog/shared\_params.c, [3117](#)
  - ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_GCC↔
    - Boot/shared\_params.c, [3121](#)
  - ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_GCC↔
    - Prog/shared\_params.c, [3126](#)
  - ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_IAR↔
    - Boot/shared\_params.c, [3129](#)
  - ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_IAR↔
    - Prog/shared\_params.c, [3134](#)
  - ARMCM7\_STM32F7\_Nucleo\_F767ZI\_CubeID↔
    - E/Boot/App/shared\_params.c, [3139](#)
  - ARMCM7\_STM32F7\_Nucleo\_F767ZI\_CubeID↔
    - E/Prog/App/shared\_params.c, [3143](#)
  - ARMCM7\_STM32F7\_Nucleo\_F767ZI\_GCC↔
    - Boot/shared\_params.c, [3147](#)

- Boot/shared\_params.c, [3147](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_GCC/↔
  - Prog/shared\_params.c, [3152](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_IAR/↔
  - Boot/shared\_params.c, [3155](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_IAR/↔
  - Prog/shared\_params.c, [3160](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_Keil/↔
  - Boot/shared\_params.c, [3165](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_Keil/↔
  - Prog/shared\_params.c, [3169](#)
- SharedParamsWriteByIndex
  - ARMCM3\_LM3S\_EK\_LM3S6965\_GCC/Boot/shared↔
    - \_params.c, [3070](#)
  - ARMCM3\_LM3S\_EK\_LM3S6965\_GCC/Boot/shared↔
    - \_params.h, [3172](#)
  - ARMCM3\_LM3S\_EK\_LM3S6965\_GCC/Prog/shared↔
    - \_params.c, [3075](#)
  - ARMCM3\_LM3S\_EK\_LM3S6965\_GCC/Prog/shared↔
    - \_params.h, [3174](#)
  - ARMCM3\_LM3S\_EK\_LM3S6965\_IAR/Boot/shared↔
    - \_params.c, [3078](#)
  - ARMCM3\_LM3S\_EK\_LM3S6965\_IAR/Boot/shared↔
    - \_params.h, [3177](#)
  - ARMCM3\_LM3S\_EK\_LM3S6965\_IAR/Prog/shared↔
    - \_params.c, [3082](#)
  - ARMCM3\_LM3S\_EK\_LM3S6965\_IAR/Prog/shared↔
    - \_params.h, [3179](#)
  - ARMCM4\_STM32F4\_Nucleo\_F429ZI\_CubelD↔
    - E/Boot/App/shared\_params.c, [3087](#)
  - ARMCM4\_STM32F4\_Nucleo\_F429ZI\_CubelD↔
    - E/Boot/App/shared\_params.h, [3181](#)
  - ARMCM4\_STM32F4\_Nucleo\_F429ZI\_CubelD↔
    - E/Prog/App/shared\_params.c, [3091](#)
  - ARMCM4\_STM32F4\_Nucleo\_F429ZI\_CubelD↔
    - E/Prog/App/shared\_params.h, [3184](#)
  - ARMCM4\_STM32F4\_Nucleo\_F429ZI\_GCC/↔
    - Boot/shared\_params.c, [3096](#)
  - ARMCM4\_STM32F4\_Nucleo\_F429ZI\_GCC/↔
    - Prog/shared\_params.c, [3100](#)
  - ARMCM4\_STM32F4\_Nucleo\_F429ZI\_GCC/↔
    - Prog/shared\_params.h, [3186](#)
  - ARMCM4\_STM32F4\_Nucleo\_F429ZI\_IAR/↔
    - Boot/shared\_params.c, [3104](#)
  - ARMCM4\_STM32F4\_Nucleo\_F429ZI\_IAR/↔
    - Prog/shared\_params.c, [3108](#)
  - ARMCM4\_STM32F4\_Nucleo\_F429ZI\_IAR/↔
    - Prog/shared\_params.h, [3188](#)
  - ARMCM4\_STM32F4\_Nucleo\_F429ZI\_Keil/↔
    - Boot/shared\_params.c, [3113](#)
  - ARMCM4\_STM32F4\_Nucleo\_F429ZI\_Keil/↔
    - Prog/shared\_params.c, [3117](#)
  - ARMCM4\_STM32F4\_Nucleo\_F429ZI\_Keil/↔
    - Prog/shared\_params.h, [3191](#)
  - ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_GCC/↔
    - Boot/shared\_params.c, [3122](#)
  - ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_GCC/↔
    - Boot/shared\_params.h, [3193](#)
  - ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_GCC/↔
    - Prog/shared\_params.c, [3126](#)
  - ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_GCC/↔
    - Prog/shared\_params.h, [3195](#)
  - ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_IAR/↔
    - Boot/shared\_params.c, [3130](#)
  - ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_IAR/↔
    - Boot/shared\_params.h, [3198](#)
  - ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_IAR/↔
    - Prog/shared\_params.c, [3134](#)
  - ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_IAR/↔
    - Prog/shared\_params.h, [3200](#)
  - ARMCM7\_STM32F7\_Nucleo\_F767ZI\_CubelD↔
    - E/Boot/App/shared\_params.c, [3139](#)
  - ARMCM7\_STM32F7\_Nucleo\_F767ZI\_CubelD↔
    - E/Boot/App/shared\_params.h, [3202](#)
  - ARMCM7\_STM32F7\_Nucleo\_F767ZI\_CubelD↔
    - E/Prog/App/shared\_params.c, [3143](#)
  - ARMCM7\_STM32F7\_Nucleo\_F767ZI\_CubelD↔
    - E/Prog/App/shared\_params.h, [3205](#)
  - ARMCM7\_STM32F7\_Nucleo\_F767ZI\_GCC/↔
    - Boot/shared\_params.c, [3148](#)
  - ARMCM7\_STM32F7\_Nucleo\_F767ZI\_GCC/↔
    - Boot/shared\_params.h, [3207](#)
  - ARMCM7\_STM32F7\_Nucleo\_F767ZI\_GCC/↔
    - Prog/shared\_params.c, [3152](#)
  - ARMCM7\_STM32F7\_Nucleo\_F767ZI\_GCC/↔
    - Prog/shared\_params.h, [3209](#)
  - ARMCM7\_STM32F7\_Nucleo\_F767ZI\_IAR/↔
    - Boot/shared\_params.c, [3156](#)
  - ARMCM7\_STM32F7\_Nucleo\_F767ZI\_IAR/↔
    - Boot/shared\_params.h, [3212](#)
  - ARMCM7\_STM32F7\_Nucleo\_F767ZI\_IAR/↔
    - Prog/shared\_params.c, [3160](#)
  - ARMCM7\_STM32F7\_Nucleo\_F767ZI\_IAR/↔
    - Prog/shared\_params.h, [3214](#)
  - ARMCM7\_STM32F7\_Nucleo\_F767ZI\_Keil/↔
    - Boot/shared\_params.c, [3165](#)
  - ARMCM7\_STM32F7\_Nucleo\_F767ZI\_Keil/↔
    - Boot/shared\_params.h, [3216](#)
  - ARMCM7\_STM32F7\_Nucleo\_F767ZI\_Keil/↔
    - Prog/shared\_params.c, [3169](#)
  - ARMCM7\_STM32F7\_Nucleo\_F767ZI\_Keil/↔
    - Prog/shared\_params.h, [3219](#)
- SharedParamsWriteChecksum
  - ARMCM3\_LM3S\_EK\_LM3S6965\_GCC/Boot/shared↔
    - \_params.c, [3071](#)
  - ARMCM3\_LM3S\_EK\_LM3S6965\_GCC/Prog/shared↔
    - \_params.c, [3075](#)
  - ARMCM3\_LM3S\_EK\_LM3S6965\_IAR/Boot/shared↔
    - \_params.c, [3079](#)
  - ARMCM3\_LM3S\_EK\_LM3S6965\_IAR/Prog/shared↔
    - \_params.c, [3083](#)
  - ARMCM4\_STM32F4\_Nucleo\_F429ZI\_CubelD↔
    - E/Boot/App/shared\_params.c, [3087](#)
  - ARMCM4\_STM32F4\_Nucleo\_F429ZI\_CubelD↔
    - E/Prog/App/shared\_params.c, [3092](#)
  - ARMCM4\_STM32F4\_Nucleo\_F429ZI\_GCC/↔

- Boot/shared\_params.c, [3096](#)
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_GCC/↔
  - Prog/shared\_params.c, [3100](#)
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_IAR/↔
  - Boot/shared\_params.c, [3104](#)
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_IAR/↔
  - Prog/shared\_params.c, [3109](#)
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_Keil/↔
  - Boot/shared\_params.c, [3113](#)
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_Keil/↔
  - Prog/shared\_params.c, [3118](#)
- ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_GCC/↔
  - Boot/shared\_params.c, [3122](#)
- ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_GCC/↔
  - Prog/shared\_params.c, [3126](#)
- ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_IAR/↔
  - Boot/shared\_params.c, [3130](#)
- ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_IAR/↔
  - Prog/shared\_params.c, [3135](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_CubeID↔
  - E/Boot/App/shared\_params.c, [3139](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_CubeID↔
  - E/Prog/App/shared\_params.c, [3144](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_GCC/↔
  - Boot/shared\_params.c, [3148](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_GCC/↔
  - Prog/shared\_params.c, [3152](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_IAR/↔
  - Boot/shared\_params.c, [3156](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_IAR/↔
  - Prog/shared\_params.c, [3161](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_Keil/↔
  - Boot/shared\_params.c, [3165](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_Keil/↔
  - Prog/shared\_params.c, [3170](#)
- Source/\_template/timer.c
  - TimerGet, [3365](#)
  - TimerInit, [3365](#)
  - TimerReset, [3365](#)
  - TimerUpdate, [3366](#)
- Source/ARMCM0\_S32K11/timer.c
  - TimerGet, [3367](#)
  - TimerInit, [3367](#)
  - TimerReset, [3367](#)
  - TimerUpdate, [3368](#)
- Source/ARMCM0\_STM32F0/timer.c
  - HAL\_GetTick, [3369](#)
  - SysTick\_Handler, [3369](#)
  - TimerGet, [3370](#)
  - TimerInit, [3370](#)
  - TimerReset, [3370](#)
  - TimerUpdate, [3371](#)
- Source/ARMCM0\_STM32G0/timer.c
  - HAL\_GetTick, [3372](#)
  - SysTick\_Handler, [3372](#)
  - TimerGet, [3373](#)
  - TimerInit, [3373](#)
  - TimerReset, [3373](#)
- TimerUpdate, [3374](#)
- Source/ARMCM0\_XMC1/timer.c
  - TimerGet, [3375](#)
  - TimerInit, [3376](#)
  - TimerReset, [3376](#)
  - TimerUpdate, [3376](#)
- Source/ARMCM33\_STM32L5/timer.c
  - HAL\_GetTick, [3378](#)
  - SysTick\_Handler, [3378](#)
  - TimerGet, [3378](#)
  - TimerInit, [3378](#)
  - TimerReset, [3379](#)
  - TimerUpdate, [3379](#)
- Source/ARMCM3\_EFM32/timer.c
  - TimerGet, [3381](#)
  - TimerInit, [3381](#)
  - TimerReset, [3381](#)
  - TimerUpdate, [3382](#)
- Source/ARMCM3\_LM3S/timer.c
  - TimerGet, [3383](#)
  - TimerInit, [3384](#)
  - TimerReset, [3384](#)
  - TimerUpdate, [3384](#)
- Source/ARMCM3\_STM32F1/timer.c
  - HAL\_GetTick, [3386](#)
  - SysTick\_Handler, [3386](#)
  - TimerGet, [3386](#)
  - TimerInit, [3386](#)
  - TimerReset, [3387](#)
  - TimerUpdate, [3387](#)
- Source/ARMCM3\_STM32F2/timer.c
  - HAL\_GetTick, [3389](#)
  - SysTick\_Handler, [3389](#)
  - TimerGet, [3389](#)
  - TimerInit, [3389](#)
  - TimerReset, [3390](#)
  - TimerUpdate, [3390](#)
- Source/ARMCM4\_S32K14/timer.c
  - TimerGet, [3391](#)
  - TimerInit, [3392](#)
  - TimerReset, [3392](#)
  - TimerUpdate, [3392](#)
- Source/ARMCM4\_STM32F3/timer.c
  - HAL\_GetTick, [3394](#)
  - SysTick\_Handler, [3394](#)
  - TimerGet, [3394](#)
  - TimerInit, [3394](#)
  - TimerReset, [3395](#)
  - TimerUpdate, [3395](#)
- Source/ARMCM4\_STM32F4/timer.c
  - HAL\_GetTick, [3397](#)
  - SysTick\_Handler, [3397](#)
  - TimerGet, [3397](#)
  - TimerInit, [3397](#)
  - TimerReset, [3398](#)
  - TimerUpdate, [3398](#)
- Source/ARMCM4\_STM32L4/timer.c
  - HAL\_GetTick, [3400](#)



- SysTick\_Handler, [3400](#)
- TimerGet, [3400](#)
- TimerInit, [3400](#)
- TimerReset, [3401](#)
- TimerUpdate, [3401](#)
- Source/ARMCM4\_TM4C/timer.c
  - TimerGet, [3403](#)
  - TimerInit, [3403](#)
  - TimerReset, [3403](#)
  - TimerUpdate, [3404](#)
- Source/ARMCM4\_XMC4/timer.c
  - TimerGet, [3405](#)
  - TimerInit, [3406](#)
  - TimerReset, [3406](#)
  - TimerUpdate, [3406](#)
- Source/ARMCM7\_STM32F7/timer.c
  - HAL\_GetTick, [3408](#)
  - SysTick\_Handler, [3408](#)
  - TimerGet, [3408](#)
  - TimerInit, [3408](#)
  - TimerReset, [3409](#)
  - TimerUpdate, [3409](#)
- Source/ARMCM7\_STM32H7/timer.c
  - HAL\_GetTick, [3411](#)
  - SysTick\_Handler, [3411](#)
  - TimerGet, [3411](#)
  - TimerInit, [3411](#)
  - TimerReset, [3412](#)
  - TimerUpdate, [3412](#)
- Source/HCS12/timer.c
  - TimerGet, [3414](#)
  - TimerInit, [3414](#)
  - TimerReset, [3414](#)
  - TimerUpdate, [3415](#)
- Source/boot.c
  - BootInit, [937](#)
  - BootTask, [937](#)
- Source/boot.h
  - BootInit, [1062](#)
  - BootTask, [1062](#)
- Source/net.c
  - NetApp, [2930](#)
  - NetInit, [2930](#)
  - NetReceivePacket, [2930](#)
  - NetServerTask, [2931](#)
  - NetTransmitPacket, [2931](#)
- Source/net.h
  - NetApp, [2956](#)
  - NetInit, [2957](#)
  - NetReceivePacket, [2957](#)
  - NetTransmitPacket, [2958](#)
- Source/timer.h
  - TimerGet, [3528](#)
  - TimerInit, [3528](#)
  - TimerReset, [3529](#)
  - TimerUpdate, [3529](#)
- start\_address
  - tFileEraseInfo, [370](#)
- startptr
  - tFifoCtrl, [368](#)
- SysClockInit
  - HCS12\_Evbplus\_Dragon12p\_CodeWarrior/↔  
Prog/main.c, [2904](#)
- SysTick\_Handler
  - Demo/ARMCM0\_STM32F0\_Discovery\_STM32↔  
F051\_GCC/Prog/timer.c, [3245](#)
  - Demo/ARMCM0\_STM32F0\_Discovery\_STM32↔  
F051\_IAR/Prog/timer.c, [3247](#)
  - Demo/ARMCM0\_STM32F0\_Discovery\_STM32↔  
F051\_Keil/Prog/timer.c, [3248](#)
  - Demo/ARMCM0\_STM32F0\_Nucleo\_F091RC\_G↔  
CC/Prog/timer.c, [3251](#)
  - Demo/ARMCM0\_STM32F0\_Nucleo\_F091RC\_I↔  
AR/Prog/timer.c, [3253](#)
  - Demo/ARMCM0\_STM32F0\_Nucleo\_F091RC↔  
Keil/Prog/timer.c, [3254](#)
  - Demo/ARMCM0\_STM32G0\_Nucleo\_G071RB↔  
GCC/Prog/timer.c, [3257](#)
  - Demo/ARMCM0\_STM32G0\_Nucleo\_G071RB\_I↔  
AR/Prog/timer.c, [3258](#)
  - Demo/ARMCM0\_STM32G0\_Nucleo\_G071RB↔  
Keil/Prog/timer.c, [3260](#)
  - Demo/ARMCM0\_XMC1\_XMC1400\_Boot\_Kit\_G↔  
CC/Prog/timer.c, [3262](#)
  - Demo/ARMCM0\_XMC1\_XMC1400\_Boot\_Kit\_IA↔  
R/Prog/timer.c, [3264](#)
  - Demo/ARMCM33\_STM32L5\_Nucleo\_L552ZE↔  
GCC/Prog/timer.c, [3267](#)
  - Demo/ARMCM33\_STM32L5\_Nucleo\_L552ZE\_I↔  
AR/Prog/timer.c, [3269](#)
  - Demo/ARMCM33\_STM32L5\_Nucleo\_L552ZE↔  
Keil/Prog/timer.c, [3270](#)
  - Demo/ARMCM3\_STM32F1\_Nucleo\_F103RB\_G↔  
CC/Prog/timer.c, [3277](#)
  - Demo/ARMCM3\_STM32F1\_Nucleo\_F103RB\_I↔  
AR/Prog/timer.c, [3279](#)
  - Demo/ARMCM3\_STM32F1\_Nucleo\_F103RB↔  
Keil/Prog/timer.c, [3280](#)
  - Demo/ARMCM3\_STM32F1\_Olimex\_STM32↔  
H103\_GCC/Prog/timer.c, [3284](#)
  - Demo/ARMCM3\_STM32F1\_Olimex\_STM32↔  
H103\_IAR/Prog/timer.c, [3285](#)
  - Demo/ARMCM3\_STM32F1\_Olimex\_STM32↔  
H103\_Keil/Prog/timer.c, [3287](#)
  - Demo/ARMCM3\_STM32F1\_Olimex\_STM32↔  
P103\_GCC/Prog/timer.c, [3290](#)
  - Demo/ARMCM3\_STM32F1\_Olimex\_STM32↔  
P103\_IAR/Prog/timer.c, [3291](#)
  - Demo/ARMCM3\_STM32F1\_Olimex\_STM32↔  
P103\_Keil/Prog/timer.c, [3293](#)
  - Demo/ARMCM3\_STM32F1\_Olimexino\_STM32↔  
\_GCC/Prog/timer.c, [3295](#)
  - Demo/ARMCM3\_STM32F1\_Olimexino\_STM32↔  
\_IAR/Prog/timer.c, [3297](#)
  - Demo/ARMCM3\_STM32F1\_Olimexino\_STM32↔  
\_Keil/Prog/timer.c, [3298](#)

- Demo/ARMCM3\_STM32F2\_Olimex\_STM32↔  
P207\_GCC/Prog/timer.c, [3301](#)
- Demo/ARMCM3\_STM32F2\_Olimex\_STM32↔  
P207\_IAR/Prog/timer.c, [3303](#)
- Demo/ARMCM3\_STM32F2\_Olimex\_STM32↔  
P207\_Keil/Prog/timer.c, [3304](#)
- Demo/ARMCM4\_S32K14\_S32K144EVB\_GCC/↔  
Prog/timer.c, [3306](#)
- Demo/ARMCM4\_S32K14\_S32K144EVB\_IAR/↔  
Prog/timer.c, [3307](#)
- Demo/ARMCM4\_STM32F3\_Discovery\_F303VC↔  
\_GCC/Prog/timer.c, [3310](#)
- Demo/ARMCM4\_STM32F3\_Discovery\_F303VC↔  
\_IAR/Prog/timer.c, [3312](#)
- Demo/ARMCM4\_STM32F3\_Discovery\_F303VC↔  
\_Keil/Prog/timer.c, [3314](#)
- Demo/ARMCM4\_STM32F3\_Nucleo\_F303K8\_G↔  
CC/Prog/timer.c, [3316](#)
- Demo/ARMCM4\_STM32F3\_Nucleo\_F303K8\_IA↔  
R/Prog/timer.c, [3318](#)
- Demo/ARMCM4\_STM32F3\_Nucleo\_F303K8\_↔  
Keil/Prog/timer.c, [3319](#)
- Demo/ARMCM4\_STM32F4\_Nucleo\_F429ZI\_G↔  
CC/Prog/timer.c, [3322](#)
- Demo/ARMCM4\_STM32F4\_Nucleo\_F429ZI\_IA↔  
R/Prog/timer.c, [3324](#)
- Demo/ARMCM4\_STM32F4\_Nucleo\_F429ZI\_↔  
Keil/Prog/timer.c, [3325](#)
- Demo/ARMCM4\_STM32F4\_Olimex\_STM32↔  
P405\_GCC/Prog/timer.c, [3328](#)
- Demo/ARMCM4\_STM32F4\_Olimex\_STM32↔  
P405\_IAR/Prog/timer.c, [3329](#)
- Demo/ARMCM4\_STM32F4\_Olimex\_STM32↔  
P405\_Keil/Prog/timer.c, [3331](#)
- Demo/ARMCM4\_STM32L4\_Nucleo\_L476RG\_G↔  
CC/Prog/timer.c, [3334](#)
- Demo/ARMCM4\_STM32L4\_Nucleo\_L476RG\_I↔  
AR/Prog/timer.c, [3335](#)
- Demo/ARMCM4\_STM32L4\_Nucleo\_L476RG\_↔  
Keil/Prog/timer.c, [3337](#)
- Demo/ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_↔  
GCC/Prog/timer.c, [3338](#)
- Demo/ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_I↔  
AR/Prog/timer.c, [3341](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F746ZG\_G↔  
CC/Prog/timer.c, [3344](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F746ZG\_I↔  
AR/Prog/timer.c, [3346](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F746ZG\_↔  
Keil/Prog/timer.c, [3347](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F767ZI\_G↔  
CC/Prog/timer.c, [3350](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F767ZI\_IA↔  
R/Prog/timer.c, [3351](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F767ZI\_↔  
Keil/Prog/timer.c, [3353](#)
- Demo/ARMCM7\_STM32H7\_Nucleo\_H743ZI\_G↔  
CC/Prog/timer.c, [3356](#)
- Demo/ARMCM7\_STM32H7\_Nucleo\_H743ZI\_IA↔  
R/Prog/timer.c, [3357](#)
- Demo/ARMCM7\_STM32H7\_Nucleo\_H743ZI\_↔  
Keil/Prog/timer.c, [3359](#)
- Source/ARMCM0\_STM32F0/timer.c, [3369](#)
- Source/ARMCM0\_STM32G0/timer.c, [3372](#)
- Source/ARMCM33\_STM32L5/timer.c, [3378](#)
- Source/ARMCM3\_STM32F1/timer.c, [3386](#)
- Source/ARMCM3\_STM32F2/timer.c, [3389](#)
- Source/ARMCM4\_STM32F3/timer.c, [3394](#)
- Source/ARMCM4\_STM32F4/timer.c, [3397](#)
- Source/ARMCM4\_STM32L4/timer.c, [3400](#)
- Source/ARMCM7\_STM32F7/timer.c, [3408](#)
- Source/ARMCM7\_STM32H7/timer.c, [3411](#)
- sysclock\_max  
tFlashPrescalerSysclockMapping, [371](#)
- sysclock\_min  
tFlashPrescalerSysclockMapping, [371](#)
- SystemClock\_Config
  - ARMCM0\_STM32F0\_Discovery\_STM32F051\_↔  
GCC/Boot/main.c, [2610](#)
  - ARMCM0\_STM32F0\_Discovery\_STM32F051\_↔  
GCC/Prog/main.c, [2612](#)
  - ARMCM0\_STM32F0\_Discovery\_STM32F051\_I↔  
AR/Boot/main.c, [2615](#)
  - ARMCM0\_STM32F0\_Discovery\_STM32F051\_I↔  
AR/Prog/main.c, [2617](#)
  - ARMCM0\_STM32F0\_Discovery\_STM32F051\_↔  
Keil/Boot/main.c, [2619](#)
  - ARMCM0\_STM32F0\_Discovery\_STM32F051\_↔  
Keil/Prog/main.c, [2622](#)
  - ARMCM0\_STM32F0\_Nucleo\_F091RC\_GCC/↔  
Boot/main.c, [2624](#)
  - ARMCM0\_STM32F0\_Nucleo\_F091RC\_GCC/↔  
Prog/main.c, [2626](#)
  - ARMCM0\_STM32F0\_Nucleo\_F091RC\_IAR/↔  
Boot/main.c, [2629](#)
  - ARMCM0\_STM32F0\_Nucleo\_F091RC\_IAR/↔  
Prog/main.c, [2631](#)
  - ARMCM0\_STM32F0\_Nucleo\_F091RC\_Keil/↔  
Boot/main.c, [2633](#)
  - ARMCM0\_STM32F0\_Nucleo\_F091RC\_Keil/↔  
Prog/main.c, [2636](#)
  - ARMCM0\_STM32G0\_Nucleo\_G071RB\_GCC/↔  
Boot/main.c, [2638](#)
  - ARMCM0\_STM32G0\_Nucleo\_G071RB\_GCC/↔  
Prog/main.c, [2640](#)
  - ARMCM0\_STM32G0\_Nucleo\_G071RB\_IAR/↔  
Boot/main.c, [2643](#)
  - ARMCM0\_STM32G0\_Nucleo\_G071RB\_IAR/↔  
Prog/main.c, [2645](#)
  - ARMCM0\_STM32G0\_Nucleo\_G071RB\_Keil/↔  
Boot/main.c, [2648](#)
  - ARMCM0\_STM32G0\_Nucleo\_G071RB\_Keil/↔  
Prog/main.c, [2650](#)
  - ARMCM33\_STM32L5\_Nucleo\_L552ZE\_GCC/↔  
Boot/main.c, [2659](#)
  - ARMCM33\_STM32L5\_Nucleo\_L552ZE\_GCC/↔

- Prog/main.c, [2661](#)  
ARMCM33\_STM32L5\_Nucleo\_L552ZE\_IAR/↔  
Boot/main.c, [2664](#)  
ARMCM33\_STM32L5\_Nucleo\_L552ZE\_IAR/↔  
Prog/main.c, [2666](#)  
ARMCM33\_STM32L5\_Nucleo\_L552ZE\_Keil/↔  
Boot/main.c, [2669](#)  
ARMCM33\_STM32L5\_Nucleo\_L552ZE\_Keil/↔  
Prog/main.c, [2671](#)  
ARMCM3\_STM32F1\_Nucleo\_F103RB\_GCC/↔  
Boot/main.c, [2689](#)  
ARMCM3\_STM32F1\_Nucleo\_F103RB\_GCC/↔  
Prog/main.c, [2691](#)  
ARMCM3\_STM32F1\_Nucleo\_F103RB\_IAR/↔  
Boot/main.c, [2694](#)  
ARMCM3\_STM32F1\_Nucleo\_F103RB\_IAR/↔  
Prog/main.c, [2696](#)  
ARMCM3\_STM32F1\_Nucleo\_F103RB\_Keil/↔  
Boot/main.c, [2699](#)  
ARMCM3\_STM32F1\_Nucleo\_F103RB\_Keil/↔  
Prog/main.c, [2701](#)  
ARMCM3\_STM32F1\_Olimex\_STM32H103\_GC↔  
C/Boot/main.c, [2704](#)  
ARMCM3\_STM32F1\_Olimex\_STM32H103\_GC↔  
C/Prog/main.c, [2706](#)  
ARMCM3\_STM32F1\_Olimex\_STM32H103\_IA↔  
R/Boot/main.c, [2709](#)  
ARMCM3\_STM32F1\_Olimex\_STM32H103\_IA↔  
R/Prog/main.c, [2711](#)  
ARMCM3\_STM32F1\_Olimex\_STM32H103\_↔  
Keil/Boot/main.c, [2714](#)  
ARMCM3\_STM32F1\_Olimex\_STM32H103\_↔  
Keil/Prog/main.c, [2716](#)  
ARMCM3\_STM32F1\_Olimex\_STM32P103\_GC↔  
C/Boot/main.c, [2719](#)  
ARMCM3\_STM32F1\_Olimex\_STM32P103\_GC↔  
C/Prog/main.c, [2721](#)  
ARMCM3\_STM32F1\_Olimex\_STM32P103\_IA↔  
R/Boot/main.c, [2724](#)  
ARMCM3\_STM32F1\_Olimex\_STM32P103\_IA↔  
R/Prog/main.c, [2726](#)  
ARMCM3\_STM32F1\_Olimex\_STM32P103\_Keil/↔  
Boot/main.c, [2729](#)  
ARMCM3\_STM32F1\_Olimex\_STM32P103\_Keil/↔  
Prog/main.c, [2731](#)  
ARMCM3\_STM32F1\_Olimexino\_STM32\_GCC/↔  
Boot/main.c, [2734](#)  
ARMCM3\_STM32F1\_Olimexino\_STM32\_GCC/↔  
Prog/main.c, [2736](#)  
ARMCM3\_STM32F1\_Olimexino\_STM32\_IAR/↔  
Boot/main.c, [2739](#)  
ARMCM3\_STM32F1\_Olimexino\_STM32\_IAR/↔  
Prog/main.c, [2741](#)  
ARMCM3\_STM32F1\_Olimexino\_STM32\_Keil/↔  
Boot/main.c, [2744](#)  
ARMCM3\_STM32F1\_Olimexino\_STM32\_Keil/↔  
Prog/main.c, [2746](#)  
ARMCM3\_STM32F2\_Olimex\_STM32P207\_GC↔  
C/Boot/main.c, [2749](#)  
ARMCM3\_STM32F2\_Olimex\_STM32P207\_GC↔  
C/Prog/main.c, [2751](#)  
ARMCM3\_STM32F2\_Olimex\_STM32P207\_IA↔  
R/Boot/main.c, [2754](#)  
ARMCM3\_STM32F2\_Olimex\_STM32P207\_IA↔  
R/Prog/main.c, [2756](#)  
ARMCM3\_STM32F2\_Olimex\_STM32P207\_Keil/↔  
Boot/main.c, [2759](#)  
ARMCM3\_STM32F2\_Olimex\_STM32P207\_Keil/↔  
Prog/main.c, [2761](#)  
ARMCM4\_STM32F3\_Discovery\_F303VC\_GCC/↔  
Boot/main.c, [2771](#)  
ARMCM4\_STM32F3\_Discovery\_F303VC\_GCC/↔  
Prog/main.c, [2774](#)  
ARMCM4\_STM32F3\_Discovery\_F303VC\_IAR/↔  
Boot/main.c, [2776](#)  
ARMCM4\_STM32F3\_Discovery\_F303VC\_IAR/↔  
Prog/main.c, [2779](#)  
ARMCM4\_STM32F3\_Discovery\_F303VC\_Keil/↔  
Boot/main.c, [2782](#)  
ARMCM4\_STM32F3\_Discovery\_F303VC\_Keil/↔  
Prog/main.c, [2784](#)  
ARMCM4\_STM32F3\_Nucleo\_F303K8\_GCC/↔  
Boot/main.c, [2787](#)  
ARMCM4\_STM32F3\_Nucleo\_F303K8\_GCC/↔  
Prog/main.c, [2789](#)  
ARMCM4\_STM32F3\_Nucleo\_F303K8\_IAR/↔  
Boot/main.c, [2792](#)  
ARMCM4\_STM32F3\_Nucleo\_F303K8\_IAR/↔  
Prog/main.c, [2794](#)  
ARMCM4\_STM32F3\_Nucleo\_F303K8\_Keil/↔  
Boot/main.c, [2797](#)  
ARMCM4\_STM32F3\_Nucleo\_F303K8\_Keil/↔  
Prog/main.c, [2799](#)  
ARMCM4\_STM32F4\_Nucleo\_F429ZI\_GCC/↔  
Boot/main.c, [2802](#)  
ARMCM4\_STM32F4\_Nucleo\_F429ZI\_GCC/↔  
Prog/main.c, [2804](#)  
ARMCM4\_STM32F4\_Nucleo\_F429ZI\_IAR/↔  
Boot/main.c, [2807](#)  
ARMCM4\_STM32F4\_Nucleo\_F429ZI\_IAR/↔  
Prog/main.c, [2809](#)  
ARMCM4\_STM32F4\_Nucleo\_F429ZI\_Keil/↔  
Boot/main.c, [2812](#)  
ARMCM4\_STM32F4\_Nucleo\_F429ZI\_Keil/↔  
Prog/main.c, [2814](#)  
ARMCM4\_STM32F4\_Olimex\_STM32P405\_GC↔  
C/Boot/main.c, [2817](#)  
ARMCM4\_STM32F4\_Olimex\_STM32P405\_GC↔  
C/Prog/main.c, [2819](#)  
ARMCM4\_STM32F4\_Olimex\_STM32P405\_IA↔  
R/Boot/main.c, [2822](#)  
ARMCM4\_STM32F4\_Olimex\_STM32P405\_IA↔  
R/Prog/main.c, [2824](#)  
ARMCM4\_STM32F4\_Olimex\_STM32P405\_Keil/↔  
Boot/main.c, [2827](#)  
ARMCM4\_STM32F4\_Olimex\_STM32P405\_Keil/↔

- Prog/main.c, [2829](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_GCC/↔  
Boot/main.c, [2832](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_GCC/↔  
Prog/main.c, [2834](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_IAR/↔  
Boot/main.c, [2837](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_IAR/↔  
Prog/main.c, [2839](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_Keil/↔  
Boot/main.c, [2842](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_Keil/↔  
Prog/main.c, [2844](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_GCC/↔  
Boot/main.c, [2855](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_GCC/↔  
Prog/main.c, [2857](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_IAR/↔  
Boot/main.c, [2860](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_IAR/↔  
Prog/main.c, [2862](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_Keil/↔  
Boot/main.c, [2865](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_Keil/↔  
Prog/main.c, [2867](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_GCC/↔  
Boot/main.c, [2870](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_GCC/↔  
Prog/main.c, [2872](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_IAR/↔  
Boot/main.c, [2875](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_IAR/↔  
Prog/main.c, [2877](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_Keil/↔  
Boot/main.c, [2880](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_Keil/↔  
Prog/main.c, [2882](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_GCC/↔  
Boot/main.c, [2885](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_GCC/↔  
Prog/main.c, [2887](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_IAR/↔  
Boot/main.c, [2890](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_IAR/↔  
Prog/main.c, [2892](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_Keil/↔  
Boot/main.c, [2895](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_Keil/↔  
Prog/main.c, [2897](#)
- SystemClockConfig
  - ARMCM4\_S32K14\_S32K144EVB\_GCC/Boot/main.↔  
c, [2763](#)
  - ARMCM4\_S32K14\_S32K144EVB\_GCC/Prog/main.↔  
c, [2765](#)
  - ARMCM4\_S32K14\_S32K144EVB\_IAR/Boot/main.↔  
c, [2767](#)
  - ARMCM4\_S32K14\_S32K144EVB\_IAR/Prog/main.↔  
c, [2768](#)
- SystemClockInit
  - HCS12\_DevKit\_S12G128\_CodeWarrior/Boot/main.↔  
c, [2899](#)
  - HCS12\_DevKit\_S12G128\_CodeWarrior/Prog/main.↔  
c, [2901](#)
- tCanBusTiming, [355](#)
  - phaseSeg1, [355](#)
  - phaseSeg2, [356](#)
  - propSeg, [356](#)
  - timeQuanta, [356](#)
  - tseg1, [357](#)
  - tseg2, [357](#)
- tCanRegs, [358](#)
  - cbtr0, [359](#)
  - cbtr1, [359](#)
  - cctl0, [359](#)
  - cctl1, [359](#)
  - cidac, [359](#)
  - cidar0, [359](#)
  - cidar1, [359](#)
  - cidar2, [360](#)
  - cidar3, [360](#)
  - cidar4, [360](#)
  - cidar5, [360](#)
  - cidar6, [360](#)
  - cidar7, [360](#)
  - cidmr0, [360](#)
  - cidmr1, [360](#)
  - cidmr2, [361](#)
  - cidmr3, [361](#)
  - cidmr4, [361](#)
  - cidmr5, [361](#)
  - cidmr6, [361](#)
  - cidmr7, [361](#)
  - crflg, [361](#)
  - crier, [361](#)
  - crxerr, [362](#)
  - ctaak, [362](#)
  - ctarq, [362](#)
  - ctbsel, [362](#)
  - ctflg, [362](#)
  - ctier, [362](#)
  - ctxerr, [362](#)
  - dummy1, [362](#)
  - rxSlot, [363](#)
  - txSlot, [363](#)
- tCanRxMsgSlot, [363](#)
  - dlr, [363](#)
  - dsr, [364](#)
  - dummy, [364](#)
  - idr, [364](#)
  - tstamp, [364](#)
- tCanTxMsgSlot, [364](#)
  - dlr, [365](#)
  - dsr, [365](#)
  - idr, [365](#)
  - tbpr, [365](#)
  - tstamp, [365](#)



- tComInterfaceld
  - com.h, [1125](#)
- tFatFsObjects, [365](#)
  - file, [366](#)
  - fs, [366](#)
- tFifoCtrl, [366](#)
  - endptr, [367](#)
  - entries, [367](#)
  - fifoctrlptr, [367](#)
  - handle, [367](#)
  - length, [367](#)
  - readptr, [368](#)
  - startptr, [368](#)
  - writeptr, [368](#)
- tFifoPipe, [368](#)
  - data, [369](#)
  - handle, [369](#)
- tFileEraseInfo, [369](#)
  - start\_address, [370](#)
  - total\_size, [370](#)
- tFirmwareUpdateState
  - file.c, [1256](#)
- tFlashBlockInfo, [370](#)
- tFlashPrescalerSysclockMapping, [371](#)
  - prescaler, [371](#)
  - sysclock\_max, [371](#)
  - sysclock\_min, [371](#)
- tFlashRegs, [372](#)
  - dfprot, [372](#)
  - fccob, [372](#)
  - fccobix, [372](#)
  - fcldiv, [373](#)
  - fcmd, [373](#)
  - fcnfg, [373](#)
  - fercnfg, [373](#)
  - ferstat, [373](#)
  - fopt, [373](#)
  - fprot, [373](#)
  - frsv0, [374](#)
  - frsv1, [374](#)
  - frsv2, [374](#)
  - frsv3, [374](#)
  - frsv4, [374](#)
  - frsv5, [374](#)
  - frsv6, [374](#)
  - frsv7, [375](#)
  - fsec, [375](#)
  - fstat, [375](#)
  - ftstmod, [375](#)
- tFlashSector, [375](#)
  - bank\_num, [376](#)
  - sector\_num, [376](#)
  - sector\_size, [376](#)
  - sector\_start, [376](#)
- tlSrFunc, [377](#)
  - func, [377](#)
  - ptr, [377](#), [378](#)
- tSharedParamsBuffer, [378](#)
- tSrecLineParseObject, [378](#)
  - address, [379](#)
  - data, [379](#)
  - line, [379](#)
- tSrecLineType
  - file.h, [1264](#)
- tSysTickRegs, [380](#)
  - CTRL, [380](#)
  - LOAD, [380](#)
  - VAL, [380](#)
- tTimerRegs, [380](#)
  - cforc, [381](#)
  - oc7d, [381](#)
  - oc7m, [381](#)
  - tc, [381](#)
  - tcnt, [382](#)
  - tctl1, [382](#)
  - tctl2, [382](#)
  - tctl3, [382](#)
  - tctl4, [382](#)
  - tflg1, [382](#)
  - tflg2, [382](#)
  - tie, [382](#)
  - tios, [383](#)
  - tscr1, [383](#)
  - tscr2, [383](#)
  - ttov, [383](#)
- tXcpInfo, [383](#)
  - connected, [384](#)
  - ctoData, [384](#)
  - ctoLen, [384](#)
  - ctoPending, [384](#)
  - mta, [385](#)
  - protection, [385](#)
  - s\_n\_k\_resource, [385](#)
- Target ARMCM0 S32K11, [333](#)
- Target ARMCM0 STM32F0, [334](#)
- Target ARMCM0 STM32G0, [335](#)
- Target ARMCM0 XMC1, [336](#)
- Target ARMCM3 EFM32, [338](#)
- Target ARMCM3 LM3S, [339](#)
- Target ARMCM3 STM32F1, [340](#)
- Target ARMCM3 STM32F2, [341](#)
- Target ARMCM33 STM32L5, [337](#)
- Target ARMCM4 S32K14, [342](#)
- Target ARMCM4 STM32F3, [343](#)
- Target ARMCM4 STM32F4, [344](#)
- Target ARMCM4 STM32L4, [345](#)
- Target ARMCM4 TM4C, [346](#)
- Target ARMCM4 XMC4, [347](#)
- Target ARMCM7 STM32F7, [348](#)
- Target ARMCM7 STM32H7, [349](#)
- Target HCS12, [352](#)
- Target Port Template, [329](#)
- tbpr
  - tCanTxMsgSlot, [365](#)
- tc
  - tTimerRegs, [381](#)

- tcnt
  - tTimerRegs, [382](#)
- tctl1
  - tTimerRegs, [382](#)
- tctl2
  - tTimerRegs, [382](#)
- tctl3
  - tTimerRegs, [382](#)
- tctl4
  - tTimerRegs, [382](#)
- Template for demo programs, [62](#)
- tflg1
  - tTimerRegs, [382](#)
- tflg2
  - tTimerRegs, [382](#)
- tie
  - tTimerRegs, [382](#)
- time.c, [3219](#), [3221](#), [3223](#), [3226](#), [3228](#)
- time.h, [3230](#), [3232](#), [3235](#), [3237](#), [3239](#)
- timeQuanta
  - tCanBusTiming, [356](#)
- Timer driver of a port, [330](#)
- timer.c, [3242](#), [3244–3246](#), [3248](#), [3249](#), [3251](#), [3252](#), [3254–3256](#), [3258](#), [3259](#), [3261](#), [3263](#), [3265](#), [3267](#), [3268](#), [3270](#), [3271](#), [3273](#), [3276–3278](#), [3280](#), [3282](#), [3283](#), [3285](#), [3286](#), [3288](#), [3289](#), [3291](#), [3292](#), [3294–3296](#), [3298](#), [3299](#), [3301](#), [3302](#), [3304](#), [3305](#), [3307](#), [3308](#), [3310](#), [3311](#), [3313](#), [3315–3317](#), [3319](#), [3320](#), [3322](#), [3323](#), [3325–3327](#), [3329](#), [3330](#), [3332](#), [3333](#), [3335](#), [3336](#), [3338](#), [3340](#), [3342](#), [3344](#), [3345](#), [3347–3349](#), [3351](#), [3352](#), [3354](#), [3355](#), [3357](#), [3358](#), [3360](#), [3362](#), [3364](#), [3366](#), [3368](#), [3371](#), [3374](#), [3377](#), [3380](#), [3382](#), [3385](#), [3388](#), [3391](#), [3393](#), [3396](#), [3399](#), [3402](#), [3404](#), [3407](#), [3410](#), [3413](#)
- timer.h, [3415](#), [3417–3419](#), [3421–3423](#), [3425–3427](#), [3429–3431](#), [3433](#), [3435](#), [3437](#), [3438](#), [3440–3442](#), [3445](#), [3447](#), [3448](#), [3450–3452](#), [3454–3456](#), [3458–3460](#), [3462–3464](#), [3466–3468](#), [3470–3472](#), [3474–3476](#), [3478–3480](#), [3482–3484](#), [3486–3488](#), [3490–3492](#), [3494–3496](#), [3498–3500](#), [3502](#), [3503](#), [3505](#), [3507](#), [3508](#), [3510–3512](#), [3514–3516](#), [3518–3520](#), [3522](#), [3523](#), [3525](#), [3528](#)
- TimerDeinit
  - ARMCM3\_LM3S\_EK\_LM3S6965\_GCC/Prog/time.↔  
c, [3220](#)
  - ARMCM3\_LM3S\_EK\_LM3S6965\_GCC/Prog/time.↔  
h, [3231](#)
  - ARMCM3\_LM3S\_EK\_LM3S6965\_IAR/Prog/time.↔  
c, [3222](#)
  - ARMCM3\_LM3S\_EK\_LM3S6965\_IAR/Prog/time.↔  
h, [3233](#)
  - ARMCM3\_LM3S\_EK\_LM3S8962\_GCC/Prog/time.↔  
c, [3224](#)
  - ARMCM3\_LM3S\_EK\_LM3S8962\_GCC/Prog/time.↔  
h, [3235](#)
  - ARMCM3\_LM3S\_EK\_LM3S8962\_IAR/Prog/time.↔  
c, [3226](#)
  - ARMCM3\_LM3S\_EK\_LM3S8962\_IAR/Prog/time.↔  
h, [3238](#)
  - ARMCM4\_TM4C\_DK\_TM4C123G\_IAR/Prog/time.↔  
c, [3229](#)
  - ARMCM4\_TM4C\_DK\_TM4C123G\_IAR/Prog/time.↔  
h, [3240](#)
  - Demo/ARMCM0\_XMC1\_XMC1400\_Boot\_Kit\_G↔  
CC/Prog/timer.c, [3262](#)
  - Demo/ARMCM0\_XMC1\_XMC1400\_Boot\_Kit\_G↔  
CC/Prog/timer.h, [3433](#)
  - Demo/ARMCM0\_XMC1\_XMC1400\_Boot\_Kit\_IA↔  
R/Prog/timer.c, [3264](#)
  - Demo/ARMCM0\_XMC1\_XMC1400\_Boot\_Kit\_IA↔  
R/Prog/timer.h, [3436](#)
  - Demo/ARMCM3\_EFM32\_Olimex\_EM32G880↔  
F128STK\_GCC/Prog/timer.c, [3272](#)
  - Demo/ARMCM3\_EFM32\_Olimex\_EM32G880↔  
F128STK\_GCC/Prog/timer.h, [3443](#)
  - Demo/ARMCM3\_EFM32\_Olimex\_EM32G880↔  
F128STK\_IAR/Prog/timer.c, [3274](#)
  - Demo/ARMCM3\_EFM32\_Olimex\_EM32G880↔  
F128STK\_IAR/Prog/timer.h, [3445](#)
  - Demo/ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_↔  
GCC/Prog/timer.c, [3339](#)
  - Demo/ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_↔  
GCC/Prog/timer.h, [3504](#)
  - Demo/ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_I↔  
AR/Prog/timer.c, [3341](#)
  - Demo/ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_I↔  
AR/Prog/timer.h, [3506](#)
  - Demo/HCS12\_DevKit\_S12G128\_CodeWarrior/↔  
Prog/timer.c, [3361](#)
  - Demo/HCS12\_DevKit\_S12G128\_CodeWarrior/↔  
Prog/timer.h, [3524](#)
  - Demo/HCS12\_Evbplus\_Dragon12p\_Code↔  
Warrior/Prog/timer.c, [3363](#)
  - Demo/HCS12\_Evbplus\_Dragon12p\_Code↔  
Warrior/Prog/timer.h, [3526](#)
- TimerGet
  - ARMCM3\_LM3S\_EK\_LM3S6965\_GCC/Prog/time.↔  
c, [3220](#)
  - ARMCM3\_LM3S\_EK\_LM3S6965\_GCC/Prog/time.↔  
h, [3231](#)
  - ARMCM3\_LM3S\_EK\_LM3S6965\_IAR/Prog/time.↔  
c, [3222](#)
  - ARMCM3\_LM3S\_EK\_LM3S6965\_IAR/Prog/time.↔  
h, [3233](#)
  - ARMCM3\_LM3S\_EK\_LM3S8962\_GCC/Prog/time.↔  
c, [3224](#)
  - ARMCM3\_LM3S\_EK\_LM3S8962\_GCC/Prog/time.↔  
h, [3235](#)
  - ARMCM3\_LM3S\_EK\_LM3S8962\_IAR/Prog/time.↔  
c, [3227](#)
  - ARMCM3\_LM3S\_EK\_LM3S8962\_IAR/Prog/time.↔  
h, [3238](#)
  - ARMCM4\_TM4C\_DK\_TM4C123G\_IAR/Prog/time.↔  
c, [3229](#)

- ARMCM4\_TM4C\_DK\_TM4C123G\_IAR/Prog/timer.c, [3240](#)
- Demo/\_template/Prog/timer.c, [3242](#)
- Demo/\_template/Prog/timer.h, [3416](#)
- Demo/ARMCM0\_STM32F0\_Discovery\_STM32F051\_CubeIDE/Prog/App/timer.c, [3244](#)
- Demo/ARMCM0\_STM32F0\_Discovery\_STM32F051\_CubeIDE/Prog/App/timer.h, [3417](#)
- Demo/ARMCM0\_STM32F0\_Discovery\_STM32F051\_GCC/Prog/timer.c, [3246](#)
- Demo/ARMCM0\_STM32F0\_Discovery\_STM32F051\_GCC/Prog/timer.h, [3419](#)
- Demo/ARMCM0\_STM32F0\_Discovery\_STM32F051\_IAR/Prog/timer.c, [3247](#)
- Demo/ARMCM0\_STM32F0\_Discovery\_STM32F051\_IAR/Prog/timer.h, [3420](#)
- Demo/ARMCM0\_STM32F0\_Discovery\_STM32F051\_Keil/Prog/timer.c, [3249](#)
- Demo/ARMCM0\_STM32F0\_Discovery\_STM32F051\_Keil/Prog/timer.h, [3421](#)
- Demo/ARMCM0\_STM32F0\_Nucleo\_F091RC\_CubeIDE/Prog/App/timer.c, [3250](#)
- Demo/ARMCM0\_STM32F0\_Nucleo\_F091RC\_CubeIDE/Prog/App/timer.h, [3423](#)
- Demo/ARMCM0\_STM32F0\_Nucleo\_F091RC\_GCC/Prog/timer.c, [3251](#)
- Demo/ARMCM0\_STM32F0\_Nucleo\_F091RC\_GCC/Prog/timer.h, [3424](#)
- Demo/ARMCM0\_STM32F0\_Nucleo\_F091RC\_IAR/Prog/timer.c, [3253](#)
- Demo/ARMCM0\_STM32F0\_Nucleo\_F091RC\_IAR/Prog/timer.h, [3425](#)
- Demo/ARMCM0\_STM32F0\_Nucleo\_F091RC\_Keil/Prog/timer.c, [3254](#)
- Demo/ARMCM0\_STM32F0\_Nucleo\_F091RC\_Keil/Prog/timer.h, [3427](#)
- Demo/ARMCM0\_STM32G0\_Nucleo\_G071RB\_CubeIDE/Prog/App/timer.c, [3256](#)
- Demo/ARMCM0\_STM32G0\_Nucleo\_G071RB\_CubeIDE/Prog/App/timer.h, [3428](#)
- Demo/ARMCM0\_STM32G0\_Nucleo\_G071RB\_GCC/Prog/timer.c, [3257](#)
- Demo/ARMCM0\_STM32G0\_Nucleo\_G071RB\_GCC/Prog/timer.h, [3429](#)
- Demo/ARMCM0\_STM32G0\_Nucleo\_G071RB\_IAR/Prog/timer.c, [3259](#)
- Demo/ARMCM0\_STM32G0\_Nucleo\_G071RB\_IAR/Prog/timer.h, [3431](#)
- Demo/ARMCM0\_STM32G0\_Nucleo\_G071RB\_Keil/Prog/timer.c, [3260](#)
- Demo/ARMCM0\_STM32G0\_Nucleo\_G071RB\_Keil/Prog/timer.h, [3432](#)
- Demo/ARMCM0\_XMC1\_XMC1400\_Boot\_Kit\_GCC/Prog/timer.c, [3262](#)
- Demo/ARMCM0\_XMC1\_XMC1400\_Boot\_Kit\_GCC/Prog/timer.h, [3434](#)
- Demo/ARMCM0\_XMC1\_XMC1400\_Boot\_Kit\_IAR/Prog/timer.c, [3264](#)
- Demo/ARMCM0\_XMC1\_XMC1400\_Boot\_Kit\_IAR/Prog/timer.h, [3436](#)
- Demo/ARMCM33\_STM32L5\_Nucleo\_L552ZE\_CubeIDE/Prog/App/timer.c, [3266](#)
- Demo/ARMCM33\_STM32L5\_Nucleo\_L552ZE\_CubeIDE/Prog/App/timer.h, [3437](#)
- Demo/ARMCM33\_STM32L5\_Nucleo\_L552ZE\_GCC/Prog/timer.c, [3268](#)
- Demo/ARMCM33\_STM32L5\_Nucleo\_L552ZE\_GCC/Prog/timer.h, [3439](#)
- Demo/ARMCM33\_STM32L5\_Nucleo\_L552ZE\_IAR/Prog/timer.c, [3269](#)
- Demo/ARMCM33\_STM32L5\_Nucleo\_L552ZE\_IAR/Prog/timer.h, [3440](#)
- Demo/ARMCM33\_STM32L5\_Nucleo\_L552ZE\_Keil/Prog/timer.c, [3271](#)
- Demo/ARMCM33\_STM32L5\_Nucleo\_L552ZE\_Keil/Prog/timer.h, [3442](#)
- Demo/ARMCM3\_EFM32\_Olimex\_EM32G880\_F128STK\_GCC/Prog/timer.c, [3272](#)
- Demo/ARMCM3\_EFM32\_Olimex\_EM32G880\_F128STK\_GCC/Prog/timer.h, [3443](#)
- Demo/ARMCM3\_EFM32\_Olimex\_EM32G880\_F128STK\_IAR/Prog/timer.c, [3274](#)
- Demo/ARMCM3\_EFM32\_Olimex\_EM32G880\_F128STK\_IAR/Prog/timer.h, [3446](#)
- Demo/ARMCM3\_STM32F1\_Nucleo\_F103RB\_CubeIDE/Prog/App/timer.c, [3276](#)
- Demo/ARMCM3\_STM32F1\_Nucleo\_F103RB\_CubeIDE/Prog/App/timer.h, [3447](#)
- Demo/ARMCM3\_STM32F1\_Nucleo\_F103RB\_GCC/Prog/timer.c, [3278](#)
- Demo/ARMCM3\_STM32F1\_Nucleo\_F103RB\_GCC/Prog/timer.h, [3449](#)
- Demo/ARMCM3\_STM32F1\_Nucleo\_F103RB\_IAR/Prog/timer.c, [3279](#)
- Demo/ARMCM3\_STM32F1\_Nucleo\_F103RB\_IAR/Prog/timer.h, [3450](#)
- Demo/ARMCM3\_STM32F1\_Nucleo\_F103RB\_Keil/Prog/timer.c, [3281](#)
- Demo/ARMCM3\_STM32F1\_Nucleo\_F103RB\_Keil/Prog/timer.h, [3452](#)
- Demo/ARMCM3\_STM32F1\_Olimex\_STM32H103\_CubeIDE/Prog/App/timer.c, [3282](#)
- Demo/ARMCM3\_STM32F1\_Olimex\_STM32H103\_CubeIDE/Prog/App/timer.h, [3453](#)
- Demo/ARMCM3\_STM32F1\_Olimex\_STM32H103\_GCC/Prog/timer.c, [3284](#)
- Demo/ARMCM3\_STM32F1\_Olimex\_STM32H103\_GCC/Prog/timer.h, [3454](#)
- Demo/ARMCM3\_STM32F1\_Olimex\_STM32H103\_IAR/Prog/timer.c, [3285](#)
- Demo/ARMCM3\_STM32F1\_Olimex\_STM32H103\_IAR/Prog/timer.h, [3456](#)
- Demo/ARMCM3\_STM32F1\_Olimex\_STM32H103\_Keil/Prog/timer.c, [3287](#)
- Demo/ARMCM3\_STM32F1\_Olimex\_STM32H103\_Keil/Prog/timer.h, [3457](#)

- Demo/ARMCM3\_STM32F1\_Olimex\_STM32↔  
P103\_CubeIDE/Prog/App/timer.c, [3288](#)
- Demo/ARMCM3\_STM32F1\_Olimex\_STM32↔  
P103\_CubeIDE/Prog/App/timer.h, [3458](#)
- Demo/ARMCM3\_STM32F1\_Olimex\_STM32↔  
P103\_GCC/Prog/timer.c, [3290](#)
- Demo/ARMCM3\_STM32F1\_Olimex\_STM32↔  
P103\_GCC/Prog/timer.h, [3460](#)
- Demo/ARMCM3\_STM32F1\_Olimex\_STM32↔  
P103\_IAR/Prog/timer.c, [3291](#)
- Demo/ARMCM3\_STM32F1\_Olimex\_STM32↔  
P103\_IAR/Prog/timer.h, [3461](#)
- Demo/ARMCM3\_STM32F1\_Olimex\_STM32↔  
P103\_Keil/Prog/timer.c, [3293](#)
- Demo/ARMCM3\_STM32F1\_Olimex\_STM32↔  
P103\_Keil/Prog/timer.h, [3462](#)
- Demo/ARMCM3\_STM32F1\_Olimexino\_STM32↔  
\_CubeIDE/Prog/App/timer.c, [3294](#)
- Demo/ARMCM3\_STM32F1\_Olimexino\_STM32↔  
\_CubeIDE/Prog/App/timer.h, [3464](#)
- Demo/ARMCM3\_STM32F1\_Olimexino\_STM32↔  
\_GCC/Prog/timer.c, [3296](#)
- Demo/ARMCM3\_STM32F1\_Olimexino\_STM32↔  
\_GCC/Prog/timer.h, [3465](#)
- Demo/ARMCM3\_STM32F1\_Olimexino\_STM32↔  
\_IAR/Prog/timer.c, [3297](#)
- Demo/ARMCM3\_STM32F1\_Olimexino\_STM32↔  
\_IAR/Prog/timer.h, [3466](#)
- Demo/ARMCM3\_STM32F1\_Olimexino\_STM32↔  
\_Keil/Prog/timer.c, [3299](#)
- Demo/ARMCM3\_STM32F1\_Olimexino\_STM32↔  
\_Keil/Prog/timer.h, [3468](#)
- Demo/ARMCM3\_STM32F2\_Olimex\_STM32↔  
P207\_CubeIDE/Prog/App/timer.c, [3300](#)
- Demo/ARMCM3\_STM32F2\_Olimex\_STM32↔  
P207\_CubeIDE/Prog/App/timer.h, [3469](#)
- Demo/ARMCM3\_STM32F2\_Olimex\_STM32↔  
P207\_GCC/Prog/timer.c, [3301](#)
- Demo/ARMCM3\_STM32F2\_Olimex\_STM32↔  
P207\_GCC/Prog/timer.h, [3470](#)
- Demo/ARMCM3\_STM32F2\_Olimex\_STM32↔  
P207\_IAR/Prog/timer.c, [3303](#)
- Demo/ARMCM3\_STM32F2\_Olimex\_STM32↔  
P207\_IAR/Prog/timer.h, [3472](#)
- Demo/ARMCM3\_STM32F2\_Olimex\_STM32↔  
P207\_Keil/Prog/timer.c, [3304](#)
- Demo/ARMCM3\_STM32F2\_Olimex\_STM32↔  
P207\_Keil/Prog/timer.h, [3473](#)
- Demo/ARMCM4\_S32K14\_S32K144EVB\_GCC/↔  
Prog/timer.c, [3306](#)
- Demo/ARMCM4\_S32K14\_S32K144EVB\_GCC/↔  
Prog/timer.h, [3474](#)
- Demo/ARMCM4\_S32K14\_S32K144EVB\_IAR/↔  
Prog/timer.c, [3307](#)
- Demo/ARMCM4\_S32K14\_S32K144EVB\_IAR/↔  
Prog/timer.h, [3476](#)
- Demo/ARMCM4\_STM32F3\_Discovery\_F303VC↔  
\_CubeIDE/Prog/App/timer.c, [3309](#)
- Demo/ARMCM4\_STM32F3\_Discovery\_F303VC↔  
\_CubeIDE/Prog/App/timer.h, [3477](#)
- Demo/ARMCM4\_STM32F3\_Discovery\_F303VC↔  
\_GCC/Prog/timer.c, [3310](#)
- Demo/ARMCM4\_STM32F3\_Discovery\_F303VC↔  
\_GCC/Prog/timer.h, [3478](#)
- Demo/ARMCM4\_STM32F3\_Discovery\_F303VC↔  
\_IAR/Prog/timer.c, [3312](#)
- Demo/ARMCM4\_STM32F3\_Discovery\_F303VC↔  
\_IAR/Prog/timer.h, [3480](#)
- Demo/ARMCM4\_STM32F3\_Discovery\_F303VC↔  
\_Keil/Prog/timer.c, [3314](#)
- Demo/ARMCM4\_STM32F3\_Discovery\_F303VC↔  
\_Keil/Prog/timer.h, [3481](#)
- Demo/ARMCM4\_STM32F3\_Nucleo\_F303K8↔  
CubeIDE/Prog/App/timer.c, [3315](#)
- Demo/ARMCM4\_STM32F3\_Nucleo\_F303K8↔  
CubeIDE/Prog/App/timer.h, [3482](#)
- Demo/ARMCM4\_STM32F3\_Nucleo\_F303K8↔  
GCC/Prog/timer.c, [3317](#)
- Demo/ARMCM4\_STM32F3\_Nucleo\_F303K8↔  
GCC/Prog/timer.h, [3484](#)
- Demo/ARMCM4\_STM32F3\_Nucleo\_F303K8↔  
IAR/Prog/timer.c, [3318](#)
- Demo/ARMCM4\_STM32F3\_Nucleo\_F303K8↔  
IAR/Prog/timer.h, [3485](#)
- Demo/ARMCM4\_STM32F3\_Nucleo\_F303K8↔  
Keil/Prog/timer.c, [3320](#)
- Demo/ARMCM4\_STM32F3\_Nucleo\_F303K8↔  
Keil/Prog/timer.h, [3486](#)
- Demo/ARMCM4\_STM32F4\_Nucleo\_F429ZI↔  
CubeIDE/Prog/App/timer.c, [3321](#)
- Demo/ARMCM4\_STM32F4\_Nucleo\_F429ZI↔  
CubeIDE/Prog/App/timer.h, [3488](#)
- Demo/ARMCM4\_STM32F4\_Nucleo\_F429ZI↔  
GCC/Prog/timer.c, [3322](#)
- Demo/ARMCM4\_STM32F4\_Nucleo\_F429ZI↔  
GCC/Prog/timer.h, [3489](#)
- Demo/ARMCM4\_STM32F4\_Nucleo\_F429ZI↔  
IAR/Prog/timer.c, [3324](#)
- Demo/ARMCM4\_STM32F4\_Nucleo\_F429ZI↔  
IAR/Prog/timer.h, [3490](#)
- Demo/ARMCM4\_STM32F4\_Nucleo\_F429ZI↔  
Keil/Prog/timer.c, [3325](#)
- Demo/ARMCM4\_STM32F4\_Nucleo\_F429ZI↔  
Keil/Prog/timer.h, [3492](#)
- Demo/ARMCM4\_STM32F4\_Olimex\_STM32↔  
P405\_CubeIDE/Prog/App/timer.c, [3327](#)
- Demo/ARMCM4\_STM32F4\_Olimex\_STM32↔  
P405\_CubeIDE/Prog/App/timer.h, [3493](#)
- Demo/ARMCM4\_STM32F4\_Olimex\_STM32↔  
P405\_GCC/Prog/timer.c, [3328](#)
- Demo/ARMCM4\_STM32F4\_Olimex\_STM32↔  
P405\_GCC/Prog/timer.h, [3494](#)
- Demo/ARMCM4\_STM32F4\_Olimex\_STM32↔  
P405\_IAR/Prog/timer.c, [3330](#)
- Demo/ARMCM4\_STM32F4\_Olimex\_STM32↔  
P405\_IAR/Prog/timer.h, [3496](#)

- Demo/ARMCM4\_STM32F4\_Olimex\_STM32↔  
P405\_Keil/Prog/timer.c, [3331](#)
- Demo/ARMCM4\_STM32F4\_Olimex\_STM32↔  
P405\_Keil/Prog/timer.h, [3497](#)
- Demo/ARMCM4\_STM32L4\_Nucleo\_L476RG↔  
CubeIDE/Prog/App/timer.c, [3332](#)
- Demo/ARMCM4\_STM32L4\_Nucleo\_L476RG↔  
CubeIDE/Prog/App/timer.h, [3498](#)
- Demo/ARMCM4\_STM32L4\_Nucleo\_L476RG\_G↔  
CC/Prog/timer.c, [3334](#)
- Demo/ARMCM4\_STM32L4\_Nucleo\_L476RG\_G↔  
CC/Prog/timer.h, [3500](#)
- Demo/ARMCM4\_STM32L4\_Nucleo\_L476RG\_I↔  
AR/Prog/timer.c, [3335](#)
- Demo/ARMCM4\_STM32L4\_Nucleo\_L476RG\_I↔  
AR/Prog/timer.h, [3501](#)
- Demo/ARMCM4\_STM32L4\_Nucleo\_L476RG↔  
Keil/Prog/timer.c, [3337](#)
- Demo/ARMCM4\_STM32L4\_Nucleo\_L476RG↔  
Keil/Prog/timer.h, [3502](#)
- Demo/ARMCM4\_XMC4\_XMC4700\_Relax\_Kit↔  
GCC/Prog/timer.c, [3339](#)
- Demo/ARMCM4\_XMC4\_XMC4700\_Relax\_Kit↔  
GCC/Prog/timer.h, [3504](#)
- Demo/ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_I↔  
AR/Prog/timer.c, [3341](#)
- Demo/ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_I↔  
AR/Prog/timer.h, [3506](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F746ZG↔  
CubeIDE/Prog/App/timer.c, [3343](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F746ZG↔  
CubeIDE/Prog/App/timer.h, [3507](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F746ZG\_G↔  
CC/Prog/timer.c, [3344](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F746ZG\_G↔  
CC/Prog/timer.h, [3509](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F746ZG\_I↔  
AR/Prog/timer.c, [3346](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F746ZG\_I↔  
AR/Prog/timer.h, [3510](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F746ZG↔  
Keil/Prog/timer.c, [3347](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F746ZG↔  
Keil/Prog/timer.h, [3512](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F767ZI↔  
CubeIDE/Prog/App/timer.c, [3349](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F767ZI↔  
CubeIDE/Prog/App/timer.h, [3513](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F767ZI\_G↔  
CC/Prog/timer.c, [3350](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F767ZI\_G↔  
CC/Prog/timer.h, [3514](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F767ZI\_IA↔  
R/Prog/timer.c, [3352](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F767ZI\_IA↔  
R/Prog/timer.h, [3516](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F767ZI↔  
Keil/Prog/timer.c, [3353](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F767ZI↔  
Keil/Prog/timer.h, [3517](#)
- Demo/ARMCM7\_STM32H7\_Nucleo\_H743ZI↔  
CubeIDE/Prog/App/timer.c, [3354](#)
- Demo/ARMCM7\_STM32H7\_Nucleo\_H743ZI↔  
CubeIDE/Prog/App/timer.h, [3518](#)
- Demo/ARMCM7\_STM32H7\_Nucleo\_H743ZI\_G↔  
CC/Prog/timer.c, [3356](#)
- Demo/ARMCM7\_STM32H7\_Nucleo\_H743ZI\_G↔  
CC/Prog/timer.h, [3520](#)
- Demo/ARMCM7\_STM32H7\_Nucleo\_H743ZI\_IA↔  
R/Prog/timer.c, [3357](#)
- Demo/ARMCM7\_STM32H7\_Nucleo\_H743ZI\_IA↔  
R/Prog/timer.h, [3521](#)
- Demo/ARMCM7\_STM32H7\_Nucleo\_H743ZI↔  
Keil/Prog/timer.c, [3359](#)
- Demo/ARMCM7\_STM32H7\_Nucleo\_H743ZI↔  
Keil/Prog/timer.h, [3522](#)
- Demo/HCS12\_DevKit\_S12G128\_CodeWarrior↔  
Prog/timer.c, [3361](#)
- Demo/HCS12\_DevKit\_S12G128\_CodeWarrior↔  
Prog/timer.h, [3524](#)
- Demo/HCS12\_Evbplus\_Dragon12p\_Code↔  
Warrior/Prog/timer.c, [3363](#)
- Demo/HCS12\_Evbplus\_Dragon12p\_Code↔  
Warrior/Prog/timer.h, [3526](#)
- Source/\_template/timer.c, [3365](#)
- Source/ARMCM0\_S32K11/timer.c, [3367](#)
- Source/ARMCM0\_STM32F0/timer.c, [3370](#)
- Source/ARMCM0\_STM32G0/timer.c, [3373](#)
- Source/ARMCM0\_XMC1/timer.c, [3375](#)
- Source/ARMCM33\_STM32L5/timer.c, [3378](#)
- Source/ARMCM3\_EFM32/timer.c, [3381](#)
- Source/ARMCM3\_LM3S/timer.c, [3383](#)
- Source/ARMCM3\_STM32F1/timer.c, [3386](#)
- Source/ARMCM3\_STM32F2/timer.c, [3389](#)
- Source/ARMCM4\_S32K14/timer.c, [3391](#)
- Source/ARMCM4\_STM32F3/timer.c, [3394](#)
- Source/ARMCM4\_STM32F4/timer.c, [3397](#)
- Source/ARMCM4\_STM32L4/timer.c, [3400](#)
- Source/ARMCM4\_TM4C/timer.c, [3403](#)
- Source/ARMCM4\_XMC4/timer.c, [3405](#)
- Source/ARMCM7\_STM32F7/timer.c, [3408](#)
- Source/ARMCM7\_STM32H7/timer.c, [3411](#)
- Source/HCS12/timer.c, [3414](#)
- Source/timer.h, [3528](#)
- TimerISRHandler
  - ARMCM3\_LM3S\_EK\_LM3S6965\_GCC/Prog/time.↔  
c, [3220](#)
  - ARMCM3\_LM3S\_EK\_LM3S6965\_GCC/Prog/time.↔  
h, [3231](#)
  - ARMCM3\_LM3S\_EK\_LM3S6965\_IAR/Prog/time.↔  
c, [3223](#)
  - ARMCM3\_LM3S\_EK\_LM3S6965\_IAR/Prog/time.↔  
h, [3234](#)
  - ARMCM3\_LM3S\_EK\_LM3S8962\_GCC/Prog/time.↔  
c, [3225](#)
  - ARMCM3\_LM3S\_EK\_LM3S8962\_GCC/Prog/time.↔



- h, [3236](#)
- ARMCM3\_LM3S\_EK\_LM3S8962\_IAR/Prog/time.↔  
c, [3227](#)
- ARMCM3\_LM3S\_EK\_LM3S8962\_IAR/Prog/time.↔  
h, [3238](#)
- ARMCM4\_TM4C\_DK\_TM4C123G\_IAR/Prog/time.↔  
c, [3229](#)
- ARMCM4\_TM4C\_DK\_TM4C123G\_IAR/Prog/time.↔  
h, [3241](#)
- Demo/ARMCM3\_EFM32\_Olimex\_EM32G880↔  
F128STK\_GCC/Prog/timer.c, [3273](#)
- Demo/ARMCM3\_EFM32\_Olimex\_EM32G880↔  
F128STK\_GCC/Prog/timer.h, [3444](#)
- Demo/ARMCM3\_EFM32\_Olimex\_EM32G880↔  
F128STK\_IAR/Prog/timer.c, [3275](#)
- Demo/ARMCM3\_EFM32\_Olimex\_EM32G880↔  
F128STK\_IAR/Prog/timer.h, [3446](#)
- Demo/HCS12\_DevKit\_S12G128\_CodeWarrior/↔  
Prog/timer.c, [3361](#)
- Demo/HCS12\_DevKit\_S12G128\_CodeWarrior/↔  
Prog/timer.h, [3524](#)
- Demo/HCS12\_Evbplus\_Dragon12p\_Code↔  
Warrior/Prog/timer.c, [3363](#)
- Demo/HCS12\_Evbplus\_Dragon12p\_Code↔  
Warrior/Prog/timer.h, [3527](#)
- TimerInit
- ARMCM3\_LM3S\_EK\_LM3S6965\_GCC/Prog/time.↔  
c, [3220](#)
- ARMCM3\_LM3S\_EK\_LM3S6965\_GCC/Prog/time.↔  
h, [3231](#)
- ARMCM3\_LM3S\_EK\_LM3S6965\_IAR/Prog/time.↔  
c, [3222](#)
- ARMCM3\_LM3S\_EK\_LM3S6965\_IAR/Prog/time.↔  
h, [3233](#)
- ARMCM3\_LM3S\_EK\_LM3S8962\_GCC/Prog/time.↔  
c, [3225](#)
- ARMCM3\_LM3S\_EK\_LM3S8962\_GCC/Prog/time.↔  
h, [3236](#)
- ARMCM3\_LM3S\_EK\_LM3S8962\_IAR/Prog/time.↔  
c, [3227](#)
- ARMCM3\_LM3S\_EK\_LM3S8962\_IAR/Prog/time.↔  
h, [3238](#)
- ARMCM4\_TM4C\_DK\_TM4C123G\_IAR/Prog/time.↔  
c, [3229](#)
- ARMCM4\_TM4C\_DK\_TM4C123G\_IAR/Prog/time.↔  
h, [3240](#)
- Demo/\_template/Prog/timer.c, [3243](#)
- Demo/\_template/Prog/timer.h, [3416](#)
- Demo/ARMCM0\_STM32F0\_Discovery\_STM32↔  
F051\_CubeIDE/Prog/App/timer.c, [3244](#)
- Demo/ARMCM0\_STM32F0\_Discovery\_STM32↔  
F051\_CubeIDE/Prog/App/timer.h, [3417](#)
- Demo/ARMCM0\_STM32F0\_Discovery\_STM32↔  
F051\_GCC/Prog/timer.c, [3246](#)
- Demo/ARMCM0\_STM32F0\_Discovery\_STM32↔  
F051\_GCC/Prog/timer.h, [3419](#)
- Demo/ARMCM0\_STM32F0\_Discovery\_STM32↔  
F051\_IAR/Prog/timer.c, [3247](#)
- Demo/ARMCM0\_STM32F0\_Discovery\_STM32↔  
F051\_IAR/Prog/timer.h, [3420](#)
- Demo/ARMCM0\_STM32F0\_Discovery\_STM32↔  
F051\_Keil/Prog/timer.c, [3249](#)
- Demo/ARMCM0\_STM32F0\_Discovery\_STM32↔  
F051\_Keil/Prog/timer.h, [3421](#)
- Demo/ARMCM0\_STM32F0\_Nucleo\_F091RC↔  
CubeIDE/Prog/App/timer.c, [3250](#)
- Demo/ARMCM0\_STM32F0\_Nucleo\_F091RC↔  
CubeIDE/Prog/App/timer.h, [3423](#)
- Demo/ARMCM0\_STM32F0\_Nucleo\_F091RC↔  
GCC/Prog/timer.c, [3252](#)
- Demo/ARMCM0\_STM32F0\_Nucleo\_F091RC↔  
GCC/Prog/timer.h, [3424](#)
- Demo/ARMCM0\_STM32F0\_Nucleo\_F091RC↔  
IAR/Prog/timer.c, [3253](#)
- Demo/ARMCM0\_STM32F0\_Nucleo\_F091RC↔  
IAR/Prog/timer.h, [3425](#)
- Demo/ARMCM0\_STM32F0\_Nucleo\_F091RC↔  
Keil/Prog/timer.c, [3255](#)
- Demo/ARMCM0\_STM32F0\_Nucleo\_F091RC↔  
Keil/Prog/timer.h, [3427](#)
- Demo/ARMCM0\_STM32G0\_Nucleo\_G071RB↔  
CubeIDE/Prog/App/timer.c, [3256](#)
- Demo/ARMCM0\_STM32G0\_Nucleo\_G071RB↔  
CubeIDE/Prog/App/timer.h, [3428](#)
- Demo/ARMCM0\_STM32G0\_Nucleo\_G071RB↔  
GCC/Prog/timer.c, [3257](#)
- Demo/ARMCM0\_STM32G0\_Nucleo\_G071RB↔  
GCC/Prog/timer.h, [3429](#)
- Demo/ARMCM0\_STM32G0\_Nucleo\_G071RB↔  
IAR/Prog/timer.c, [3259](#)
- Demo/ARMCM0\_STM32G0\_Nucleo\_G071RB↔  
IAR/Prog/timer.h, [3431](#)
- Demo/ARMCM0\_STM32G0\_Nucleo\_G071RB↔  
Keil/Prog/timer.c, [3260](#)
- Demo/ARMCM0\_STM32G0\_Nucleo\_G071RB↔  
Keil/Prog/timer.h, [3432](#)
- Demo/ARMCM0\_XMC1\_XMC1400\_Boot\_Kit↔  
GCC/Prog/timer.c, [3262](#)
- Demo/ARMCM0\_XMC1\_XMC1400\_Boot\_Kit↔  
GCC/Prog/timer.h, [3434](#)
- Demo/ARMCM0\_XMC1\_XMC1400\_Boot\_Kit↔  
IAR/Prog/timer.c, [3265](#)
- Demo/ARMCM0\_XMC1\_XMC1400\_Boot\_Kit↔  
IAR/Prog/timer.h, [3436](#)
- Demo/ARMCM33\_STM32L5\_Nucleo\_L552ZE↔  
CubeIDE/Prog/App/timer.c, [3266](#)
- Demo/ARMCM33\_STM32L5\_Nucleo\_L552ZE↔  
CubeIDE/Prog/App/timer.h, [3438](#)
- Demo/ARMCM33\_STM32L5\_Nucleo\_L552ZE↔  
GCC/Prog/timer.c, [3268](#)
- Demo/ARMCM33\_STM32L5\_Nucleo\_L552ZE↔  
GCC/Prog/timer.h, [3439](#)
- Demo/ARMCM33\_STM32L5\_Nucleo\_L552ZE↔  
IAR/Prog/timer.c, [3269](#)
- Demo/ARMCM33\_STM32L5\_Nucleo\_L552ZE↔  
IAR/Prog/timer.h, [3440](#)

- Demo/ARMCM33\_STM32L5\_Nucleo\_L552ZE\_↔  
Keil/Prog/timer.c, [3271](#)
- Demo/ARMCM33\_STM32L5\_Nucleo\_L552ZE\_↔  
Keil/Prog/timer.h, [3442](#)
- Demo/ARMCM3\_EFM32\_Olimex\_EM32G880\_↔  
F128STK\_GCC/Prog/timer.c, [3272](#)
- Demo/ARMCM3\_EFM32\_Olimex\_EM32G880\_↔  
F128STK\_GCC/Prog/timer.h, [3443](#)
- Demo/ARMCM3\_EFM32\_Olimex\_EM32G880\_↔  
F128STK\_IAR/Prog/timer.c, [3275](#)
- Demo/ARMCM3\_EFM32\_Olimex\_EM32G880\_↔  
F128STK\_IAR/Prog/timer.h, [3446](#)
- Demo/ARMCM3\_STM32F1\_Nucleo\_F103RB\_↔  
CubeIDE/Prog/App/timer.c, [3276](#)
- Demo/ARMCM3\_STM32F1\_Nucleo\_F103RB\_↔  
CubeIDE/Prog/App/timer.h, [3448](#)
- Demo/ARMCM3\_STM32F1\_Nucleo\_F103RB\_G\_↔  
CC/Prog/timer.c, [3278](#)
- Demo/ARMCM3\_STM32F1\_Nucleo\_F103RB\_G\_↔  
CC/Prog/timer.h, [3449](#)
- Demo/ARMCM3\_STM32F1\_Nucleo\_F103RB\_I\_↔  
AR/Prog/timer.c, [3279](#)
- Demo/ARMCM3\_STM32F1\_Nucleo\_F103RB\_I\_↔  
AR/Prog/timer.h, [3450](#)
- Demo/ARMCM3\_STM32F1\_Nucleo\_F103RB\_↔  
Keil/Prog/timer.c, [3281](#)
- Demo/ARMCM3\_STM32F1\_Nucleo\_F103RB\_↔  
Keil/Prog/timer.h, [3452](#)
- Demo/ARMCM3\_STM32F1\_Olimex\_STM32\_↔  
H103\_CubeIDE/Prog/App/timer.c, [3282](#)
- Demo/ARMCM3\_STM32F1\_Olimex\_STM32\_↔  
H103\_CubeIDE/Prog/App/timer.h, [3453](#)
- Demo/ARMCM3\_STM32F1\_Olimex\_STM32\_↔  
H103\_GCC/Prog/timer.c, [3284](#)
- Demo/ARMCM3\_STM32F1\_Olimex\_STM32\_↔  
H103\_GCC/Prog/timer.h, [3454](#)
- Demo/ARMCM3\_STM32F1\_Olimex\_STM32\_↔  
H103\_IAR/Prog/timer.c, [3286](#)
- Demo/ARMCM3\_STM32F1\_Olimex\_STM32\_↔  
H103\_IAR/Prog/timer.h, [3456](#)
- Demo/ARMCM3\_STM32F1\_Olimex\_STM32\_↔  
H103\_Keil/Prog/timer.c, [3287](#)
- Demo/ARMCM3\_STM32F1\_Olimex\_STM32\_↔  
H103\_Keil/Prog/timer.h, [3457](#)
- Demo/ARMCM3\_STM32F1\_Olimex\_STM32\_↔  
P103\_CubeIDE/Prog/App/timer.c, [3289](#)
- Demo/ARMCM3\_STM32F1\_Olimex\_STM32\_↔  
P103\_CubeIDE/Prog/App/timer.h, [3458](#)
- Demo/ARMCM3\_STM32F1\_Olimex\_STM32\_↔  
P103\_GCC/Prog/timer.c, [3290](#)
- Demo/ARMCM3\_STM32F1\_Olimex\_STM32\_↔  
P103\_GCC/Prog/timer.h, [3460](#)
- Demo/ARMCM3\_STM32F1\_Olimex\_STM32\_↔  
P103\_IAR/Prog/timer.c, [3292](#)
- Demo/ARMCM3\_STM32F1\_Olimex\_STM32\_↔  
P103\_IAR/Prog/timer.h, [3461](#)
- Demo/ARMCM3\_STM32F1\_Olimex\_STM32\_↔  
P103\_Keil/Prog/timer.c, [3293](#)
- Demo/ARMCM3\_STM32F1\_Olimex\_STM32\_↔  
P103\_Keil/Prog/timer.h, [3462](#)
- Demo/ARMCM3\_STM32F1\_Olimexino\_STM32\_↔  
\_CubeIDE/Prog/App/timer.c, [3294](#)
- Demo/ARMCM3\_STM32F1\_Olimexino\_STM32\_↔  
\_CubeIDE/Prog/App/timer.h, [3464](#)
- Demo/ARMCM3\_STM32F1\_Olimexino\_STM32\_↔  
\_GCC/Prog/timer.c, [3296](#)
- Demo/ARMCM3\_STM32F1\_Olimexino\_STM32\_↔  
\_GCC/Prog/timer.h, [3465](#)
- Demo/ARMCM3\_STM32F1\_Olimexino\_STM32\_↔  
\_IAR/Prog/timer.c, [3297](#)
- Demo/ARMCM3\_STM32F1\_Olimexino\_STM32\_↔  
\_IAR/Prog/timer.h, [3466](#)
- Demo/ARMCM3\_STM32F1\_Olimexino\_STM32\_↔  
\_Keil/Prog/timer.c, [3299](#)
- Demo/ARMCM3\_STM32F1\_Olimexino\_STM32\_↔  
\_Keil/Prog/timer.h, [3468](#)
- Demo/ARMCM3\_STM32F2\_Olimex\_STM32\_↔  
P207\_CubeIDE/Prog/App/timer.c, [3300](#)
- Demo/ARMCM3\_STM32F2\_Olimex\_STM32\_↔  
P207\_CubeIDE/Prog/App/timer.h, [3469](#)
- Demo/ARMCM3\_STM32F2\_Olimex\_STM32\_↔  
P207\_GCC/Prog/timer.c, [3302](#)
- Demo/ARMCM3\_STM32F2\_Olimex\_STM32\_↔  
P207\_GCC/Prog/timer.h, [3470](#)
- Demo/ARMCM3\_STM32F2\_Olimex\_STM32\_↔  
P207\_IAR/Prog/timer.c, [3303](#)
- Demo/ARMCM3\_STM32F2\_Olimex\_STM32\_↔  
P207\_IAR/Prog/timer.h, [3472](#)
- Demo/ARMCM3\_STM32F2\_Olimex\_STM32\_↔  
P207\_Keil/Prog/timer.c, [3305](#)
- Demo/ARMCM3\_STM32F2\_Olimex\_STM32\_↔  
P207\_Keil/Prog/timer.h, [3473](#)
- Demo/ARMCM4\_S32K14\_S32K144EVB\_GCC\_↔  
Prog/timer.c, [3306](#)
- Demo/ARMCM4\_S32K14\_S32K144EVB\_GCC\_↔  
Prog/timer.h, [3474](#)
- Demo/ARMCM4\_S32K14\_S32K144EVB\_IAR\_↔  
Prog/timer.c, [3308](#)
- Demo/ARMCM4\_S32K14\_S32K144EVB\_IAR\_↔  
Prog/timer.h, [3476](#)
- Demo/ARMCM4\_STM32F3\_Discovery\_F303VC\_↔  
\_CubeIDE/Prog/App/timer.c, [3309](#)
- Demo/ARMCM4\_STM32F3\_Discovery\_F303VC\_↔  
\_CubeIDE/Prog/App/timer.h, [3477](#)
- Demo/ARMCM4\_STM32F3\_Discovery\_F303VC\_↔  
\_GCC/Prog/timer.c, [3311](#)
- Demo/ARMCM4\_STM32F3\_Discovery\_F303VC\_↔  
\_GCC/Prog/timer.h, [3478](#)
- Demo/ARMCM4\_STM32F3\_Discovery\_F303VC\_↔  
\_IAR/Prog/timer.c, [3312](#)
- Demo/ARMCM4\_STM32F3\_Discovery\_F303VC\_↔  
\_IAR/Prog/timer.h, [3480](#)
- Demo/ARMCM4\_STM32F3\_Discovery\_F303VC\_↔  
\_Keil/Prog/timer.c, [3314](#)
- Demo/ARMCM4\_STM32F3\_Discovery\_F303VC\_↔  
\_Keil/Prog/timer.h, [3481](#)

- Demo/ARMCM4\_STM32F3\_Nucleo\_F303K8\_↔  
CubeIDE/Prog/App/timer.c, [3315](#)
- Demo/ARMCM4\_STM32F3\_Nucleo\_F303K8\_↔  
CubeIDE/Prog/App/timer.h, [3482](#)
- Demo/ARMCM4\_STM32F3\_Nucleo\_F303K8\_G↔  
CC/Prog/timer.c, [3317](#)
- Demo/ARMCM4\_STM32F3\_Nucleo\_F303K8\_G↔  
CC/Prog/timer.h, [3484](#)
- Demo/ARMCM4\_STM32F3\_Nucleo\_F303K8\_IA↔  
R/Prog/timer.c, [3318](#)
- Demo/ARMCM4\_STM32F3\_Nucleo\_F303K8\_IA↔  
R/Prog/timer.h, [3485](#)
- Demo/ARMCM4\_STM32F3\_Nucleo\_F303K8\_↔  
Keil/Prog/timer.c, [3320](#)
- Demo/ARMCM4\_STM32F3\_Nucleo\_F303K8\_↔  
Keil/Prog/timer.h, [3486](#)
- Demo/ARMCM4\_STM32F4\_Nucleo\_F429ZI\_↔  
CubeIDE/Prog/App/timer.c, [3321](#)
- Demo/ARMCM4\_STM32F4\_Nucleo\_F429ZI\_↔  
CubeIDE/Prog/App/timer.h, [3488](#)
- Demo/ARMCM4\_STM32F4\_Nucleo\_F429ZI\_G↔  
CC/Prog/timer.c, [3323](#)
- Demo/ARMCM4\_STM32F4\_Nucleo\_F429ZI\_G↔  
CC/Prog/timer.h, [3489](#)
- Demo/ARMCM4\_STM32F4\_Nucleo\_F429ZI\_IA↔  
R/Prog/timer.c, [3324](#)
- Demo/ARMCM4\_STM32F4\_Nucleo\_F429ZI\_IA↔  
R/Prog/timer.h, [3490](#)
- Demo/ARMCM4\_STM32F4\_Nucleo\_F429ZI\_↔  
Keil/Prog/timer.c, [3326](#)
- Demo/ARMCM4\_STM32F4\_Nucleo\_F429ZI\_↔  
Keil/Prog/timer.h, [3492](#)
- Demo/ARMCM4\_STM32F4\_Olimex\_STM32↔  
P405\_CubeIDE/Prog/App/timer.c, [3327](#)
- Demo/ARMCM4\_STM32F4\_Olimex\_STM32↔  
P405\_CubeIDE/Prog/App/timer.h, [3493](#)
- Demo/ARMCM4\_STM32F4\_Olimex\_STM32↔  
P405\_GCC/Prog/timer.c, [3328](#)
- Demo/ARMCM4\_STM32F4\_Olimex\_STM32↔  
P405\_GCC/Prog/timer.h, [3494](#)
- Demo/ARMCM4\_STM32F4\_Olimex\_STM32↔  
P405\_IAR/Prog/timer.c, [3330](#)
- Demo/ARMCM4\_STM32F4\_Olimex\_STM32↔  
P405\_IAR/Prog/timer.h, [3496](#)
- Demo/ARMCM4\_STM32F4\_Olimex\_STM32↔  
P405\_Keil/Prog/timer.c, [3331](#)
- Demo/ARMCM4\_STM32F4\_Olimex\_STM32↔  
P405\_Keil/Prog/timer.h, [3497](#)
- Demo/ARMCM4\_STM32L4\_Nucleo\_L476RG\_↔  
CubeIDE/Prog/App/timer.c, [3333](#)
- Demo/ARMCM4\_STM32L4\_Nucleo\_L476RG\_↔  
CubeIDE/Prog/App/timer.h, [3498](#)
- Demo/ARMCM4\_STM32L4\_Nucleo\_L476RG\_G↔  
CC/Prog/timer.c, [3334](#)
- Demo/ARMCM4\_STM32L4\_Nucleo\_L476RG\_G↔  
CC/Prog/timer.h, [3500](#)
- Demo/ARMCM4\_STM32L4\_Nucleo\_L476RG\_IA↔  
AR/Prog/timer.c, [3336](#)
- Demo/ARMCM4\_STM32L4\_Nucleo\_L476RG\_IA↔  
AR/Prog/timer.h, [3501](#)
- Demo/ARMCM4\_STM32L4\_Nucleo\_L476RG\_↔  
Keil/Prog/timer.c, [3337](#)
- Demo/ARMCM4\_STM32L4\_Nucleo\_L476RG\_↔  
Keil/Prog/timer.h, [3502](#)
- Demo/ARMCM4\_XMC4\_XMC4700\_Relax\_Kit↔  
GCC/Prog/timer.c, [3339](#)
- Demo/ARMCM4\_XMC4\_XMC4700\_Relax\_Kit↔  
GCC/Prog/timer.h, [3504](#)
- Demo/ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_IA↔  
AR/Prog/timer.c, [3341](#)
- Demo/ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_IA↔  
AR/Prog/timer.h, [3506](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F746ZG\_↔  
CubeIDE/Prog/App/timer.c, [3343](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F746ZG\_↔  
CubeIDE/Prog/App/timer.h, [3508](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F746ZG\_G↔  
CC/Prog/timer.c, [3345](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F746ZG\_G↔  
CC/Prog/timer.h, [3509](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F746ZG\_IA↔  
AR/Prog/timer.c, [3346](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F746ZG\_IA↔  
AR/Prog/timer.h, [3510](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F746ZG\_↔  
Keil/Prog/timer.c, [3348](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F746ZG\_↔  
Keil/Prog/timer.h, [3512](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F767ZI\_↔  
CubeIDE/Prog/App/timer.c, [3349](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F767ZI\_↔  
CubeIDE/Prog/App/timer.h, [3513](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F767ZI\_G↔  
CC/Prog/timer.c, [3350](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F767ZI\_G↔  
CC/Prog/timer.h, [3514](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F767ZI\_IA↔  
R/Prog/timer.c, [3352](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F767ZI\_IA↔  
R/Prog/timer.h, [3516](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F767ZI\_↔  
Keil/Prog/timer.c, [3353](#)
- Demo/ARMCM7\_STM32F7\_Nucleo\_F767ZI\_↔  
Keil/Prog/timer.h, [3517](#)
- Demo/ARMCM7\_STM32H7\_Nucleo\_H743ZI\_↔  
CubeIDE/Prog/App/timer.c, [3355](#)
- Demo/ARMCM7\_STM32H7\_Nucleo\_H743ZI\_↔  
CubeIDE/Prog/App/timer.h, [3518](#)
- Demo/ARMCM7\_STM32H7\_Nucleo\_H743ZI\_G↔  
CC/Prog/timer.c, [3356](#)
- Demo/ARMCM7\_STM32H7\_Nucleo\_H743ZI\_G↔  
CC/Prog/timer.h, [3520](#)
- Demo/ARMCM7\_STM32H7\_Nucleo\_H743ZI\_IA↔  
R/Prog/timer.c, [3358](#)
- Demo/ARMCM7\_STM32H7\_Nucleo\_H743ZI\_IA↔  
R/Prog/timer.h, [3521](#)



- Demo/ARMCM7\_STM32H7\_Nucleo\_H743ZI\_↔  
Keil/Prog/timer.c, [3359](#)
- Demo/ARMCM7\_STM32H7\_Nucleo\_H743ZI\_↔  
Keil/Prog/timer.h, [3522](#)
- Demo/HCS12\_DevKit\_S12G128\_CodeWarrior\_↔  
Prog/timer.c, [3361](#)
- Demo/HCS12\_DevKit\_S12G128\_CodeWarrior\_↔  
Prog/timer.h, [3524](#)
- Demo/HCS12\_Evbplus\_Dragon12p\_Code↔  
Warrior/Prog/timer.c, [3363](#)
- Demo/HCS12\_Evbplus\_Dragon12p\_Code↔  
Warrior/Prog/timer.h, [3526](#)
- Source/\_template/timer.c, [3365](#)
- Source/ARMCM0\_S32K11/timer.c, [3367](#)
- Source/ARMCM0\_STM32F0/timer.c, [3370](#)
- Source/ARMCM0\_STM32G0/timer.c, [3373](#)
- Source/ARMCM0\_XMC1/timer.c, [3376](#)
- Source/ARMCM33\_STM32L5/timer.c, [3378](#)
- Source/ARMCM3\_EFM32/timer.c, [3381](#)
- Source/ARMCM3\_LM3S/timer.c, [3384](#)
- Source/ARMCM3\_STM32F1/timer.c, [3386](#)
- Source/ARMCM3\_STM32F2/timer.c, [3389](#)
- Source/ARMCM4\_S32K14/timer.c, [3392](#)
- Source/ARMCM4\_STM32F3/timer.c, [3394](#)
- Source/ARMCM4\_STM32F4/timer.c, [3397](#)
- Source/ARMCM4\_STM32L4/timer.c, [3400](#)
- Source/ARMCM4\_TM4C/timer.c, [3403](#)
- Source/ARMCM4\_XMC4/timer.c, [3406](#)
- Source/ARMCM7\_STM32F7/timer.c, [3408](#)
- Source/ARMCM7\_STM32H7/timer.c, [3411](#)
- Source/HCS12/timer.c, [3414](#)
- Source/timer.h, [3528](#)
- TimerInterrupt
  - Demo/\_template/Prog/timer.c, [3243](#)
- TimerReset
  - Source/\_template/timer.c, [3365](#)
  - Source/ARMCM0\_S32K11/timer.c, [3367](#)
  - Source/ARMCM0\_STM32F0/timer.c, [3370](#)
  - Source/ARMCM0\_STM32G0/timer.c, [3373](#)
  - Source/ARMCM0\_XMC1/timer.c, [3376](#)
  - Source/ARMCM33\_STM32L5/timer.c, [3379](#)
  - Source/ARMCM3\_EFM32/timer.c, [3381](#)
  - Source/ARMCM3\_LM3S/timer.c, [3384](#)
  - Source/ARMCM3\_STM32F1/timer.c, [3387](#)
  - Source/ARMCM3\_STM32F2/timer.c, [3390](#)
  - Source/ARMCM4\_S32K14/timer.c, [3392](#)
  - Source/ARMCM4\_STM32F3/timer.c, [3395](#)
  - Source/ARMCM4\_STM32F4/timer.c, [3398](#)
  - Source/ARMCM4\_STM32L4/timer.c, [3401](#)
  - Source/ARMCM4\_TM4C/timer.c, [3403](#)
  - Source/ARMCM4\_XMC4/timer.c, [3406](#)
  - Source/ARMCM7\_STM32F7/timer.c, [3409](#)
  - Source/ARMCM7\_STM32H7/timer.c, [3412](#)
  - Source/HCS12/timer.c, [3414](#)
  - Source/timer.h, [3529](#)
- TimerSet
  - ARMCM3\_LM3S\_EK\_LM3S6965\_GCC/Prog/time.↔  
c, [3221](#)
  - ARMCM3\_LM3S\_EK\_LM3S6965\_GCC/Prog/time.↔  
h, [3232](#)
  - ARMCM3\_LM3S\_EK\_LM3S6965\_IAR/Prog/time.↔  
c, [3223](#)
  - ARMCM3\_LM3S\_EK\_LM3S6965\_IAR/Prog/time.↔  
h, [3234](#)
  - ARMCM3\_LM3S\_EK\_LM3S8962\_GCC/Prog/time.↔  
c, [3225](#)
  - ARMCM3\_LM3S\_EK\_LM3S8962\_GCC/Prog/time.↔  
h, [3236](#)
  - ARMCM3\_LM3S\_EK\_LM3S8962\_IAR/Prog/time.↔  
c, [3227](#)
  - ARMCM3\_LM3S\_EK\_LM3S8962\_IAR/Prog/time.↔  
h, [3239](#)
  - ARMCM4\_TM4C\_DK\_TM4C123G\_IAR/Prog/time.↔  
c, [3229](#)
  - ARMCM4\_TM4C\_DK\_TM4C123G\_IAR/Prog/time.↔  
h, [3241](#)
  - Demo/ARMCM0\_XMC1\_XMC1400\_Boot\_Kit\_G↔  
CC/Prog/timer.c, [3263](#)
  - Demo/ARMCM0\_XMC1\_XMC1400\_Boot\_Kit\_G↔  
CC/Prog/timer.h, [3434](#)
  - Demo/ARMCM0\_XMC1\_XMC1400\_Boot\_Kit\_IA↔  
R/Prog/timer.c, [3265](#)
  - Demo/ARMCM0\_XMC1\_XMC1400\_Boot\_Kit\_IA↔  
R/Prog/timer.h, [3436](#)
  - Demo/ARMCM3\_EFM32\_Olimex\_EM32G880↔  
F128STK\_GCC/Prog/timer.c, [3273](#)
  - Demo/ARMCM3\_EFM32\_Olimex\_EM32G880↔  
F128STK\_GCC/Prog/timer.h, [3444](#)
  - Demo/ARMCM3\_EFM32\_Olimex\_EM32G880↔  
F128STK\_IAR/Prog/timer.c, [3275](#)
  - Demo/ARMCM3\_EFM32\_Olimex\_EM32G880↔  
F128STK\_IAR/Prog/timer.h, [3446](#)
  - Demo/ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_↔  
GCC/Prog/timer.c, [3339](#)
  - Demo/ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_↔  
GCC/Prog/timer.h, [3504](#)
  - Demo/ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_I↔  
AR/Prog/timer.c, [3342](#)
  - Demo/ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_I↔  
AR/Prog/timer.h, [3506](#)
  - Demo/HCS12\_DevKit\_S12G128\_CodeWarrior\_↔  
Prog/timer.c, [3361](#)
  - Demo/HCS12\_DevKit\_S12G128\_CodeWarrior\_↔  
Prog/timer.h, [3525](#)
  - Demo/HCS12\_Evbplus\_Dragon12p\_Code↔  
Warrior/Prog/timer.c, [3364](#)
  - Demo/HCS12\_Evbplus\_Dragon12p\_Code↔  
Warrior/Prog/timer.h, [3527](#)
- TimerUpdate
  - Source/\_template/timer.c, [3366](#)
  - Source/ARMCM0\_S32K11/timer.c, [3368](#)
  - Source/ARMCM0\_STM32F0/timer.c, [3371](#)
  - Source/ARMCM0\_STM32G0/timer.c, [3374](#)
  - Source/ARMCM0\_XMC1/timer.c, [3376](#)
  - Source/ARMCM33\_STM32L5/timer.c, [3379](#)
  - Source/ARMCM3\_EFM32/timer.c, [3382](#)

- Source/ARMCM3\_LM3S/timer.c, [3384](#)
- Source/ARMCM3\_STM32F1/timer.c, [3387](#)
- Source/ARMCM3\_STM32F2/timer.c, [3390](#)
- Source/ARMCM4\_S32K14/timer.c, [3392](#)
- Source/ARMCM4\_STM32F3/timer.c, [3395](#)
- Source/ARMCM4\_STM32F4/timer.c, [3398](#)
- Source/ARMCM4\_STM32L4/timer.c, [3401](#)
- Source/ARMCM4\_TM4C/timer.c, [3404](#)
- Source/ARMCM4\_XMC4/timer.c, [3406](#)
- Source/ARMCM7\_STM32F7/timer.c, [3409](#)
- Source/ARMCM7\_STM32H7/timer.c, [3412](#)
- Source/HCS12/timer.c, [3415](#)
- Source/timer.h, [3529](#)
- tios
  - tTimerRegs, [383](#)
- total\_size
  - tFileEraseInfo, [370](#)
- tscr1
  - tTimerRegs, [383](#)
- tscr2
  - tTimerRegs, [383](#)
- tseg1
  - tCanBusTiming, [357](#)
- tseg2
  - tCanBusTiming, [357](#)
- tstamp
  - tCanRxMsgSlot, [364](#)
  - tCanTxMsgSlot, [365](#)
- ttov
  - tTimerRegs, [383](#)
- txSlot
  - tCanRegs, [363](#)
- Type definitions of a port, [331](#)
- types.h, [3530](#), [3531](#), [3533](#), [3535](#), [3538](#), [3539](#), [3542](#), [3543](#), [3545](#), [3547](#), [3549](#), [3551](#), [3553](#), [3555](#), [3557](#), [3559](#), [3561](#), [3563](#), [3565](#)
- USB driver of a port, [332](#)
- uip\_tcp\_appstate\_t, [385](#)
- UnusedISR
  - ARMCM3\_EFM32\_Olimex\_EM32G880F128ST↔  
K\_GCC/Boot/vectors.c, [3624](#)
  - ARMCM3\_EFM32\_Olimex\_EM32G880F128ST↔  
K\_GCC/Prog/vectors.c, [3625](#)
  - ARMCM3\_EFM32\_Olimex\_EM32G880F128ST↔  
K\_IAR/Boot/vectors.c, [3627](#)
  - ARMCM3\_EFM32\_Olimex\_EM32G880F128ST↔  
K\_IAR/Prog/vectors.c, [3628](#)
  - ARMCM3\_LM3S\_EK\_LM3S6965\_GCC/Boot/vectors.↔  
c, [3629](#)
  - ARMCM3\_LM3S\_EK\_LM3S6965\_GCC/Prog/vectors.↔  
c, [3631](#)
  - ARMCM3\_LM3S\_EK\_LM3S6965\_IAR/Boot/vectors.↔  
c, [3632](#)
  - ARMCM3\_LM3S\_EK\_LM3S6965\_IAR/Prog/vectors.↔  
c, [3633](#)
  - ARMCM3\_LM3S\_EK\_LM3S8962\_GCC/Boot/vectors.↔  
c, [3635](#)
  - ARMCM3\_LM3S\_EK\_LM3S8962\_GCC/Prog/vectors.↔  
c, [3636](#)
  - ARMCM3\_LM3S\_EK\_LM3S8962\_IAR/Boot/vectors.↔  
c, [3638](#)
  - ARMCM3\_LM3S\_EK\_LM3S8962\_IAR/Prog/vectors.↔  
c, [3639](#)
  - ARMCM4\_TM4C\_DK\_TM4C123G\_IAR/Boot/vectors.↔  
c, [3640](#)
  - ARMCM4\_TM4C\_DK\_TM4C123G\_IAR/Prog/vectors.↔  
c, [3642](#)
- usb.c, [3568](#), [3574](#), [3580](#), [3587](#), [3593](#), [3599](#), [3607](#), [3613](#)
- usb.h, [3620](#)
  - UsbConnectHook, [3620](#)
  - UsbEnterLowPowerModeHook, [3621](#)
  - UsbFree, [3621](#)
  - UsbInit, [3621](#)
  - UsbLeaveLowPowerModeHook, [3621](#)
  - UsbReceivePacket, [3622](#)
  - UsbTransmitPacket, [3622](#)
- UsbBulkRxHandler
  - ARMCM4\_TM4C/usb.c, [3601](#)
- UsbBulkTxHandler
  - ARMCM4\_TM4C/usb.c, [3601](#)
- UsbConnectHook
  - ARMCM33\_STM32L5\_Nucleo\_L552ZE\_CubeID↔  
E/Boot/App/hooks.c, [1699](#)
  - ARMCM33\_STM32L5\_Nucleo\_L552ZE\_GCC↔  
Boot/hooks.c, [1705](#)
  - ARMCM33\_STM32L5\_Nucleo\_L552ZE\_IAR↔  
Boot/hooks.c, [1711](#)
  - ARMCM33\_STM32L5\_Nucleo\_L552ZE\_Keil↔  
Boot/hooks.c, [1717](#)
  - ARMCM3\_STM32F1\_Olimex\_STM32H103↔  
CubeIDE/Boot/App/hooks.c, [1779](#)
  - ARMCM3\_STM32F1\_Olimex\_STM32H103\_GC↔  
C/Boot/hooks.c, [1785](#)
  - ARMCM3\_STM32F1\_Olimex\_STM32H103\_IA↔  
R/Boot/hooks.c, [1791](#)
  - ARMCM3\_STM32F1\_Olimex\_STM32H103↔  
Keil/Boot/hooks.c, [1797](#)
  - ARMCM3\_STM32F1\_Olimexino\_STM32\_Cube↔  
DE/Boot/App/hooks.c, [1835](#)
  - ARMCM3\_STM32F1\_Olimexino\_STM32\_GCC↔  
Boot/hooks.c, [1844](#)
  - ARMCM3\_STM32F1\_Olimexino\_STM32\_IAR↔  
Boot/hooks.c, [1853](#)
  - ARMCM3\_STM32F1\_Olimexino\_STM32\_Keil↔  
Boot/hooks.c, [1862](#)
  - ARMCM4\_STM32F3\_Discovery\_F303VC\_Cube↔  
IDE/Boot/App/hooks.c, [1910](#)
  - ARMCM4\_STM32F3\_Discovery\_F303VC\_GCC↔  
Boot/hooks.c, [1916](#)
  - ARMCM4\_STM32F3\_Discovery\_F303VC\_IAR↔  
Boot/hooks.c, [1922](#)
  - ARMCM4\_STM32F3\_Discovery\_F303VC\_Keil↔  
Boot/hooks.c, [1928](#)
  - ARMCM4\_STM32F4\_Nucleo\_F429ZI\_CubeID↔  
E/Boot/App/hooks.c, [1954](#)

- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_GCC/↔  
Boot/hooks.c, [1960](#)
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_IAR/↔  
Boot/hooks.c, [1966](#)
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_Keil/↔  
Boot/hooks.c, [1972](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_↔  
CubeIDE/Boot/App/hooks.c, [1980](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_GC↔  
C/Boot/hooks.c, [1988](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_IA↔  
R/Boot/hooks.c, [1997](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_Keil/↔  
Boot/hooks.c, [2006](#)
- ARMCM4\_TM4C\_DK\_TM4C123G\_IAR/Boot/hooks.↔  
c, [2035](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_CubeID↔  
E/Boot/App/hooks.c, [2057](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_GCC/↔  
Boot/hooks.c, [2064](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_IAR/↔  
Boot/hooks.c, [2070](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_Keil/↔  
Boot/hooks.c, [2076](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_CubeID↔  
E/Boot/App/hooks.c, [2102](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_GCC/↔  
Boot/hooks.c, [2108](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_IAR/↔  
Boot/hooks.c, [2114](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_Keil/↔  
Boot/hooks.c, [2120](#)
- usb.h, [3620](#)
- UsbEnterLowPowerModeHook
  - ARMCM33\_STM32L5\_Nucleo\_L552ZE\_CubeID↔  
E/Boot/App/hooks.c, [1699](#)
  - ARMCM33\_STM32L5\_Nucleo\_L552ZE\_GCC/↔  
Boot/hooks.c, [1705](#)
  - ARMCM33\_STM32L5\_Nucleo\_L552ZE\_IAR/↔  
Boot/hooks.c, [1711](#)
  - ARMCM33\_STM32L5\_Nucleo\_L552ZE\_Keil/↔  
Boot/hooks.c, [1717](#)
  - ARMCM3\_STM32F1\_Olimex\_STM32H103\_↔  
CubeIDE/Boot/App/hooks.c, [1779](#)
  - ARMCM3\_STM32F1\_Olimex\_STM32H103\_GC↔  
C/Boot/hooks.c, [1785](#)
  - ARMCM3\_STM32F1\_Olimex\_STM32H103\_IA↔  
R/Boot/hooks.c, [1791](#)
  - ARMCM3\_STM32F1\_Olimex\_STM32H103\_↔  
Keil/Boot/hooks.c, [1797](#)
  - ARMCM3\_STM32F1\_Olimexino\_STM32\_CubeID↔  
DE/Boot/App/hooks.c, [1836](#)
  - ARMCM3\_STM32F1\_Olimexino\_STM32\_GCC/↔  
Boot/hooks.c, [1845](#)
  - ARMCM3\_STM32F1\_Olimexino\_STM32\_IAR/↔  
Boot/hooks.c, [1854](#)
  - ARMCM3\_STM32F1\_Olimexino\_STM32\_Keil/↔  
Boot/hooks.c, [1863](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_Cube↔  
IDE/Boot/App/hooks.c, [1910](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_GCC/↔  
Boot/hooks.c, [1916](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_IAR/↔  
Boot/hooks.c, [1922](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_Keil/↔  
Boot/hooks.c, [1928](#)
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_CubeID↔  
E/Boot/App/hooks.c, [1954](#)
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_GCC/↔  
Boot/hooks.c, [1960](#)
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_IAR/↔  
Boot/hooks.c, [1966](#)
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_Keil/↔  
Boot/hooks.c, [1972](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_↔  
CubeIDE/Boot/App/hooks.c, [1980](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_GC↔  
C/Boot/hooks.c, [1989](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_IA↔  
R/Boot/hooks.c, [1998](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_Keil/↔  
Boot/hooks.c, [2007](#)
- ARMCM4\_TM4C\_DK\_TM4C123G\_IAR/Boot/hooks.↔  
c, [2036](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_CubeID↔  
E/Boot/App/hooks.c, [2058](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_GCC/↔  
Boot/hooks.c, [2064](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_IAR/↔  
Boot/hooks.c, [2070](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_Keil/↔  
Boot/hooks.c, [2076](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_CubeID↔  
E/Boot/App/hooks.c, [2102](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_GCC/↔  
Boot/hooks.c, [2108](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_IAR/↔  
Boot/hooks.c, [2114](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_Keil/↔  
Boot/hooks.c, [2120](#)
- usb.h, [3621](#)
- UsbFifoMgrCreate
  - \_template/usb.c, [3569](#)
  - ARMCM33\_STM32L5/usb.c, [3576](#)
  - ARMCM3\_STM32F1/usb.c, [3582](#)
  - ARMCM4\_STM32F3/usb.c, [3588](#)
  - ARMCM4\_STM32F4/usb.c, [3595](#)
  - ARMCM4\_TM4C/usb.c, [3602](#)
  - ARMCM7\_STM32F7/usb.c, [3609](#)
  - ARMCM7\_STM32H7/usb.c, [3615](#)
- UsbFifoMgrInit
  - \_template/usb.c, [3570](#)
  - ARMCM33\_STM32L5/usb.c, [3576](#)
  - ARMCM3\_STM32F1/usb.c, [3582](#)
  - ARMCM4\_STM32F3/usb.c, [3589](#)
  - ARMCM4\_STM32F4/usb.c, [3595](#)

- ARMCM4\_TM4C/usb.c, [3602](#)
- ARMCM7\_STM32F7/usb.c, [3609](#)
- ARMCM7\_STM32H7/usb.c, [3615](#)
- UsbFifoMgrRead
  - \_template/usb.c, [3570](#)
  - ARMCM33\_STM32L5/usb.c, [3576](#)
  - ARMCM3\_STM32F1/usb.c, [3583](#)
  - ARMCM4\_STM32F3/usb.c, [3589](#)
  - ARMCM4\_STM32F4/usb.c, [3595](#)
  - ARMCM4\_TM4C/usb.c, [3603](#)
  - ARMCM7\_STM32F7/usb.c, [3609](#)
  - ARMCM7\_STM32H7/usb.c, [3616](#)
- UsbFifoMgrScan
  - \_template/usb.c, [3570](#)
  - ARMCM33\_STM32L5/usb.c, [3577](#)
  - ARMCM3\_STM32F1/usb.c, [3583](#)
  - ARMCM4\_STM32F3/usb.c, [3589](#)
  - ARMCM4\_STM32F4/usb.c, [3596](#)
  - ARMCM4\_TM4C/usb.c, [3603](#)
  - ARMCM7\_STM32F7/usb.c, [3610](#)
  - ARMCM7\_STM32H7/usb.c, [3616](#)
- UsbFifoMgrWrite
  - \_template/usb.c, [3571](#)
  - ARMCM33\_STM32L5/usb.c, [3577](#)
  - ARMCM3\_STM32F1/usb.c, [3583](#)
  - ARMCM4\_STM32F3/usb.c, [3590](#)
  - ARMCM4\_STM32F4/usb.c, [3596](#)
  - ARMCM4\_TM4C/usb.c, [3603](#)
  - ARMCM7\_STM32F7/usb.c, [3610](#)
  - ARMCM7\_STM32H7/usb.c, [3616](#)
- UsbFree
  - \_template/usb.c, [3571](#)
  - ARMCM33\_STM32L5/usb.c, [3578](#)
  - ARMCM3\_STM32F1/usb.c, [3584](#)
  - ARMCM4\_STM32F3/usb.c, [3590](#)
  - ARMCM4\_STM32F4/usb.c, [3597](#)
  - ARMCM4\_TM4C/usb.c, [3604](#)
  - ARMCM7\_STM32F7/usb.c, [3611](#)
  - ARMCM7\_STM32H7/usb.c, [3617](#)
  - usb.h, [3621](#)
- UsbInit
  - \_template/usb.c, [3571](#)
  - ARMCM33\_STM32L5/usb.c, [3578](#)
  - ARMCM3\_STM32F1/usb.c, [3584](#)
  - ARMCM4\_STM32F3/usb.c, [3590](#)
  - ARMCM4\_STM32F4/usb.c, [3597](#)
  - ARMCM4\_TM4C/usb.c, [3604](#)
  - ARMCM7\_STM32F7/usb.c, [3611](#)
  - ARMCM7\_STM32H7/usb.c, [3617](#)
  - usb.h, [3621](#)
- UsbLeaveLowPowerModeHook
  - ARMCM33\_STM32L5\_Nucleo\_L552ZE\_CubeID↔  
E/Boot/App/hooks.c, [1699](#)
  - ARMCM33\_STM32L5\_Nucleo\_L552ZE\_GCC/↔  
Boot/hooks.c, [1705](#)
  - ARMCM33\_STM32L5\_Nucleo\_L552ZE\_IAR/↔  
Boot/hooks.c, [1711](#)
  - ARMCM33\_STM32L5\_Nucleo\_L552ZE\_Keil/↔  
Boot/hooks.c, [1717](#)
  - ARMCM3\_STM32F1\_Olimex\_STM32H103\_↔  
CubeIDE/Boot/App/hooks.c, [1779](#)
  - ARMCM3\_STM32F1\_Olimex\_STM32H103\_GC↔  
C/Boot/hooks.c, [1785](#)
  - ARMCM3\_STM32F1\_Olimex\_STM32H103\_IA↔  
R/Boot/hooks.c, [1791](#)
  - ARMCM3\_STM32F1\_Olimex\_STM32H103\_↔  
Keil/Boot/hooks.c, [1797](#)
  - ARMCM3\_STM32F1\_Olimexino\_STM32\_Cube↔  
DE/Boot/App/hooks.c, [1836](#)
  - ARMCM3\_STM32F1\_Olimexino\_STM32\_GCC/↔  
Boot/hooks.c, [1845](#)
  - ARMCM3\_STM32F1\_Olimexino\_STM32\_IAR/↔  
Boot/hooks.c, [1854](#)
  - ARMCM3\_STM32F1\_Olimexino\_STM32\_Keil/↔  
Boot/hooks.c, [1863](#)
  - ARMCM4\_STM32F3\_Discovery\_F303VC\_Cube↔  
IDE/Boot/App/hooks.c, [1910](#)
  - ARMCM4\_STM32F3\_Discovery\_F303VC\_GCC/↔  
Boot/hooks.c, [1916](#)
  - ARMCM4\_STM32F3\_Discovery\_F303VC\_IAR/↔  
Boot/hooks.c, [1922](#)
  - ARMCM4\_STM32F3\_Discovery\_F303VC\_Keil/↔  
Boot/hooks.c, [1928](#)
  - ARMCM4\_STM32F4\_Nucleo\_F429ZI\_CubeID↔  
E/Boot/App/hooks.c, [1954](#)
  - ARMCM4\_STM32F4\_Nucleo\_F429ZI\_GCC/↔  
Boot/hooks.c, [1960](#)
  - ARMCM4\_STM32F4\_Nucleo\_F429ZI\_IAR/↔  
Boot/hooks.c, [1966](#)
  - ARMCM4\_STM32F4\_Nucleo\_F429ZI\_Keil/↔  
Boot/hooks.c, [1972](#)
  - ARMCM4\_STM32F4\_Olimex\_STM32P405\_↔  
CubeIDE/Boot/App/hooks.c, [1980](#)
  - ARMCM4\_STM32F4\_Olimex\_STM32P405\_GC↔  
C/Boot/hooks.c, [1989](#)
  - ARMCM4\_STM32F4\_Olimex\_STM32P405\_IA↔  
R/Boot/hooks.c, [1998](#)
  - ARMCM4\_STM32F4\_Olimex\_STM32P405\_Keil/↔  
Boot/hooks.c, [2007](#)
  - ARMCM4\_TM4C\_DK\_TM4C123G\_IAR/Boot/hooks.↔  
c, [2036](#)
  - ARMCM7\_STM32F7\_Nucleo\_F746ZG\_CubeID↔  
E/Boot/App/hooks.c, [2058](#)
  - ARMCM7\_STM32F7\_Nucleo\_F746ZG\_GCC/↔  
Boot/hooks.c, [2064](#)
  - ARMCM7\_STM32F7\_Nucleo\_F746ZG\_IAR/↔  
Boot/hooks.c, [2070](#)
  - ARMCM7\_STM32F7\_Nucleo\_F746ZG\_Keil/↔  
Boot/hooks.c, [2076](#)
  - ARMCM7\_STM32H7\_Nucleo\_H743ZI\_CubeID↔  
E/Boot/App/hooks.c, [2102](#)
  - ARMCM7\_STM32H7\_Nucleo\_H743ZI\_GCC/↔  
Boot/hooks.c, [2108](#)
  - ARMCM7\_STM32H7\_Nucleo\_H743ZI\_IAR/↔  
Boot/hooks.c, [2114](#)

- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_Keil/↔
  - Boot/hooks.c, 2120
  - usb.h, 3621
- UsbReceiveByte
  - \_template/usb.c, 3572
  - ARMCM33\_STM32L5/usb.c, 3578
  - ARMCM3\_STM32F1/usb.c, 3584
  - ARMCM4\_STM32F3/usb.c, 3591
  - ARMCM4\_STM32F4/usb.c, 3597
  - ARMCM4\_TM4C/usb.c, 3604
  - ARMCM7\_STM32F7/usb.c, 3611
  - ARMCM7\_STM32H7/usb.c, 3617
- UsbReceivePacket
  - \_template/usb.c, 3572
  - ARMCM33\_STM32L5/usb.c, 3578
  - ARMCM3\_STM32F1/usb.c, 3585
  - ARMCM4\_STM32F3/usb.c, 3591
  - ARMCM4\_STM32F4/usb.c, 3597
  - ARMCM4\_TM4C/usb.c, 3605
  - ARMCM7\_STM32F7/usb.c, 3611
  - ARMCM7\_STM32H7/usb.c, 3618
  - usb.h, 3622
- UsbReceivePipeBulkOUT
  - \_template/usb.c, 3573
  - ARMCM33\_STM32L5/usb.c, 3579
  - ARMCM3\_STM32F1/usb.c, 3585
  - ARMCM4\_STM32F3/usb.c, 3591
  - ARMCM4\_STM32F4/usb.c, 3598
  - ARMCM4\_TM4C/usb.c, 3605
  - ARMCM7\_STM32F7/usb.c, 3612
  - ARMCM7\_STM32H7/usb.c, 3618
- UsbTransmitByte
  - \_template/usb.c, 3573
  - ARMCM33\_STM32L5/usb.c, 3579
  - ARMCM3\_STM32F1/usb.c, 3585
  - ARMCM4\_STM32F3/usb.c, 3592
  - ARMCM4\_STM32F4/usb.c, 3598
  - ARMCM4\_TM4C/usb.c, 3606
  - ARMCM7\_STM32F7/usb.c, 3612
  - ARMCM7\_STM32H7/usb.c, 3618
- UsbTransmitPacket
  - \_template/usb.c, 3573
  - ARMCM33\_STM32L5/usb.c, 3579
  - ARMCM3\_STM32F1/usb.c, 3586
  - ARMCM4\_STM32F3/usb.c, 3592
  - ARMCM4\_STM32F4/usb.c, 3598
  - ARMCM4\_TM4C/usb.c, 3606
  - ARMCM7\_STM32F7/usb.c, 3612
  - ARMCM7\_STM32H7/usb.c, 3619
  - usb.h, 3622
- UsbTransmitPipeBulkIN
  - \_template/usb.c, 3574
  - ARMCM33\_STM32L5/usb.c, 3580
  - ARMCM3\_STM32F1/usb.c, 3586
  - ARMCM4\_STM32F3/usb.c, 3592
  - ARMCM4\_STM32F4/usb.c, 3599
  - ARMCM4\_TM4C/usb.c, 3606
  - ARMCM7\_STM32F7/usb.c, 3613
- ARMCM7\_STM32H7/usb.c, 3619
- User Program, 63, 66, 69, 72, 75, 78, 81, 84, 87, 90, 93, 96, 99, 102, 105, 108, 111, 114, 117, 120, 123, 126, 129, 132, 135, 138, 141, 144, 147, 150, 153, 156, 159, 162, 165, 168, 171, 174, 177, 180, 183, 186, 189, 192, 195, 198, 201, 204, 207, 210, 213, 216, 219, 222, 225, 228, 231, 234, 237, 240, 243, 246, 249, 252, 255, 258, 261, 264, 267, 270, 273, 276, 279, 282, 285, 288, 291, 294, 297, 300, 303, 306, 309, 312, 319, 322
- VAL
  - tSysTickRegs, 380
- VCT\_USER\_PROGRAM\_VECTOR\_TABLE\_START↔
  - ADDR
  - HCS12\_DevKit\_S12G128\_CodeWarrior/Boot/vectors.↔  
c, 3647
  - HCS12\_Evbplus\_Dragon12p\_CodeWarrior/↔  
Boot/vectors.c, 3669
- Vector0\_handler
  - HCS12\_DevKit\_S12G128\_CodeWarrior/Boot/vectors.↔  
c, 3647
  - HCS12\_Evbplus\_Dragon12p\_CodeWarrior/↔  
Boot/vectors.c, 3670
- Vector10\_handler
  - HCS12\_DevKit\_S12G128\_CodeWarrior/Boot/vectors.↔  
c, 3647
  - HCS12\_Evbplus\_Dragon12p\_CodeWarrior/↔  
Boot/vectors.c, 3670
- Vector11\_handler
  - HCS12\_DevKit\_S12G128\_CodeWarrior/Boot/vectors.↔  
c, 3648
  - HCS12\_Evbplus\_Dragon12p\_CodeWarrior/↔  
Boot/vectors.c, 3671
- Vector12\_handler
  - HCS12\_DevKit\_S12G128\_CodeWarrior/Boot/vectors.↔  
c, 3648
  - HCS12\_Evbplus\_Dragon12p\_CodeWarrior/↔  
Boot/vectors.c, 3671
- Vector13\_handler
  - HCS12\_DevKit\_S12G128\_CodeWarrior/Boot/vectors.↔  
c, 3648
  - HCS12\_Evbplus\_Dragon12p\_CodeWarrior/↔  
Boot/vectors.c, 3671
- Vector14\_handler
  - HCS12\_DevKit\_S12G128\_CodeWarrior/Boot/vectors.↔  
c, 3648
  - HCS12\_Evbplus\_Dragon12p\_CodeWarrior/↔  
Boot/vectors.c, 3671
- Vector15\_handler
  - HCS12\_DevKit\_S12G128\_CodeWarrior/Boot/vectors.↔  
c, 3649
  - HCS12\_Evbplus\_Dragon12p\_CodeWarrior/↔  
Boot/vectors.c, 3672
- Vector16\_handler
  - HCS12\_DevKit\_S12G128\_CodeWarrior/Boot/vectors.↔  
c, 3649





[illegible]

- Vector59\_handler
  - HCS12\_DevKit\_S12G128\_CodeWarrior/Boot/vectors.↔  
c, [3661](#)
  - HCS12\_Evbplus\_Dragon12p\_CodeWarrior/↔  
Boot/vectors.c, [3684](#)
- Vector5\_handler
  - HCS12\_DevKit\_S12G128\_CodeWarrior/Boot/vectors.↔  
c, [3661](#)
  - HCS12\_Evbplus\_Dragon12p\_CodeWarrior/↔  
Boot/vectors.c, [3684](#)
- Vector60\_handler
  - HCS12\_DevKit\_S12G128\_CodeWarrior/Boot/vectors.↔  
c, [3661](#)
  - HCS12\_Evbplus\_Dragon12p\_CodeWarrior/↔  
Boot/vectors.c, [3684](#)
- Vector61\_handler
  - HCS12\_DevKit\_S12G128\_CodeWarrior/Boot/vectors.↔  
c, [3661](#)
  - HCS12\_Evbplus\_Dragon12p\_CodeWarrior/↔  
Boot/vectors.c, [3684](#)
- Vector62\_handler
  - HCS12\_DevKit\_S12G128\_CodeWarrior/Boot/vectors.↔  
c, [3662](#)
  - HCS12\_Evbplus\_Dragon12p\_CodeWarrior/↔  
Boot/vectors.c, [3685](#)
- Vector6\_handler
  - HCS12\_DevKit\_S12G128\_CodeWarrior/Boot/vectors.↔  
c, [3662](#)
  - HCS12\_Evbplus\_Dragon12p\_CodeWarrior/↔  
Boot/vectors.c, [3685](#)
- Vector7\_handler
  - HCS12\_DevKit\_S12G128\_CodeWarrior/Boot/vectors.↔  
c, [3662](#)
  - HCS12\_Evbplus\_Dragon12p\_CodeWarrior/↔  
Boot/vectors.c, [3685](#)
- Vector8\_handler
  - HCS12\_DevKit\_S12G128\_CodeWarrior/Boot/vectors.↔  
c, [3662](#)
  - HCS12\_Evbplus\_Dragon12p\_CodeWarrior/↔  
Boot/vectors.c, [3685](#)
- Vector9\_handler
  - HCS12\_DevKit\_S12G128\_CodeWarrior/Boot/vectors.↔  
c, [3663](#)
  - HCS12\_Evbplus\_Dragon12p\_CodeWarrior/↔  
Boot/vectors.c, [3686](#)
- VectorBase\_Config
  - ARMCM0\_STM32G0\_Nucleo\_G071RB\_GCC/↔  
Prog/main.c, [2641](#)
  - ARMCM0\_STM32G0\_Nucleo\_G071RB\_IAR/↔  
Prog/main.c, [2646](#)
  - ARMCM0\_STM32G0\_Nucleo\_G071RB\_Keil/↔  
Prog/main.c, [2651](#)
  - ARMCM33\_STM32L5\_Nucleo\_L552ZE\_GCC/↔  
Prog/main.c, [2661](#)
  - ARMCM33\_STM32L5\_Nucleo\_L552ZE\_IAR/↔  
Prog/main.c, [2666](#)
  - ARMCM33\_STM32L5\_Nucleo\_L552ZE\_Keil/↔  
Prog/main.c, [2671](#)
  - ARMCM3\_STM32F1\_Nucleo\_F103RB\_GCC/↔  
Prog/main.c, [2692](#)
  - ARMCM3\_STM32F1\_Nucleo\_F103RB\_IAR/↔  
Prog/main.c, [2697](#)
  - ARMCM3\_STM32F1\_Nucleo\_F103RB\_Keil/↔  
Prog/main.c, [2702](#)
  - ARMCM3\_STM32F1\_Olimex\_STM32H103\_GC↔  
C/Prog/main.c, [2707](#)
  - ARMCM3\_STM32F1\_Olimex\_STM32H103\_IA↔  
R/Prog/main.c, [2712](#)
  - ARMCM3\_STM32F1\_Olimex\_STM32H103\_↔  
Keil/Prog/main.c, [2717](#)
  - ARMCM3\_STM32F1\_Olimex\_STM32P103\_GC↔  
C/Prog/main.c, [2722](#)
  - ARMCM3\_STM32F1\_Olimex\_STM32P103\_IA↔  
R/Prog/main.c, [2727](#)
  - ARMCM3\_STM32F1\_Olimex\_STM32P103\_Keil/↔  
Prog/main.c, [2732](#)
  - ARMCM3\_STM32F1\_Olimexino\_STM32\_GCC/↔  
Prog/main.c, [2737](#)
  - ARMCM3\_STM32F1\_Olimexino\_STM32\_IAR/↔  
Prog/main.c, [2742](#)
  - ARMCM3\_STM32F1\_Olimexino\_STM32\_Keil/↔  
Prog/main.c, [2747](#)
  - ARMCM3\_STM32F2\_Olimex\_STM32P207\_GC↔  
C/Prog/main.c, [2752](#)
  - ARMCM3\_STM32F2\_Olimex\_STM32P207\_IA↔  
R/Prog/main.c, [2757](#)
  - ARMCM3\_STM32F2\_Olimex\_STM32P207\_Keil/↔  
Prog/main.c, [2762](#)
  - ARMCM4\_STM32F3\_Discovery\_F303VC\_GCC/↔  
Prog/main.c, [2774](#)
  - ARMCM4\_STM32F3\_Discovery\_F303VC\_IAR/↔  
Prog/main.c, [2779](#)
  - ARMCM4\_STM32F3\_Discovery\_F303VC\_Keil/↔  
Prog/main.c, [2784](#)
  - ARMCM4\_STM32F3\_Nucleo\_F303K8\_GCC/↔  
Prog/main.c, [2789](#)
  - ARMCM4\_STM32F3\_Nucleo\_F303K8\_IAR/↔  
Prog/main.c, [2794](#)
  - ARMCM4\_STM32F3\_Nucleo\_F303K8\_Keil/↔  
Prog/main.c, [2799](#)
  - ARMCM4\_STM32F4\_Nucleo\_F429ZI\_GCC/↔  
Prog/main.c, [2804](#)
  - ARMCM4\_STM32F4\_Nucleo\_F429ZI\_IAR/↔  
Prog/main.c, [2809](#)
  - ARMCM4\_STM32F4\_Nucleo\_F429ZI\_Keil/↔  
Prog/main.c, [2814](#)
  - ARMCM4\_STM32F4\_Olimex\_STM32P405\_GC↔  
C/Prog/main.c, [2819](#)
  - ARMCM4\_STM32F4\_Olimex\_STM32P405\_IA↔  
R/Prog/main.c, [2824](#)
  - ARMCM4\_STM32F4\_Olimex\_STM32P405\_Keil/↔  
Prog/main.c, [2829](#)
  - ARMCM4\_STM32L4\_Nucleo\_L476RG\_GCC/↔  
Prog/main.c, [2834](#)
  - ARMCM4\_STM32L4\_Nucleo\_L476RG\_IAR/↔  
Prog/main.c, [2839](#)



- ARMCM4\_STM32L4\_Nucleo\_L476RG\_Keil/↔  
  Prog/main.c, [2844](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_GCC/↔  
  Prog/main.c, [2858](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_IAR/↔  
  Prog/main.c, [2863](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_Keil/↔  
  Prog/main.c, [2868](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_GCC/↔  
  Prog/main.c, [2873](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_IAR/↔  
  Prog/main.c, [2878](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_Keil/↔  
  Prog/main.c, [2883](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_GCC/↔  
  Prog/main.c, [2888](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_IAR/↔  
  Prog/main.c, [2893](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_Keil/↔  
  Prog/main.c, [2898](#)
- vectors.c, [3623](#), [3624](#), [3626–3628](#), [3630](#), [3631](#), [3633–3635](#), [3637–3639](#), [3641](#), [3642](#), [3663](#), [3664](#), [3686](#)
- writeptr  
  tFifoCtrl, [368](#)
- XCP\_PACKET\_RECEIVED\_HOOK\_EN  
  xcp.h, [3703](#)
- xcp.c, [3687](#)
  - XcpCmdBuildChecksum, [3689](#)
  - XcpCmdConnect, [3689](#)
  - XcpCmdDisconnect, [3690](#)
  - XcpCmdGetId, [3690](#)
  - XcpCmdGetSeed, [3691](#)
  - XcpCmdGetStatus, [3691](#)
  - XcpCmdProgram, [3691](#)
  - XcpCmdProgramClear, [3692](#)
  - XcpCmdProgramMax, [3692](#)
  - XcpCmdProgramPrepare, [3692](#)
  - XcpCmdProgramReset, [3693](#)
  - XcpCmdProgramStart, [3693](#)
  - XcpCmdSetMta, [3694](#)
  - XcpCmdShortUpload, [3694](#)
  - XcpCmdSynch, [3694](#)
  - XcpCmdUnlock, [3695](#)
  - XcpCmdUpload, [3695](#)
  - XcpComputeChecksum, [3695](#)
  - XcpGetSeed, [3696](#)
  - XcpGetSeedHook, [3696](#)
  - XcpInit, [3697](#)
  - XcplsConnected, [3697](#)
  - XcpPacketReceived, [3697](#)
  - XcpPacketTransmitted, [3698](#)
  - XcpProtectResources, [3698](#)
  - XcpSetCtoError, [3698](#)
  - XcpTransmitPacket, [3699](#)
  - XcpVerifyKey, [3699](#)
  - XcpVerifyKeyHook, [3700](#)
- xcp.h, [3700](#)
  - XCP\_PACKET\_RECEIVED\_HOOK\_EN, [3703](#)
  - XcpInit, [3704](#)
  - XcplsConnected, [3704](#)
  - XcpPacketReceived, [3704](#)
  - XcpPacketTransmitted, [3705](#)
- XcpCmdBuildChecksum  
  xcp.c, [3689](#)
- XcpCmdConnect  
  xcp.c, [3689](#)
- XcpCmdDisconnect  
  xcp.c, [3690](#)
- XcpCmdGetId  
  xcp.c, [3690](#)
- XcpCmdGetSeed  
  xcp.c, [3691](#)
- XcpCmdGetStatus  
  xcp.c, [3691](#)
- XcpCmdProgram  
  xcp.c, [3691](#)
- XcpCmdProgramClear  
  xcp.c, [3692](#)
- XcpCmdProgramMax  
  xcp.c, [3692](#)
- XcpCmdProgramPrepare  
  xcp.c, [3692](#)
- XcpCmdProgramReset  
  xcp.c, [3693](#)
- XcpCmdProgramStart  
  xcp.c, [3693](#)
- XcpCmdSetMta  
  xcp.c, [3694](#)
- XcpCmdShortUpload  
  xcp.c, [3694](#)
- XcpCmdSynch  
  xcp.c, [3694](#)
- XcpCmdUnlock  
  xcp.c, [3695](#)
- XcpCmdUpload  
  xcp.c, [3695](#)
- XcpComputeChecksum  
  xcp.c, [3695](#)
- XcpGetSeed  
  xcp.c, [3696](#)
- XcpGetSeedHook  
  \_template/Boot/hooks.c, [1624](#)
  - ARMCM0\_STM32F0\_Discovery\_STM32F051\_↔  
  CubeIDE/Boot/App/hooks.c, [1629](#)
  - ARMCM0\_STM32F0\_Discovery\_STM32F051\_↔  
  GCC/Boot/hooks.c, [1634](#)
  - ARMCM0\_STM32F0\_Discovery\_STM32F051\_I↔  
  AR/Boot/hooks.c, [1639](#)
  - ARMCM0\_STM32F0\_Discovery\_STM32F051\_↔  
  Keil/Boot/hooks.c, [1644](#)
  - ARMCM0\_STM32F0\_Nucleo\_F091RC\_CubeID↔  
  E/Boot/App/hooks.c, [1649](#)
  - ARMCM0\_STM32F0\_Nucleo\_F091RC\_GCC/↔  
  Boot/hooks.c, [1654](#)

- ARMCM0\_STM32F0\_Nucleo\_F091RC\_IAR/↔  
Boot/hooks.c, [1659](#)
- ARMCM0\_STM32F0\_Nucleo\_F091RC\_Keil/↔  
Boot/hooks.c, [1664](#)
- ARMCM0\_STM32G0\_Nucleo\_G071RB\_CubeID↔  
E/Boot/App/hooks.c, [1669](#)
- ARMCM0\_STM32G0\_Nucleo\_G071RB\_GCC/↔  
Boot/hooks.c, [1674](#)
- ARMCM0\_STM32G0\_Nucleo\_G071RB\_IAR/↔  
Boot/hooks.c, [1679](#)
- ARMCM0\_STM32G0\_Nucleo\_G071RB\_Keil/↔  
Boot/hooks.c, [1684](#)
- ARMCM0\_XMC1\_XMC1400\_Boot\_Kit\_GCC/↔  
Boot/hooks.c, [1689](#)
- ARMCM0\_XMC1\_XMC1400\_Boot\_Kit\_IAR/↔  
Boot/hooks.c, [1694](#)
- ARMCM33\_STM32L5\_Nucleo\_L552ZE\_CubeID↔  
E/Boot/App/hooks.c, [1700](#)
- ARMCM33\_STM32L5\_Nucleo\_L552ZE\_GCC/↔  
Boot/hooks.c, [1705](#)
- ARMCM33\_STM32L5\_Nucleo\_L552ZE\_IAR/↔  
Boot/hooks.c, [1711](#)
- ARMCM33\_STM32L5\_Nucleo\_L552ZE\_Keil/↔  
Boot/hooks.c, [1717](#)
- ARMCM3\_EFM32\_Olimex\_EM32G880F128ST↔  
K\_GCC/Boot/hooks.c, [1723](#)
- ARMCM3\_EFM32\_Olimex\_EM32G880F128ST↔  
K\_IAR/Boot/hooks.c, [1728](#)
- ARMCM3\_LM3S\_EK\_LM3S6965\_GCC/Boot/hooks.↔  
c, [1735](#)
- ARMCM3\_LM3S\_EK\_LM3S6965\_IAR/Boot/hooks.↔  
c, [1743](#)
- ARMCM3\_LM3S\_EK\_LM3S8962\_GCC/Boot/hooks.↔  
c, [1748](#)
- ARMCM3\_LM3S\_EK\_LM3S8962\_IAR/Boot/hooks.↔  
c, [1754](#)
- ARMCM3\_STM32F1\_Nucleo\_F103RB\_CubeID↔  
E/Boot/App/hooks.c, [1759](#)
- ARMCM3\_STM32F1\_Nucleo\_F103RB\_GCC/↔  
Boot/hooks.c, [1764](#)
- ARMCM3\_STM32F1\_Nucleo\_F103RB\_IAR/↔  
Boot/hooks.c, [1769](#)
- ARMCM3\_STM32F1\_Nucleo\_F103RB\_Keil/↔  
Boot/hooks.c, [1774](#)
- ARMCM3\_STM32F1\_Olimex\_STM32H103\_↔  
CubeIDE/Boot/App/hooks.c, [1779](#)
- ARMCM3\_STM32F1\_Olimex\_STM32H103\_GC↔  
C/Boot/hooks.c, [1785](#)
- ARMCM3\_STM32F1\_Olimex\_STM32H103\_IA↔  
R/Boot/hooks.c, [1791](#)
- ARMCM3\_STM32F1\_Olimex\_STM32H103\_↔  
Keil/Boot/hooks.c, [1797](#)
- ARMCM3\_STM32F1\_Olimex\_STM32P103\_↔  
CubeIDE/Boot/App/hooks.c, [1805](#)
- ARMCM3\_STM32F1\_Olimex\_STM32P103\_GC↔  
C/Boot/hooks.c, [1812](#)
- ARMCM3\_STM32F1\_Olimex\_STM32P103\_IA↔  
R/Boot/hooks.c, [1820](#)
- ARMCM3\_STM32F1\_Olimex\_STM32P103\_Keil/↔  
Boot/hooks.c, [1828](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_Cube↔  
DE/Boot/App/hooks.c, [1836](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_GCC/↔  
Boot/hooks.c, [1845](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_IAR/↔  
Boot/hooks.c, [1854](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_Keil/↔  
Boot/hooks.c, [1863](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_↔  
CubeIDE/Boot/App/hooks.c, [1871](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_GC↔  
C/Boot/hooks.c, [1878](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_IA↔  
R/Boot/hooks.c, [1886](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_Keil/↔  
Boot/hooks.c, [1894](#)
- ARMCM4\_S32K14\_S32K144EVB\_GCC/Boot/hooks.↔  
c, [1899](#)
- ARMCM4\_S32K14\_S32K144EVB\_IAR/Boot/hooks.↔  
c, [1905](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_Cube↔  
IDE/Boot/App/hooks.c, [1910](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_GCC/↔  
Boot/hooks.c, [1916](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_IAR/↔  
Boot/hooks.c, [1922](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_Keil/↔  
Boot/hooks.c, [1928](#)
- ARMCM4\_STM32F3\_Nucleo\_F303K8\_CubeID↔  
E/Boot/App/hooks.c, [1934](#)
- ARMCM4\_STM32F3\_Nucleo\_F303K8\_GCC/↔  
Boot/hooks.c, [1939](#)
- ARMCM4\_STM32F3\_Nucleo\_F303K8\_IAR/↔  
Boot/hooks.c, [1944](#)
- ARMCM4\_STM32F3\_Nucleo\_F303K8\_Keil/↔  
Boot/hooks.c, [1949](#)
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_CubeID↔  
E/Boot/App/hooks.c, [1954](#)
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_GCC/↔  
Boot/hooks.c, [1960](#)
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_IAR/↔  
Boot/hooks.c, [1966](#)
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_Keil/↔  
Boot/hooks.c, [1972](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_↔  
CubeIDE/Boot/App/hooks.c, [1980](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_GC↔  
C/Boot/hooks.c, [1989](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_IA↔  
R/Boot/hooks.c, [1998](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_Keil/↔  
Boot/hooks.c, [2007](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_CubeID↔  
E/Boot/App/hooks.c, [2013](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_GCC/↔  
Boot/hooks.c, [2018](#)

- ARMCM4\_STM32L4\_Nucleo\_L476RG\_IAR/↔  
Boot/hooks.c, [2023](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_Keil/↔  
Boot/hooks.c, [2028](#)
- ARMCM4\_TM4C\_DK\_TM4C123G\_IAR/Boot/hooks.↔  
c, [2036](#)
- ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_GCC/↔  
Boot/hooks.c, [2044](#)
- ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_IAR/↔  
Boot/hooks.c, [2052](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_CubeID↔  
E/Boot/App/hooks.c, [2058](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_GCC/↔  
Boot/hooks.c, [2064](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_IAR/↔  
Boot/hooks.c, [2070](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_Keil/↔  
Boot/hooks.c, [2076](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_CubeID↔  
E/Boot/App/hooks.c, [2082](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_GCC/↔  
Boot/hooks.c, [2087](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_IAR/↔  
Boot/hooks.c, [2092](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_Keil/↔  
Boot/hooks.c, [2097](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_CubeID↔  
E/Boot/App/hooks.c, [2102](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_GCC/↔  
Boot/hooks.c, [2108](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_IAR/↔  
Boot/hooks.c, [2114](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_Keil/↔  
Boot/hooks.c, [2120](#)
- HCS12\_DevKit\_S12G128\_CodeWarrior/Boot/hooks.↔  
c, [2126](#)
- HCS12\_Evbplus\_Dragon12p\_CodeWarrior/↔  
Boot/hooks.c, [2131](#)
- xcp.c, [3696](#)
- XcpInit  
xcp.c, [3697](#)  
xcp.h, [3704](#)
- XcpIsConnected  
xcp.c, [3697](#)  
xcp.h, [3704](#)
- XcpPacketReceived  
xcp.c, [3697](#)  
xcp.h, [3704](#)
- XcpPacketTransmitted  
xcp.c, [3698](#)  
xcp.h, [3705](#)
- XcpProtectResources  
xcp.c, [3698](#)
- XcpSetCtoError  
xcp.c, [3698](#)
- XcpTransmitPacket  
xcp.c, [3699](#)
- XcpVerifyKey  
xcp.c, [3699](#)
- XcpVerifyKeyHook  
\_template/Boot/hooks.c, [1624](#)
- ARMCM0\_STM32F0\_Discovery\_STM32F051\_↔  
CubeIDE/Boot/App/hooks.c, [1629](#)
- ARMCM0\_STM32F0\_Discovery\_STM32F051\_↔  
GCC/Boot/hooks.c, [1634](#)
- ARMCM0\_STM32F0\_Discovery\_STM32F051\_I↔  
AR/Boot/hooks.c, [1639](#)
- ARMCM0\_STM32F0\_Discovery\_STM32F051\_↔  
Keil/Boot/hooks.c, [1644](#)
- ARMCM0\_STM32F0\_Nucleo\_F091RC\_CubeID↔  
E/Boot/App/hooks.c, [1649](#)
- ARMCM0\_STM32F0\_Nucleo\_F091RC\_GCC/↔  
Boot/hooks.c, [1654](#)
- ARMCM0\_STM32F0\_Nucleo\_F091RC\_IAR/↔  
Boot/hooks.c, [1659](#)
- ARMCM0\_STM32F0\_Nucleo\_F091RC\_Keil/↔  
Boot/hooks.c, [1664](#)
- ARMCM0\_STM32G0\_Nucleo\_G071RB\_CubeID↔  
E/Boot/App/hooks.c, [1669](#)
- ARMCM0\_STM32G0\_Nucleo\_G071RB\_GCC/↔  
Boot/hooks.c, [1674](#)
- ARMCM0\_STM32G0\_Nucleo\_G071RB\_IAR/↔  
Boot/hooks.c, [1679](#)
- ARMCM0\_STM32G0\_Nucleo\_G071RB\_Keil/↔  
Boot/hooks.c, [1684](#)
- ARMCM0\_XMC1\_XMC1400\_Boot\_Kit\_GCC/↔  
Boot/hooks.c, [1689](#)
- ARMCM0\_XMC1\_XMC1400\_Boot\_Kit\_IAR/↔  
Boot/hooks.c, [1694](#)
- ARMCM33\_STM32L5\_Nucleo\_L552ZE\_CubeID↔  
E/Boot/App/hooks.c, [1700](#)
- ARMCM33\_STM32L5\_Nucleo\_L552ZE\_GCC/↔  
Boot/hooks.c, [1706](#)
- ARMCM33\_STM32L5\_Nucleo\_L552ZE\_IAR/↔  
Boot/hooks.c, [1712](#)
- ARMCM33\_STM32L5\_Nucleo\_L552ZE\_Keil/↔  
Boot/hooks.c, [1718](#)
- ARMCM3\_EFM32\_Olimex\_EM32G880F128ST↔  
K\_GCC/Boot/hooks.c, [1723](#)
- ARMCM3\_EFM32\_Olimex\_EM32G880F128ST↔  
K\_IAR/Boot/hooks.c, [1728](#)
- ARMCM3\_LM3S\_EK\_LM3S6965\_GCC/Boot/hooks.↔  
c, [1736](#)
- ARMCM3\_LM3S\_EK\_LM3S6965\_IAR/Boot/hooks.↔  
c, [1743](#)
- ARMCM3\_LM3S\_EK\_LM3S8962\_GCC/Boot/hooks.↔  
c, [1749](#)
- ARMCM3\_LM3S\_EK\_LM3S8962\_IAR/Boot/hooks.↔  
c, [1754](#)
- ARMCM3\_STM32F1\_Nucleo\_F103RB\_CubeID↔  
E/Boot/App/hooks.c, [1759](#)
- ARMCM3\_STM32F1\_Nucleo\_F103RB\_GCC/↔  
Boot/hooks.c, [1764](#)
- ARMCM3\_STM32F1\_Nucleo\_F103RB\_IAR/↔  
Boot/hooks.c, [1769](#)
- ARMCM3\_STM32F1\_Nucleo\_F103RB\_Keil/↔

- Boot/hooks.c, [1774](#)
- ARMCM3\_STM32F1\_Olimex\_STM32H103\_↔  
CubeIDE/Boot/App/hooks.c, [1780](#)
- ARMCM3\_STM32F1\_Olimex\_STM32H103\_GC↔  
C/Boot/hooks.c, [1786](#)
- ARMCM3\_STM32F1\_Olimex\_STM32H103\_IAR↔  
R/Boot/hooks.c, [1792](#)
- ARMCM3\_STM32F1\_Olimex\_STM32H103\_↔  
Keil/Boot/hooks.c, [1798](#)
- ARMCM3\_STM32F1\_Olimex\_STM32P103\_↔  
CubeIDE/Boot/App/hooks.c, [1805](#)
- ARMCM3\_STM32F1\_Olimex\_STM32P103\_GC↔  
C/Boot/hooks.c, [1813](#)
- ARMCM3\_STM32F1\_Olimex\_STM32P103\_IAR↔  
R/Boot/hooks.c, [1821](#)
- ARMCM3\_STM32F1\_Olimex\_STM32P103\_Keil/↔  
Boot/hooks.c, [1828](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_Cube↔  
IDE/Boot/App/hooks.c, [1837](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_GCC/↔  
Boot/hooks.c, [1846](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_IAR/↔  
Boot/hooks.c, [1855](#)
- ARMCM3\_STM32F1\_Olimexino\_STM32\_Keil/↔  
Boot/hooks.c, [1864](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_↔  
CubeIDE/Boot/App/hooks.c, [1871](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_GC↔  
C/Boot/hooks.c, [1879](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_IAR↔  
R/Boot/hooks.c, [1887](#)
- ARMCM3\_STM32F2\_Olimex\_STM32P207\_Keil/↔  
Boot/hooks.c, [1894](#)
- ARMCM4\_S32K14\_S32K144EVB\_GCC/Boot/hooks.↔  
c, [1900](#)
- ARMCM4\_S32K14\_S32K144EVB\_IAR/Boot/hooks.↔  
c, [1905](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_Cube↔  
IDE/Boot/App/hooks.c, [1911](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_GCC/↔  
Boot/hooks.c, [1917](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_IAR/↔  
Boot/hooks.c, [1923](#)
- ARMCM4\_STM32F3\_Discovery\_F303VC\_Keil/↔  
Boot/hooks.c, [1929](#)
- ARMCM4\_STM32F3\_Nucleo\_F303K8\_CubeID↔  
E/Boot/App/hooks.c, [1934](#)
- ARMCM4\_STM32F3\_Nucleo\_F303K8\_GCC/↔  
Boot/hooks.c, [1939](#)
- ARMCM4\_STM32F3\_Nucleo\_F303K8\_IAR/↔  
Boot/hooks.c, [1944](#)
- ARMCM4\_STM32F3\_Nucleo\_F303K8\_Keil/↔  
Boot/hooks.c, [1949](#)
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_CubeID↔  
E/Boot/App/hooks.c, [1955](#)
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_GCC/↔  
Boot/hooks.c, [1961](#)
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_IAR/↔  
Boot/hooks.c, [1967](#)
- ARMCM4\_STM32F4\_Nucleo\_F429ZI\_Keil/↔  
Boot/hooks.c, [1973](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_↔  
CubeIDE/Boot/App/hooks.c, [1981](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_GC↔  
C/Boot/hooks.c, [1990](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_IAR↔  
R/Boot/hooks.c, [1999](#)
- ARMCM4\_STM32F4\_Olimex\_STM32P405\_Keil/↔  
Boot/hooks.c, [2008](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_CubeID↔  
E/Boot/App/hooks.c, [2013](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_GCC/↔  
Boot/hooks.c, [2018](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_IAR/↔  
Boot/hooks.c, [2023](#)
- ARMCM4\_STM32L4\_Nucleo\_L476RG\_Keil/↔  
Boot/hooks.c, [2028](#)
- ARMCM4\_TM4C\_DK\_TM4C123G\_IAR/Boot/hooks.↔  
c, [2037](#)
- ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_GCC/↔  
Boot/hooks.c, [2044](#)
- ARMCM4\_XMC4\_XMC4700\_Relax\_Kit\_IAR/↔  
Boot/hooks.c, [2052](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_CubeID↔  
E/Boot/App/hooks.c, [2059](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_GCC/↔  
Boot/hooks.c, [2065](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_IAR/↔  
Boot/hooks.c, [2071](#)
- ARMCM7\_STM32F7\_Nucleo\_F746ZG\_Keil/↔  
Boot/hooks.c, [2077](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_CubeID↔  
E/Boot/App/hooks.c, [2082](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_GCC/↔  
Boot/hooks.c, [2087](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_IAR/↔  
Boot/hooks.c, [2092](#)
- ARMCM7\_STM32F7\_Nucleo\_F767ZI\_Keil/↔  
Boot/hooks.c, [2097](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_CubeID↔  
E/Boot/App/hooks.c, [2103](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_GCC/↔  
Boot/hooks.c, [2109](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_IAR/↔  
Boot/hooks.c, [2115](#)
- ARMCM7\_STM32H7\_Nucleo\_H743ZI\_Keil/↔  
Boot/hooks.c, [2121](#)
- HCS12\_DevKit\_S12G128\_CodeWarrior/Boot/hooks.↔  
c, [2126](#)
- HCS12\_Evbplus\_Dragon12p\_CodeWarrior/↔  
Boot/hooks.c, [2132](#)
- xcp.c, [3700](#)