RSL15 Firmware Reference

M-20871-005
June 2023

© SCILLC, 2023
Previous Edition © 2022

Table of Contents

Page

RSL15 Firmware Reference 1

1. Introduction 61

1.1 Purpose 61

1.2 Intended Audience 61

1.3 Conventions 61

1.4 Further Reading 62

2. Firmware Overview 63

2.1 Introduction 63

2.1.1 Compliance Exceptions 63

2.2 Supported Data Types 63

2.3 Firmware Components 64

2.4 Firmware Naming Conventions 64

2.5 Firmware Resource Usage 65

2.6 Versions 67

3. Hardware Definitions 69

3.1 Register and Register Bit-Field Definitions 69

3.2 Memory Map Definition 70

3.3 Interrupt Vector Definition 71

4. Hardware Abstraction Layer 72

4.1 Usage 72

5. CMSIS Library 73

5.1 Introduction 73

6. CMSIS Drivers 74

6.1 Introduction 74

6.2 Using the CMSIS Drivers 74

6.2.1 Adding a CMSIS Driver 74

6.2.2 Configuring CMSIS Drivers with RTE_Device.h 75

7. Program ROM 77

7.1 Overview 77

7.1.1 Versions 77

7.2 ROM Initialization Sequence 77

7.2.1 ROM Basic Initialization 78

7.2.2 ROM Status Variables & Status Codes 78

7.3 Application Validation and Boot 80

7.4 Vector Tables 81

7.5 Security Subsystem 84

7.5.1 Secure Boot Process 84

7.6 ROM Life Cycle States (LCS) & Operational Modes 84

7.7 Data Exchange Unit (DEU) Support 85

7.8 Reserved Flash Sectors 85

8. Flash Library 87

8.1 Library Forms 87

8.2 Flash Library Usage 87

9. Security and Life Cycle Provisioning Elements 88

9.1 Overview 88

9.2 Third Party Documentation 88

10. Arm CryptoCell-312 Security IP 89

10.1 Arm CryptoCell-312 Mbed TLS 89

11. Event Kernel 90

11.1 Overview 90

11.1.1 Include Files 90

11.1.2 Kernel Environment 90

11.2 Messages 90

11.2.1 Overview 90

11.2.2 Message Format 91

11.2.3 Parameter Management 91

11.2.4 Message Queue Primitives 91

11.2.4.1 Message Allocation 91

Prototype: 91

Parameters: 92

Return: 92

Description: 92

11.2.4.2 Message Allocation Macro (Static Structure) 92

Prototype: 92

Parameters: 92

Return: 92

Description: 92

11.2.4.3 Message Allocation Macro (Variable Structure) 93

Prototype: 93

Parameters: 93

Return: 93

Description: 93

11.2.4.4 Message Free 93

Prototype: 93

Parameters: 93

Return: 93

Description: 94

11.2.4.5 Message Free Macro 94

Prototype: 94

Parameters: 94

Return: 94

Description: 94

11.2.4.6 Message Send 94

Prototype: 94

Parameters: 94

Return: 94

Description: 94

11.2.4.7 Message Send Basic 95

Prototype: 95

Parameters: 95

Return: 95

Description: 95

11.2.4.8 Message Forward 95

Prototype: 95

Parameters: 95

Return: 95

Description: 96

11.3 Scheduler 96

11.3.1 Overview 96

11.3.2 Requirements 96

11.3.2.1 Scheduling Algorithm 96

11.3.2.2 Save Service 97

11.4 Tasks 97

11.5 Kernel Timer 98

11.5.1 Overview 98

11.5.2 Time Definition 98

11.5.3 Timer Object 98

11.5.4 Timer Setting 98

11.5.5 Time Primitives 99

11.5.5.1 Timer Set 99

Prototype: 99

Parameters: 99

Return: 100

Description: 100

11.5.5.2 Timer Clear 100

Prototype: 100

Parameters: 100

Return: 100

Description: 100

11.5.5.3 Timer Activity 100

Prototype: 100

Parameters: 100

Return: 101

Description: 101

11.5.5.4 Timer Adjust 101

Prototype: 101

Parameters: 101

Return: 101

Description: 101

11.5.5.5 Timer Expiry 101

11.6 Useful Macros 102

12. Bluetooth Stack 103

12.1 Introduction 103

12.1.1 Include and Object Files 103

12.1.2 Bluetooth Stack 103

12.1.3 Stack Support Functions 104

12.1.4 Managing Bluetooth Low Energy Stack RAM Usage 104

12.2 HCI 106

12.2.1 HCI Software Architecture 107

12.3 GATT 108

12.4 GAP 109

12.4.1 Non-Connected Procedures 109

12.4.1.1 Activity Overview 110

12.4.1.2 Advertising Activity 111

12.4.1.2.1 Advertising Properties 112

12.4.1.2.2 Legacy Advertising Activity 113

12.4.1.2.3 Extended Advertising Activity 113

12.4.1.2.4 Periodic Advertising Activity 114

12.4.1.3 Scanning Activity 114

12.4.1.4 Initiating Activity 115

12.4.1.5 Periodic Synchronization Activity 116

13. Bluetooth and Kernel Library 117

13.1 Use of the Event Kernel and Bluetooth Stack 117

13.1.1 The Kernel Scheduler 117

13.1.2 Message Handlers 117

13.1.3 Core Bluetooth Profiles 118

13.1.4 Standard Profiles and Services 118

13.1.5 Custom Profiles and Services 118

13.1.6 Event Kernel and Bluetooth Low Energy Library Initialization and Execution 118

13.2 Baseband and Kernel Functions 119

13.2.1 BLE_Initialize 119

Prototype: 119

Parameters: 120

Return: 120

Description: 120

Example 120

13.2.2 BLE_Baseband_Sleep 120

Prototype: 120

Parameters: 120

Return: 121

Description: 121

Example: 122

13.2.3 BLE_Kernel_Process 122

Prototype: 122

Parameters: 122

Return: 122

Description: 122

Example: 122

13.2.4 BLE_Baseband_Is_Awake 123

Prototype: 123

Parameters: 123

Return: 123

Description: 123

Example: 123

13.3 Managing Bluetooth Low Energy Stack RAM Usage 123

13.4 Bluetooth Low Energy Link Layer Metrics 125

13.4.1 BLE_Link_Metrics 127

13.5 Bluetooth Stack Supporting API Functions 128

13.5.1 Device_BLE_Param_Get 128

Prototype: 128

13.5.2 platform_reset 129

Prototype: 129

13.5.3 srand_func 129

Prototype: 129

13.5.4 rand_func 129

Prototype: 129

13.5.5 Device_RF_RSSI_Convert 130

Prototype: 130

13.5.6 Device_RF_TxPwr_Get_dBm 130

Prototype: 130

13.5.7 Device_RF_TxPwr_Get_Idx 130

Prototype: 130

13.5.8 BLE_Set_RxStatusCallBack 130

Prototype: 130

Parameters: 130

Return: 130

Description: 130

Examples: 131

13.5.9 Device_RF_SetMaxPwrIdx 131

Prototype: 131

13.5.10 BLE_Set_ScanConIndStatusCallBack 132

Prototype: 132

Parameters: 132

Return: 132

Description: 132

Example: 132

13.5.11 Hci_Vs_Cmd_App_Func 132

Prototype: 132

Parameters: 133

Return: 133

Description: 133

Example: 133

13.5.12 BLE_Set_MaxRFConnectionPwr 133

Prototype: 133

Parameters: 133

Return: 134

Description: 134

Example: 134

13.5.13 uint32_t ke_get_max_mem_usage 134

Prototype: 134

Parameters: 134

Return: 134

Description: 134

Example: 134

13.5.14 uint16_t ke_get_mem_usage 135

Prototype: 135

Parameters: 135

Return: 135

Description: 135

Example: 135

13.6 Bluetooth Low Energy Abstraction 135

14. swmTrace Library 137

14.1 Introduction 137

14.2 onsemi IDE Setup 137

14.3 Usage 137

15. CMSIS Reference 138

15.1 Summary 138

15.2 CMSIS Reference Variable Documentation 140

15.2.1 RSL15_Sys_Version 140

15.2.2 __Heap_Begin__ 140

15.2.3 __Heap_Limit__ 140

15.2.4 __stack_limit 140

15.2.5 __stack 140

15.2.6 __data_init__ 141

15.2.7 __data_start__ 141

15.2.8 __data_end__ 141

15.2.9 __bss_start__ 141

15.2.10 __bss_end__ 141

15.2.11 __preinit_array_start__ 142

15.2.12 __preinit_array_end__ 142

15.2.13 __init_array_start__ 142

15.2.14 __init_array_end__ 142

15.2.15 flash_layout 143

15.2.16 SystemCoreClock 143

15.3 CMSIS Reference Data Structures Type Documentation 143

15.3.1 flash_region 143

15.4 CMSIS Reference Macro Definition Documentation 144

15.4.1 RSL15_SYS_VER_MAJOR 144

15.4.2 RSL15_SYS_VER_MINOR 144

15.4.3 RSL15_SYS_VER_REVISION 144

15.4.4 RSL15_SYS_VER 144

15.4.5 __ARMv8MML_REV 145

15.4.6 __CM33_REV 145

15.4.7 __FPU_PRESENT 145

15.4.8 __DSP_PRESENT 145

15.4.9 __SAUREGION_PRESENT 146

15.4.10 __MPU_PRESENT 146

15.4.11 __VTOR_PRESENT 146

15.4.12 __NVIC_PRIO_BITS 146

15.4.13 __Vendor_SysTickConfig 146

15.4.14 I2C_REF_VALID 147

15.4.15 LIN_REF_VALID 147

15.4.16 PCM_REF_VALID 147

15.4.17 PWM_REF_VALID 147

15.4.18 SPI_REF_VALID 148

15.4.19 UART_REF_VALID 148

15.4.20 TIMER_REF_VALID 148

15.4.21 DMA_REF_VALID 148

15.4.22 FLASH_REF_VALID 149

15.4.23 GPIO_PAD_COUNT 149

15.4.24 GPIO_GROUP_LOW_PAD_RANGE 149

15.4.25 GPIO_EVENT_CHANNEL_COUNT 149

15.4.26 GPIO0 149

15.4.27 GPIO1 150

15.4.28 GPIO2 150

15.4.29 GPIO3 150

15.4.30 GPIO4 150

15.4.31 GPIO5 151

15.4.32 GPIO6 151

15.4.33 GPIO7 151

15.4.34 GPIO8 151

15.4.35 GPIO9 151

15.4.36 GPIO10 152

15.4.37 GPIO11 152

15.4.38 GPIO12 152

15.4.39 GPIO13 152

15.4.40 GPIO14 152

15.4.41 GPIO15 153

15.4.42 SYS_DUMMY_READ 153

15.4.43 SYS_DUMMY_WRITE 153

15.4.44 ERRNO_NO_ERROR 153

15.4.45 ERRNO_GENERAL_FAILURE 154

15.4.46 DEFAULT_FREQ 154

15.4.47 STANDBYCLK_DEFAULT_FREQ 154

15.4.48 RFCLK_BASE_FREQ 154

15.4.49 EXTCLK_MAX_FREQ 154

15.4.50 SWCLK_MAX_FREQ 155

15.4.51 RCOSC_MAX_FREQ 155

15.5 CMSIS Reference Function Documentation 155

15.5.1 _start 155

15.5.2 _sbrk 156

15.5.3 SystemInit 157

15.5.4 SystemCoreClockUpdate 158

16. Hardware Abstraction Layer Reference 159

16.1 Summary 159

16.2 Activity Counter 159

16.2.1 Summary 159

16.2.2 Activity Counter Function Documentation 159

16.2.2.1 Sys_ACNT_Start 159

16.2.2.2 Sys_ACNT_Stop 159

16.2.2.3 Sys_ACNT_Clear 160

16.3 Analog Control System 160

16.3.1 Summary 160

16.3.2 Analog Control System Function Documentation 160

16.3.2.1 Sys_ACS_WriteRegister 161

16.4 Asynchronous Clock Counter 161

16.4.1 Summary 161

16.4.2 Asynchronous Clock Counter Function Documentation 161

16.4.2.1 Sys_ASCC_GPIOConfig 161

16.4.2.2 Sys_ASCC_Config 162

16.4.2.3 Sys_ASCC_StartCounters 163

16.5 Baseband Interface 163

16.5.1 Summary 163

16.5.2 Baseband Interface Macro Definition Documentation 164

16.5.2.1 BLE_NONE 164

16.5.2.2 BLE_RISING_EDGE 164

16.5.2.3 BLE_FALLING_EDGE 164

16.5.2.4 BLE_TRANSITION 164

16.5.3 Baseband Interface Function Documentation 165

16.5.3.1 Sys_BBIF_CoexIntConfig 165

16.6 Clock Configuration 166

16.6.1 Summary 166

16.6.2 Clock Configuration Variable Documentation 166

16.6.2.1 SystemCoreClock 166

16.6.3 Clock Configuration Function Documentation 166

16.6.3.1 Sys_Clocks_XTALClk_Wait 166

16.6.3.2 Sys_Clocks_RCSystemClkConfig 167

16.6.3.3 Sys_Clocks_SystemClkConfig 167

16.6.3.4 Sys_Clocks_XTALClkConfig 168

16.6.3.5 Sys_Clocks_DividerConfig 169

16.7 Arm Cortex-M Processor 170

16.7.1 Summary 170

16.7.2 Arm Cortex-M Processor Macro Definition Documentation 170

16.7.2.1 SYS_WAIT_FOR_EVENT 170

16.7.2.2 SYS_WAIT_FOR_INTERRUPT 171

16.8 Cyclic Redundancy Check 171

16.8.1 Summary 171

16.8.2 Cyclic Redundancy Check Macro Definition Documentation 172

16.8.2.1 SYS_CRC_CONFIG 172

16.8.2.2 SYS_CRC_32INITVALUE 172

16.8.2.3 SYS_CRC_CCITTINITVALUE 173

16.8.2.4 SYS_CRC_GETCURRENTVALUE 174

16.8.2.5 SYS_CRC_GETFINALVALUE 174

16.8.2.6 SYS_CRC_ADD 175

16.8.3 Cyclic Redundancy Check Function Documentation 175

16.8.3.1 Sys_Set_CRC_Config 175

16.8.3.2 Sys_CRC_32InitValue 176

16.8.3.3 Sys_CRC_CCITTInitValue 177

16.8.3.4 Sys_CRC_GetCurrentValue 177

16.8.3.5 Sys_CRC_GetFinalValue 178

16.8.3.6 Sys_CRC_Add 179

16.9 Direct Memory Access 179

16.9.1 Summary 180

16.9.2 Direct Memory Access Macro Definition Documentation 180

16.9.2.1 SYS_DMA_CHANNELCONFIG 180

16.9.2.2 SYS_DMA_MODE_ENABLE 181

16.9.3 Direct Memory Access Function Documentation 182

16.9.3.1 Sys_DMA_ChannelConfig 182

16.9.3.2 Sys_DMA_Mode_Enable 183

16.9.3.3 Sys_DMA_Get_Status 184

16.9.3.4 Sys_DMA_Clear_Status 184

16.9.3.5 Sys_DMA_Set_Ctrl 185

16.10 Flash Copier 186

16.10.1 Summary 186

16.10.2 Flash Copier Function Documentation 186

16.10.2.1 Sys_Flash_Copy 186

16.10.2.2 Sys_Flash_Compare 187

16.10.2.3 Sys_Flash_CalculateCRC 188

16.11 General-Purpose I/O Interface 189

16.11.1 Summary 189

16.11.2 General-Purpose I/O Interface Macro Definition Documentation 189

16.11.2.1 GPIO_LEVEL1_DRIVE 189

16.11.2.2 GPIO_LEVEL2_DRIVE 190

16.11.2.3 GPIO_LEVEL3_DRIVE 190

16.11.2.4 GPIO_LEVEL4_DRIVE 190

16.11.2.5 SYS_GPIO_CONFIG 190

16.11.3 General-Purpose I/O Interface Function Documentation 191

16.11.3.1 Sys_GPIO_NMIConfig 191

16.11.3.2 Sys_GPIO_IntConfig 192

16.11.3.3 Sys_GPIO_CM33JTAGConfig 192

16.11.3.4 Sys_GPIO_Set_High 193

16.11.3.5 Sys_GPIO_Set_Low 194

16.11.3.6 Sys_GPIO_Toggle 194

16.11.3.7 Sys_GPIO_Read 195

16.11.3.8 Sys_GPIO_Write 196

16.11.3.9 Sys_GPIO_Set_SingleDirection 196

16.11.3.10 Sys_GPIO_Set_Direction 197

16.12 I2C 197

16.12.1 Summary 197

16.12.2 I2C Macro Definition Documentation 198

16.12.2.1 I2C_CONFIG_MASK 198

16.12.2.2 I2C_PADS_NUM 199

16.12.2.3 SYS_I2C_GPIOCONFIG 199

16.12.2.4 SYS_I2C_CONFIG 199

16.12.2.5 SYS_I2C_STARTREAD 200

16.12.2.6 SYS_I2C_STARTWRITE 201

16.12.2.7 SYS_I2C_ACK 201

16.12.2.8 SYS_I2C_NACK 202

16.12.2.9 SYS_I2C_LASTDATA 202

16.12.2.10 SYS_I2C_RESET 203

16.12.2.11 SYS_I2C_NACKANDSTOP 203

16.12.3 I2C Function Documentation 203

16.12.3.1 Sys_I2C_GPIOConfig 204

16.12.3.2 Sys_I2C_Config 204

16.12.3.3 Sys_I2C_StartRead 205

16.12.3.4 Sys_I2C_StartWrite 206

16.12.3.5 Sys_I2C_ACK 206

16.12.3.6 Sys_I2C_NACK 207

16.12.3.7 Sys_I2C_LastData 208

16.12.3.8 Sys_I2C_Reset 208

16.12.3.9 Sys_I2C_NackAndStop 209

16.13 LIN 209

16.13.1 Summary 209

16.13.2 LIN Function Documentation 210

16.13.2.1 Sys_LIN_GPIOConfig 210

16.13.2.2 Sys_LIN_Enable 210

16.13.2.3 Sys_LIN_Disable 211

16.13.2.4 Sys_LIN_ClearErrors 211

16.14 LSAD 212

16.14.1 Summary 212

16.14.2 LSAD Typedef Documentation 214

16.14.2.1 lsad_io_t 214

16.14.2.2 lsad_in_t 214

16.14.2.3 lsad_channel_t 214

16.14.2.4 lsad_prescale_t 215

16.14.2.5 lsad_mode_t 215

16.14.3 LSAD Data Structures Type Documentation 215

16.14.3.1 F_LSAD_TRIM 215

16.14.4 LSAD Enumeration Type Documentation 215

16.14.4.1 lsad_io 215

16.14.4.2 lsad_pol 216

16.14.4.3 lsad_channel 217

16.14.4.4 lsad_prescale 217

16.14.4.5 lsad_mode 218

16.14.5 LSAD Macro Definition Documentation 218

16.14.5.1 LSAD_BIT_RES 218

16.14.5.2 RES_15BIT 219

16.14.5.3 RES_16BIT 219

16.14.5.4 V_REF_DIFF_MV 219

16.14.5.5 RESOLUTION_DIV 219

16.14.5.6 ADC_VAL_MV 219

16.14.5.7 LSAD_OFFSET_ERROR_CONV_QUOTIENT 220

16.14.5.8 LSAD_GAIN_ERROR_CONV_QUOTIENT 220

16.14.5.9 ERROR_LSAD_INPUT_CFG 220

16.14.5.10 PRE_SEL_SIZE 220

16.14.5.11 LSAD_CFG_ARR_LENGTH 220

16.14.5.12 LSAD_CFG_ARR_WIDTH 220

16.14.5.13 GPIO_IDX 220

16.14.5.14 IO_CFG_IDX 221

16.14.5.15 MASK_IDX 221

16.14.5.16 POS_SEL_IDX 221

16.14.5.17 NEG_SEL_IDX 221

16.14.5.18 NO_CFG 221

16.14.5.19 LSAD_SPECIAL_CFG_ARR_LENGTH 221

16.14.5.20 LSAD_SPECIAL_CFG_ARR_WIDTH 221

16.14.5.21 LSAD_OCCUPIED_INPUT_ARR_WIDTH 222

16.14.5.22 USER_CFG_IDX 222

16.14.5.23 POS_SEL_SPEC_IDX 222

16.14.5.24 NEG_SEL_SPEC_IDX 222

16.14.5.25 SEL_IDX_SHIFT_DEF 222

16.14.5.26 LSAD_CH_NUM 222

16.14.5.27 LSAD_INPUTS_NUM 223

16.14.5.28 OCCUPIED 223

16.14.5.29 NOT_OCCUPIED ... 223

16.14.5.30 EVEN_GPIO_NUM ... 223

16.14.5.31 ODD_GPIO_NUM ... 223

16.14.5.32 OCC_STATE_IDX ... 223

16.14.5.33 EVENODD_IDX ... 223

16.14.5.34 INPUT_MASK_IDX ... 224

16.14.5.35 NEG_INPUTSEL_START ... 224

16.14.5.36 POS_INPUTSEL_START ... 224

16.14.6 LSAD Function Documentation ... 224

16.14.6.1 Sys_LSAD_Gain_Offset ... 224

16.14.6.2 Sys_LSAD_TempSensor_Gain_Offset ... 225

16.14.6.3 Sys_LSAD_TrimsInit ... 225

16.14.6.4 Sys_LSAD_ModeConfig ... 226

16.14.6.5 Sys_LSAD_AlarmConfig ... 226

16.14.6.6 Sys_LSAD_InterruptEnable ... 227

16.14.6.7 Sys_LSAD_Start ... 228

16.14.6.8 Sys_LSAD_Stop ... 228

16.14.6.9 Sys_LSAD_InputConfig ... 229

16.14.6.10 Sys_LSAD_Special_InputConfig ... 229

16.14.6.11 Sys_LSAD_PreSelectWrite ... 230

16.14.6.12 Sys_LSAD_GPIO_InputConfig ... 231

16.14.6.13 Sys_LSAD_GetRawData ... 232

16.14.6.14 Sys_LSAD_ConvertToMVUntrimmed ... 232

16.14.6.15 Sys_LSAD_ConvertToMVTrimmed ... 233

16.14.6.16 Sys_LSAD_NewSampleClear ... 234

16.15 Nested Vectored Interrupt Controller ... 234

16.15.1 Summary 234

16.15.2 Nested Vectored Interrupt Controller Function Documentation 234

16.15.2.1 Sys_NVIC_DisableAllInt 234

16.15.2.2 Sys_NVIC_ClearAllPendingInt 235

16.16 Power Supply 235

16.16.1 Summary 235

16.16.2 Power Supply Function Documentation 236

16.16.2.1 Sys_Power_RFEnable 236

16.16.2.2 Sys_Power_CC312_Enable 236

16.16.2.3 Sys_Power_CC312_Disable 236

16.16.2.4 Sys_Power_CC312AO_Enable 237

16.16.2.5 Sys_Power_CC312AO_Disable 237

16.16.2.6 Sys_Power_FPU_Enable 238

16.16.2.7 Sys_Power_FPU_Disable 238

16.17 HAL Power Modes 239

16.17.1 Summary 239

16.17.2 HAL Power Modes Typedef Documentation 240

16.17.2.1 p_application_return 240

16.17.2.2 p_appplication_gpio_config 240

16.17.2.3 sleep_mode_ret_regulator_cfg 241

16.17.2.4 standby_ret_trim_cfg 241

16.17.2.5 clock_cfg 241

16.17.2.6 sleep_mode_cfg 241

16.17.2.7 standby_mode_cfg 241

16.17.2.8 deepsleep_mode_cfg 242

16.17.3 HAL Power Modes Data Structures Type Documentation 242

16.17.3.1 sleep_mode_ret_regulator_cfg_t 242

16.17.3.2 standby_ret_trim_cfg_t 242

16.17.3.3 clock_cfg_t 243

16.17.3.4 sleep_mode_cfg_t 243

16.17.3.5 standby_mode_cfg_t 244

16.17.3.6 deepsleep_mode_cfg_t 245

16.17.4 HAL Power Modes Enumeration Type Documentation 246

16.17.4.1 Sleep_Retention_t 246

16.17.5 HAL Power Modes Macro Definition Documentation 246

16.17.5.1 BLE_PRESENT 246

16.17.5.2 BLE_NOT_PRESENT 247

16.17.5.3 RF_REGISTERS_IMAGE_SIZE_BYTES 247

16.17.5.4 VDDMRETENTION_TRIM_MINIMUM 247

16.17.5.5 VDDMRETENTION_TRIM 247

16.17.5.6 VDDMRETENTION_TRIM_MAXIMUM 247

16.17.5.7 VDDTRETENTION_DISABLE 248

16.17.5.8 VDDTRETENTION_ENABLE 248

16.17.5.9 VDDCRETENTION_TRIM_MINIMUM 248

16.17.5.10 VDDCRETENTION_TRIM 248

16.17.5.11 VDDCRETENTION_TRIM_MAXIMUM 249

16.17.5.12 VDDACSRETENTION_TRIM_MINIMUM 249

16.17.5.13 VDDACSRETENTION_TRIM 249

16.17.5.14 VDDACSRETENTION_TRIM_MAXIMUM 249

16.17.5.15 RSL15_CHIP_ID_SV2 249

16.17.5.16 CHIP_ID_IS_SV2 250

16.17.5.17 CP_ENABLED_IN_SLEEP 250

16.17.5.18 STANDBYCLK_IS_ENABLED 250

16.17.6 HAL Power Modes Function Documentation 250

16.17.6.1 Sys_PowerModes_Sleep_Init 250

16.17.6.2 Sys_PowerModes_Sleep_Enter 251

16.17.6.3 Sys_PowerModes_Standby_Init 252

16.17.6.4 Sys_PowerModes_Standby_Enter 252

16.17.6.5 Sys_PowerModes_DeepSleep_Init 253

16.17.6.6 Sys_PowerModes_DeepSleep_Enter 254

16.17.6.7 Sys_PowerModes_Wakeup_WithReset 254

16.17.6.8 Sys_PowerModes_Wakeup_ConfigEnable 255

16.17.6.9 Sys_PowerModes_Wakeup_ConfigDisable 256

16.18 Pulse-Width Modulation 256

16.18.1 Summary 256

16.18.2 Pulse-Width Modulation Macro Definition Documentation 257

16.18.2.1 PWM_CHANNELS 257

16.18.2.2 HIGH_FRACTIONAL 257

16.18.3 Pulse-Width Modulation Function Documentation 257

16.18.3.1 Sys_PWM_GPIOConfig 257

16.18.3.2 Sys_PWM_Config 258

16.18.3.3 Sys_PWM_Enable 259

16.18.3.4 Sys_PWM_Disable 259

16.18.3.5 Sys_PWM_Reset_Channel 260

16.19 RFFE Radio Frequency Front End 260

16.19.1 Summary 261

16.19.2 RFFE Radio Frequency Front End Macro Definition Documentation 262

16.19.2.1 STABILIZATION_DELAY 262

16.19.2.2 MEASUREMENT_DELAY 262

16.19.2.3 V_TO_MV 262

16.19.2.4 V_TO_MV_F 262

16.19.2.5 DEF_CHANNEL 263

16.19.2.6 MAX_LSAD_CHANNEL 263

16.19.2.7 VDDPA_EN 263

16.19.2.8 VDDPA_DIS 263

16.19.2.9 VCC_VDDRF_MARGIN 263

16.19.2.10 TRIM_MARGIN 264

16.19.2.11 MV_PER_DBM_VDDPA 264

16.19.2.12 MV_PER_DBM_VDDRF 264

16.19.2.13 STEPS_PER_DBM_VDDRF 264

16.19.2.14 STEPS_PER_DBM_VDDPA 265

16.19.2.15 RF_MAX_POWER 265

16.19.2.16 RF_MAX_POWER_NO_VDDPA 265

16.19.2.17 RF_NO_VDDPA_TYPICAL_POWER 265

16.19.2.18 RF_DEFAULT_POWER 265

16.19.2.19 RF_MIN_POWER 266

16.19.2.20 PA_PWR_BYTE_0DBM 266

16.19.2.21 PA_ENABLE_BIAS_SETTING 266

16.19.2.22 PA_DISABLE_BIAS_SETTING 266

16.19.2.23 SW_CTRL_DELAY_3_BYTE 266

16.19.2.24 RAMPUP_DELAY_3_BYTE 267

16.19.2.25 DISABLE_DELAY_3_BYTE 267

16.19.2.26 ERRNO_TX_POWER_MARKER 267

16.19.2.27 ERRNO_NO_TRIMS 267

16.19.2.28 ERRNO_RFFE_MISSINGSETTING_ERROR 267

16.19.2.29 ERRNO_RFFE_INVALIDSETTING_ERROR 268

16.19.2.30 ERRNO_RFFE_VCC_INSUFFICIENT 268

16.19.2.31 WARNING_RFFE_VLOW_POWER_STATE 268

16.19.2.32 WARNING_RFFE_PA_ENABLED_STATE 268

16.19.2.33 CONVERT 268

16.19.2.34 SWAP 269

16.19.2.35 SYS_RFFE_SETTXPOWER 270

16.19.3 RFFE Radio Frequency Front End Function Documentation 271

16.19.3.1 Sys_RFFE_GetTXPower 271

16.19.3.2 Sys_RFFE_SetTXPower 271

16.20 Real-Time Clock 273

16.20.1 Summary 273

16.20.2 Real-Time Clock Function Documentation 273

16.20.2.1 Sys_RTC_Config 273

16.20.2.2 Sys_RTC_Count_Threshold 274

16.20.2.3 Sys_RTC_Value_Seconds 274

16.20.2.4 Sys_RTC_Value 275

16.21 SAR_ADC 275

16.21.1 Summary 275

16.21.2 SAR_ADC Macro Definition Documentation 276

16.21.2.1 VDDSAR_FREQ_THRESHOLD 276

16.21.2.2 MAX_SENSOR_CLK_HIGH 276

16.21.2.3 MAX_SENSOR_CLK_LOW 276

16.21.3 SAR_ADC Function Documentation 276

16.21.3.1 Sys_Calibrate_SARADC 276

16.22 Simple Assertions 278

16.22.1 Summary 278

16.22.2 Simple Assertions Macro Definition Documentation 278

16.22.2.1 SYS_ASSERT 278

16.23 Ultra-Low Power Data Acquisition Subsystem 278

16.23.1 Summary 278

16.23.2 Ultra-Low Power Data Acquisition Subsystem Function Documentation 279

16.23.2.1 Sys_Sensor_SARConfig 279

16.23.2.2 Sys_Sensor_PulseCountConfig 280

16.23.2.3 Sys_Sensor_StorageConfig 281

16.23.2.4 Sys_Sensor_DelayConfig 282

16.23.2.5 Sys_Sensor_Enable 283

16.23.2.6 Sys_Sensor_Disable 284

16.23.2.7 Sys_Sensor_TimerReset 284

16.23.2.8 Sys_Sensor_CurrentState 285

16.23.2.9 Sys_Sensor_CurrentCountValue 285

16.24 SPI 286

16.24.1 Summary 286

16.24.2 SPI Macro Definition Documentation 286

16.24.2.1 SPI_CONFIG_MASK 286

16.24.2.2 SPI_PADS_NUM 287

16.24.2.3 IS_GPIO_REAL 287

16.24.2.4 SYS_SPI_CONFIG 287

16.24.2.5 SYS_SPI_TRANSFERCONFIG 288

16.24.2.6 SYS_SPI_READ 289

16.24.2.7 SYS_SPI_WRITE 289

16.24.2.8 SYS_SPI_GPIOCONFIG 290

16.24.2.9 SYS_DSPI_GPIOCONFIG 291

16.24.2.10 SYS_QSPI_GPIOCONFIG 292

16.24.3 SPI Function Documentation 292

16.24.3.1 Sys_SPI_Config 292

16.24.3.2 Sys_SPI_TransferConfig 293

16.24.3.3 Sys_SPI_Read 294

16.24.3.4 Sys_SPI_Write 294

16.24.3.5 Sys_SPI_GPIOConfig 295

16.24.3.6 Sys_DSPI_GPIOConfig 296

16.24.3.7 Sys_QSPI_GPIOConfig 297

16.25 General-Purpose Timer 298

16.25.1 Summary 298

16.25.2 General-Purpose Timer Macro Definition Documentation 298

16.25.2.1 SYS_TIMER_CONFIG 298

16.25.2.2 SYS_TIMER_START 298

16.25.2.3 SYS_TIMER_STOP 298

16.25.3 General-Purpose Timer Function Documentation 299

16.25.3.1 Sys_Timer_Config 299

16.25.3.2 Sys_Timer_Start 300

16.25.3.3 Sys_Timer_Stop 300

16.26 Time of Flight 301

16.26.1 Summary 301

16.26.2 Time of Flight Function Documentation 301

16.26.2.1 Sys_TOF_Config 301

16.26.2.2 Sys_TOF_Start 302

16.26.2.3 Sys_TOF_Stop 302

16.27 Trimming Support 303

16.27.1 Summary 303

16.27.2 Trimming Support Variable Documentation 305

16.27.2.1 trim_args1 305

16.27.2.2 trim_args2 305

16.27.3 Trimming Support Enumeration Type Documentation 305

16.27.3.1 TrimTarget_t 305

16.27.3.2 TrimName_t 309

16.27.4 Trimming Support Macro Definition Documentation 311

16.27.4.1 NULL_POINTER 311

16.27.4.2 MIN_32_BIT 311

16.27.4.3 MAX_32_BIT 311

16.27.4.4 MIN_18_BIT 311

16.27.4.5 MAX_18_BIT 311

16.27.4.6 MIN_16_BIT 312

16.27.4.7 MAX_16_BIT 312

16.27.4.8 MIN_8_BIT 312

16.27.4.9 MAX_8_BIT 312

16.27.4.10 MAX_4_BIT 312

16.27.4.11 ERROR_NO_ERROR 313

16.27.4.12 ERROR_NULL 313

16.27.4.13 ERROR_NO_TRIM_FOUND 313

16.27.4.14 ERROR_INVALID_TRIM 313

16.27.4.15 ERROR_INVALID_CRC 313

16.27.4.16 ERROR_BG_INVALID 314

16.27.4.17 ERROR_BG_V_INVALID 314

16.27.4.18 ERROR_BG_I_INVALID 314

16.27.4.19 ERROR_DCDC_INVALID 314

16.27.4.20 ERROR_VDDC_INVALID 315

16.27.4.21 ERROR_VDDC_STBY_INVALID 315

16.27.4.22 ERROR_VDDM_INVALID 315

16.27.4.23 ERROR_VDDM_STBY_INVALID 315

16.27.4.24 ERROR_VDDRF_INVALID 315

16.27.4.25 ERROR_VDDPA_INVALID 316

16.27.4.26 ERROR_VDDPA_MIN_INVALID 316

16.27.4.27 ERROR_VDDIF_INVALID 316

16.27.4.28 ERROR_VDDFLASH_INVALID 316

16.27.4.29 ERROR_RCOSC_INVALID 316

16.27.4.30 ERROR_RCOSC32_INVALID 317

16.27.4.31 ERROR_LSAD_INVALID 317

16.27.4.32 ERROR_TEMPERATURE_INVALID 317

16.27.4.33 ERROR_THERMISTOR_INVALID 317

16.27.4.34 ERROR_MEASURED_INVALID 317

16.27.4.35 ERROR_TRIM_CUSTOM_SIGNATURE_INVALID 318

16.27.4.36 ERROR_TRIM_CUSTOM_ICH_INVALID 318

16.27.4.37 ERROR_TRIM_CUSTOM_XTAL_INVALID 318

16.27.4.38 TR_REG_TRIM_MASK 318

16.27.4.39 TRIM_8_BIT_TRIM_MASK 318

16.27.4.40 TRIM_16_BIT_TRIM_MASK 319

16.27.4.41 TRIM_NVR7_2_BIT_RC_FSEL_MASK 319

16.27.4.42 TRIM_NVR7_VCC_DCDC_Pos 319

16.27.4.43 LSAD_HF 319

16.27.4.44 LSAD_LF 319

16.27.4.45 LSAD_OFFSET 320

16.27.4.46 LSAD_OFFSET_MASK 320

16.27.4.47 LSAD_GAIN 320

16.27.4.48 LSAD_GAIN_MASK 320

16.27.4.49 TRIM 320

16.27.4.50 TRIM_SUPPLEMENTAL 321

16.27.4.51 TRIM_CUSTOM_SIP1_SIGNATURE 321

16.27.4.52 TRIM_CUSTOM_CUST_SIGNATURE 321

16.27.4.53 ICH_TRIM_DEFAULT 321

16.27.4.54 SYS_TRIM_LOAD_DEFAULT 321

16.27.4.55 SYS_TRIM_LOAD_SUPPLEMENTAL 322

16.27.4.56 SYS_TRIM_LOAD_CUSTOM 323

16.27.5 Trimming Support Function Documentation 323

16.27.5.1 Sys_Trim_LoadTrims 323

16.27.5.2 Sys_Trim_LoadSingleTrim 324

16.27.5.3 Sys_Trim_VerifyTrims 326

16.27.5.4 Sys_Trim_CheckCRC 326

16.27.5.5 Sys_Trim_GetTrim 327

16.27.5.6 Sys_Trim_LoadBandgap 328

16.27.5.7 Sys_Trim_LoadDCDC 329

16.27.5.8 Sys_Trim_LoadVDDC 329

16.27.5.9 Sys_Trim_LoadVDDM 330

16.27.5.10 Sys_Trim_LoadVDDPA 331

16.27.5.11 Sys_Trim_LoadVDDRF 332

16.27.5.12 Sys_Trim_LoadCustom 332

16.27.5.13 Sys_Trim_LoadVDDFLASH 333

16.27.5.14 Sys_Trim_LoadRCOSC 334

16.27.5.15 Sys_Trim_LoadRCOSC32 334

16.27.5.16 Sys_Trim_LoadThermistor 335

16.27.5.17 Sys_Trim_GetLSADTrim 336

16.28 UART 337

16.28.1 Summary 337

16.28.2 UART Macro Definition Documentation 337

16.28.2.1 UART_PADS_NUM 337

16.28.2.2 SYS_UART_GPIOCONFIG 337

16.28.2.3 SYS_UART_CONFIG 338

16.28.3 UART Function Documentation 339

16.28.3.1 Sys_UART_GPIOConfig 340

16.28.3.2 Sys_UART_Config 340

16.29 Watchdog 341

16.29.1 Summary 341

16.29.2 Watchdog Macro Definition Documentation 341

16.29.2.1 SYS_WATCHDOG_REFRESH 341

17. Flash Library Reference 343

17.1 Summary 343

17.2 Detailed Description 343

17.3 Flash Library Reference Variable Documentation 344

17.3.1 FlashLib_Version 344

17.4 Flash Library Reference Macro Definition Documentation 344

17.4.1 FLASH_FW_VER_MAJOR 344

17.4.2 FLASH_FW_VER_MINOR 344

17.4.3 FLASH_FW_VER_REVISION 344

17.4.4 FLASH_FW_VER 345

17.4.5 FLASH0 345

17.4.6 FLASH_INSTANCE_NUM 345

17.4.7 FLASH_0_DESCR_NUM 345

17.4.8 RSL15_284_FLASH_SET 345

17.5 Flash Library Reference Function Documentation 346

17.5.1 Flash_Initialize 346

17.5.2 Flash_WriteWord 347

17.5.3 Flash_WriteBuffer 348

17.5.4 Flash_WriteDouble 349

17.5.5 Flash_ReadWord 351

17.5.6 Flash_ReadBuffer 351

17.5.7 Flash_ReadDouble 352

17.5.8 Flash_EraseFlashBank 354

17.5.9 Flash_EraseChip 355

17.5.10 Flash_EraseSector 355

17.5.11 Flash_BlankCheck 356

18. CMSIS Drivers Reference 358

18.1 Summary 358

18.2 CMSIS Drivers Reference Typedef Documentation 358

18.2.1 ARM_DRIVER_VERSION 358

18.2.2 ARM_POWER_STATE 359

18.3 CMSIS Drivers Reference Data Structures Type Documentation 359

18.3.1 _ARM_DRIVER_VERSION 359

18.4 CMSIS Drivers Reference Enumeration Type Documentation 359

18.4.1 _ARM_POWER_STATE 359

18.5 CMSIS Drivers Reference Macro Definition Documentation 360

18.5.1 ARM_DRIVER_VERSION_MAJOR_MINOR 360

18.5.2 ARM_DRIVER_OK 360

18.5.3 ARM_DRIVER_ERROR 360

18.5.4 ARM_DRIVER_ERROR_BUSY 361

18.5.5 ARM_DRIVER_ERROR_TIMEOUT 361

18.5.6 ARM_DRIVER_ERROR_UNSUPPORTED 361

18.5.7 ARM_DRIVER_ERROR_PARAMETER 361

18.5.8 ARM_DRIVER_ERROR_SPECIFIC 362

18.6 CMSIS DMA Driver 362

18.6.1 Summary 362

18.6.2 CMSIS DMA Driver Typedef Documentation 363

18.6.2.1 DMA_SEL_t 363

18.6.2.2 DMA_TRG_t 364

18.6.2.3 DMA_SRC_STEP_t 364

18.6.2.4 DMA_DST_STEP_t 364

18.6.2.5 DMA_SRC_DST_TRANS_LENGHT_SEL_t 364

18.6.2.6 DMA_DATA_MODE_t 364

18.6.2.7 DMA_BYTE_ORDER_t 365

18.6.2.8 DMA_WORD_SIZE_t 365

18.6.2.9 DMA_CH_PRI_t 365

18.6.2.10 ADC_EVENT_SRC_t 365

18.6.2.11 DMA_SignalEvent_t 365

18.6.2.12 DMA_CFG_t 366

18.6.2.13 DMA_ADDR_CFG_t 366

18.6.2.14 DMA_PRI_CFG_t 366

18.6.2.15 DMA_STATUS_t 366

18.6.2.16 DRIVER_DMA_t 366

18.6.3 CMSIS DMA Driver Data Structures Type Documentation 367

18.6.3.1 _DMA_CFG_t 367

18.6.3.2 _DMA_ADDR_CFG_t 367

18.6.3.3 _DMA_PRI_CFG_t 368

18.6.3.4 _DMA_STATUS_t 368

18.6.3.5 _DRIVER_DMA_t 369

18.6.4 CMSIS DMA Driver Enumeration Type Documentation 370

18.6.4.1 _DMA_SEL_t 370

18.6.4.2 _DMA_TRG_t 371

18.6.4.3 _DMA_SRC_STEP_t 372

18.6.4.4 _DMA_DST_STEP_t 375

18.6.4.5 _DMA_SRC_DST_TRANS_LENGHT_SEL_t 377

18.6.4.6 _DMA_DATA_MODE_t 378

18.6.4.7 _DMA_BYTE_ORDER_t 379

18.6.4.8 _DMA_WORD_SIZE_t 379

18.6.4.9 _DMA_CH_PRI_t 383

18.6.4.10 _ADC_EVENT_SRC_t 384

18.6.5 CMSIS DMA Driver Macro Definition Documentation 385

18.6.5.1 ARM_DMA_API_VERSION 385

18.6.5.2 DMA_ERROR_UNCONFIGURED 385

18.6.6 CMSIS DMA Driver Function Documentation 385

18.6.6.1 DMA_GetVersion 385

18.6.6.2 DMA_Initialize 386

18.6.6.3 DMA_Configure 386

18.6.6.4 DMA_ConfigureWord 387

18.6.6.5 DMA_ConfigureAddr 387

18.6.6.6 DMA_SetInterruptPriority 388

18.6.6.7 DMA_CreateConfigWord 388

18.6.6.8 DMA_SetConfigWord 389

18.6.6.9 DMA_Stop 389

18.6.6.10 DMA_GetCounterValue 390

18.6.6.11 DMA_GetStatus 390

18.6.6.12 DMA_SignalEvent 391

18.7 CMSIS GPIO Driver 391

18.7.1 Summary 391

18.7.2 CMSIS GPIO Driver Typedef Documentation 393

18.7.2.1 GPIO_SEL_t 393

18.7.2.2 GPIO_DIR_t 393

18.7.2.3 GPIO_INT_SEL_t 394

18.7.2.4 GPIO_DRIVE_t 394

18.7.2.5 GPIO_LPF_t 394

18.7.2.6 GPIO_PULL_t 394

18.7.2.7 GPIO_OUTPUT_LEVEL_t 394

18.7.2.8 GPIO_MODE_t 395

18.7.2.9 GPIO_FUNC_REGISTERS_t 395

18.7.2.10 GPIO_EN_DIS_t 395

18.7.2.11 GPIO_EVENT_t 395

18.7.2.12 GPIO_DBC_CLK_t 395

18.7.2.13 GPIO_DRIVE_STRENGTHS_t 396

18.7.2.14 GPIO_SignalEvent_t 396

18.7.2.15 GPIO_DBF_CFG_t 396

18.7.2.16 GPIO_PRI_CFG_t 396

18.7.2.17 GPIO_CFG_t 397

18.7.2.18 GPIO_PAD_CFG_t 397

18.7.2.19 GPIO_INT_CFG_t 397

18.7.2.20 GPIO_EXTCLK_CFG_t 397

18.7.2.21 GPIO_JTAG_SW_CFG_t 397

18.7.2.22 DRIVER_GPIO_t 398

18.7.3 CMSIS GPIO Driver Data Structures Type Documentation 398

18.7.3.1 _GPIO_DBF_CFG_t 398

18.7.3.2 _GPIO_PRI_CFG_t 398

18.7.3.3 _GPIO_CFG_t 399

18.7.3.4 _GPIO_PAD_CFG_t 399

18.7.3.5 _GPIO_INT_CFG_t 400

18.7.3.6 _GPIO_EXTCLK_CFG_t 400

18.7.3.7 _GPIO_JTAG_SW_CFG_t 401

18.7.3.8 _DRIVER_GPIO_t 401

18.7.4 CMSIS GPIO Driver Enumeration Type Documentation 402

18.7.4.1 _GPIO_SEL_t 402

18.7.4.2 _GPIO_DIR_t 404

18.7.4.3 _GPIO_INT_SEL_t 407

18.7.4.4 _GPIO_DRIVE_t 407

18.7.4.5 _GPIO_LPF_t 408

18.7.4.6 _GPIO_PULL_t 409

18.7.4.7 _GPIO_OUTPUT_LEVEL_t 409

18.7.4.8 _GPIO_MODE_t 410

18.7.4.9 _GPIO_FUNC_REGISTERS_t 415

18.7.4.10 _GPIO_EN_DIS_t 417

18.7.4.11 _GPIO_EVENT_t 417

18.7.4.12 _GPIO_DBC_CLK_t 418

18.7.4.13 _GPIO_DRIVE_STRENGTHS_t 419

18.7.5 CMSIS GPIO Driver Macro Definition Documentation 419

18.7.5.1 ARM_GPIO_API_VERSION 419

18.7.5.2 GPIO_EVENT_0_IRQ 420

18.7.5.3 GPIO_EVENT_1_IRQ 420

18.7.5.4 GPIO_EVENT_2_IRQ 420

18.7.5.5 GPIO_EVENT_3_IRQ 420

18.7.6 CMSIS GPIO Driver Function Documentation 420

18.7.6.1 GPIO_GetVersion 420

18.7.6.2 GPIO_Initialize 421

18.7.6.3 GPIO_Configure 421

18.7.6.4 GPIO_ConfigurePad 422

18.7.6.5 GPIO_ConfigureInterrupt 422

18.7.6.6 GPIO_SetInterruptPriority 423

18.7.6.7 GPIO_ConfigureJTAG 423

18.7.6.8 GPIO_SetHigh 424

18.7.6.9 GPIO_ToggleValue 424

18.7.6.10 GPIO_SetLow 425

18.7.6.11 GPIO_ReadValue 425

18.7.6.12 GPIO_ResetAltFuncRegister 426

18.7.6.13 GPIO_SignalEvent 426

18.8 CMSIS I2C Driver 427

18.8.1 Summary 427

18.8.2 CMSIS I2C Driver Typedef Documentation 428

18.8.2.1 ARM_I2C_STATUS 428

18.8.2.2 ARM_I2C_SignalEvent_t 428

18.8.2.3 ARM_I2C_CAPABILITIES 429

18.8.2.4 ARM_DRIVER_I2C 429

18.8.3 CMSIS I2C Driver Data Structures Type Documentation 429

18.8.3.1 _ARM_I2C_STATUS 429

18.8.3.2 _ARM_I2C_CAPABILITIES 430

18.8.3.3 _ARM_DRIVER_I2C 430

18.8.4 CMSIS I2C Driver Macro Definition Documentation 431

18.8.4.1 ARM_I2C_API_VERSION 431

18.8.4.2 ARM_I2C_OWN_ADDRESS 431

18.8.4.3 ARM_I2C_BUS_SPEED 432

18.8.4.4 ARM_I2C_BUS_CLEAR 432

18.8.4.5 ARM_I2C_ABORT_TRANSFER 432

18.8.4.6 ARM_I2C_BUS_SPEED_STANDARD 432

18.8.4.7 ARM_I2C_BUS_SPEED_FAST 432

18.8.4.8 ARM_I2C_BUS_SPEED_FAST_PLUS 433

18.8.4.9 ARM_I2C_BUS_SPEED_HIGH 433

18.8.4.10 ARM_I2C_ADDRESS_10BIT 433

18.8.4.11 ARM_I2C_ADDRESS_GC 433

18.8.4.12 ARM_I2C_EVENT_TRANSFER_DONE 434

18.8.4.13 ARM_I2C_EVENT_TRANSFER_INCOMPLETE 434

18.8.4.14 ARM_I2C_EVENT_SLAVE_TRANSMIT 434

18.8.4.15 ARM_I2C_EVENT_SLAVE_RECEIVE 434

18.8.4.16 ARM_I2C_EVENT_ADDRESS_NACK 434

18.8.4.17 ARM_I2C_EVENT_GENERAL_CALL 435

18.8.4.18 ARM_I2C_EVENT_ARBITRATION_LOST 435

18.8.4.19 ARM_I2C_EVENT_BUS_ERROR 435

18.8.4.20 ARM_I2C_EVENT_BUS_CLEAR 435

18.8.5 CMSIS I2C Driver Function Documentation 435

18.8.5.1 ARM_I2C_GetVersion 436

18.8.5.2 ARM_I2C_GetCapabilities 436

18.8.5.3 ARM_I2C_Initialize 436

18.8.5.4 ARM_I2C_Uninitialize 437

18.8.5.5 ARM_I2C_PowerControl 437

18.8.5.6 ARM_I2C_MasterTransmit 438

18.8.5.7 ARM_I2C_MasterReceive 438

18.8.5.8 ARM_I2C_SlaveTransmit 439

18.8.5.9 ARM_I2C_SlaveReceive 439

18.8.5.10 ARM_I2C_GetDataCount 440

18.8.5.11 ARM_I2C_Control 440

18.8.5.12 ARM_I2C_GetStatus 441

18.8.5.13 ARM_I2C_SignalEvent 441

18.9 CMSIS PWM Driver 441

18.9.1 Summary 442

18.9.2 CMSIS PWM Driver Typedef Documentation 442

18.9.2.1 PWM_SEL_t 443

18.9.2.2 PWM_CFG_t 443

18.9.2.3 DRIVER_PWM_t 443

18.9.3 CMSIS PWM Driver Data Structures Type Documentation 443

18.9.3.1 _PWM_CFG_t 443

18.9.3.2 _DRIVER_PWM_t 444

18.9.4 CMSIS PWM Driver Enumeration Type Documentation 445

18.9.4.1 _PWM_SEL_t 445

18.9.5 CMSIS PWM Driver Macro Definition Documentation 446

18.9.5.1 ARM_PWM_API_VERSION 446

18.9.5.2 PWM_ERROR_UNCONFIGURED 446

18.9.6 CMSIS PWM Driver Function Documentation 446

18.9.6.1 PWM_GetVersion 446

18.9.6.2 PWM_Initialize 447

18.9.6.3 PWM_Configure 447

18.9.6.4 PWM_SelectClock 447

18.9.6.5 PWM_Reset 448

18.9.6.6 PWM_SetDithering 449

18.9.6.7 PWM_SetPeriod 449

18.9.6.8 PWM_SetDutyCycle 450

18.9.6.9 PWM_SetHighPeriod 450

18.9.6.10 PWM_SetOffset 451

18.9.6.11 PWM_Start 451

18.9.6.12 PWM_Stop 452

18.10 CMSIS SPI Driver 452

18.10.1 Summary 452

18.10.2 CMSIS SPI Driver Typedef Documentation 454

18.10.2.1 ARM_SPI_STATUS 454

18.10.2.2 ARM_SPI_SignalEvent_t 455

18.10.2.3 ARM_SPI_CAPABILITIES 455

18.10.2.4 ARM_DRIVER_SPI 455

18.10.3 CMSIS SPI Driver Data Structures Type Documentation 455

18.10.3.1 _ARM_SPI_STATUS 455

18.10.3.2 _ARM_SPI_CAPABILITIES 456

18.10.3.3 _ARM_DRIVER_SPI 456

18.10.4 CMSIS SPI Driver Macro Definition Documentation 457

18.10.4.1 ARM_SPI_API_VERSION 457

18.10.4.2 ARM_SPI_CONTROL_Pos 457

18.10.4.3 ARM_SPI_CONTROL_Msk 458

18.10.4.4 ARM_SPI_MODE_INACTIVE 458

18.10.4.5 ARM_SPI_MODE_MASTER 458

18.10.4.6 ARM_SPI_MODE_SLAVE 458

18.10.4.7 ARM_SPI_MODE_MASTER_SIMPLEX 458

18.10.4.8 ARM_SPI_MODE_SLAVE_SIMPLEX 459

18.10.4.9 ARM_SPI_FRAME_FORMAT_Pos 459

18.10.4.10 ARM_SPI_FRAME_FORMAT_Msk 459

18.10.4.11 ARM_SPI_CPOL0_CPHA0 459

18.10.4.12 ARM_SPI_CPOL0_CPHA1 460

18.10.4.13 ARM_SPI_CPOL1_CPHA0 460

18.10.4.14 ARM_SPI_CPOL1_CPHA1 460

18.10.4.15 ARM_SPI_TI_SSI 460

18.10.4.16 ARM_SPI_MICROWIRE 460

18.10.4.17 ARM_SPI_DATA_BITS_Pos 461

18.10.4.18 ARM_SPI_DATA_BITS_Msk 461

18.10.4.19 ARM_SPI_DATA_BITS 461

18.10.4.20 ARM_SPI_BIT_ORDER_Pos 461

18.10.4.21 ARM_SPI_BIT_ORDER_Msk 461

18.10.4.22 ARM_SPI_MSB_LSB 462

18.10.4.23 ARM_SPI_LSB_MSB 462

18.10.4.24 ARM_SPI_SS_MASTER_MODE_Pos 462

18.10.4.25 ARM_SPI_SS_MASTER_MODE_Msk 462

18.10.4.26 ARM_SPI_SS_MASTER_UNUSED 462

18.10.4.27 ARM_SPI_SS_MASTER_SW 463

18.10.4.28 ARM_SPI_SS_MASTER_HW_OUTPUT 463

18.10.4.29 ARM_SPI_SS_MASTER_HW_INPUT 463

18.10.4.30 ARM_SPI_SS_SLAVE_MODE_Pos 463

18.10.4.31 ARM_SPI_SS_SLAVE_MODE_Msk 463

18.10.4.32 ARM_SPI_SS_SLAVE_HW 464

18.10.4.33 ARM_SPI_SS_SLAVE_SW 464

18.10.4.34 ARM_SPI_SET_BUS_SPEED 464

18.10.4.35 ARM_SPI_GET_BUS_SPEED 464

18.10.4.36 ARM_SPI_SET_DEFAULT_TX_VALUE 464

18.10.4.37 ARM_SPI_CONTROL_SS 465

18.10.4.38 ARM_SPI_ABORT_TRANSFER 465

18.10.4.39 ARM_SPI_SS_INACTIVE 465

18.10.4.40 ARM_SPI_SS_ACTIVE 465

18.10.4.41 ARM_SPI_ERROR_MODE 466

18.10.4.42 ARM_SPI_ERROR_FRAME_FORMAT 466

18.10.4.43 ARM_SPI_ERROR_DATA_BITS 466

18.10.4.44 ARM_SPI_ERROR_BIT_ORDER 466

18.10.4.45 ARM_SPI_ERROR_SS_MODE 466

18.10.4.46 ARM_SPI_EVENT_TRANSFER_COMPLETE 467

18.10.4.47 ARM_SPI_EVENT_DATA_LOST 467

18.10.4.48 ARM_SPI_EVENT_MODE_FAULT 467

18.10.5 CMSIS SPI Driver Function Documentation 467

18.10.5.1 ARM_SPI_GetVersion 467

18.10.5.2 ARM_SPI_GetCapabilities 468

18.10.5.3 ARM_SPI_Initialize 468

18.10.5.4 ARM_SPI_Uninitialize 468

18.10.5.5 ARM_SPI_PowerControl 469

18.10.5.6 ARM_SPI_Send 469

18.10.5.7 ARM_SPI_Receive 470

18.10.5.8 ARM_SPI_Transfer 470

18.10.5.9 ARM_SPI_GetDataCount 471

18.10.5.10 ARM_SPI_Control 471

18.10.5.11 ARM_SPI_GetStatus 472

18.10.5.12 ARM_SPI_SignalEvent 472

18.11 CMSIS Timer Driver 473

18.11.1 Summary 473

18.11.2 CMSIS Timer Driver Typedef Documentation 474

18.11.2.1 TIMER_SEL_t 474

18.11.2.2 TIMER_MODE_t 474

18.11.2.3 TIMER_CLKSRC_t 475

18.11.2.4 TIMER_PRESCALE_t 475

18.11.2.5 TIMER_MULTI_COUNT_t 475

18.11.2.6 TIMER_GPIO_STATUS_t 475

18.11.2.7 TIMER_GPIO_INT_MODE_t 475

18.11.2.8 TIMER_GPIO_t 476

18.11.2.9 TIMER_SYSTICK_CLKSRC_t 476

18.11.2.10 ADC_EVENT_t 476

18.11.2.11 TIMER_SignalEvent_t 476

18.11.2.12 TIMER_t 476

18.11.2.13 SYSTICK_t 477

18.11.2.14 TIMER_CFG_t 477

18.11.2.15 TIMER_PRI_CFG_t 477

18.11.2.16 DRIVER_TIMER_t 477

18.11.3 CMSIS Timer Driver Data Structures Type Documentation 478

18.11.3.1 _TIMER_t 478

18.11.3.2 _SYSTICK_t 478

18.11.3.3 _TIMER_PRI_CFG_t 479

18.11.3.4 _DRIVER_TIMER_t 479

18.11.4 CMSIS Timer Driver Enumeration Type Documentation 480

18.11.4.1 _TIMER_SEL_t 480

18.11.4.2 _TIMER_MODE_t 481

18.11.4.3 _TIMER_CLKSRC_t 482

18.11.4.4 _TIMER_PRESCALE_t 483

18.11.4.5 _TIMER_MULTI_COUNT_t 484

18.11.4.6 _TIMER_GPIO_STATUS_t 486

18.11.4.7 _TIMER_GPIO_INT_MODE_t 486

18.11.4.8 _TIMER_GPIO_t 487

18.11.4.9 _TIMER_SYSTICK_CLKSRC_t 487

18.11.4.10 _ADC_EVENT_t 488

18.11.5 CMSIS Timer Driver Macro Definition Documentation 489

18.11.5.1 ARM_TIMER_API_VERSION 489

18.11.5.2 TIMER_ERROR_UNCONFIGURED 490

18.11.6 CMSIS Timer Driver Function Documentation 490

18.11.6.1 TIMER_GetVersion 490

18.11.6.2 TIMER_Initialize 490

18.11.6.3 TIMER_Configure 491

18.11.6.4 TIMER_SetInterruptPriority 491

18.11.6.5 TIMER_Start 492

18.11.6.6 TIMER_Stop 492

18.11.6.7 TIMER_SetValue 493

18.11.6.8 TIMER_GetValue 493

18.11.6.9 TIMER_GetValueCapture 494

18.11.6.10 TIMER_GetSysTickState 494

18.11.6.11 TIMER_SignalEvent 495

18.11.6.12 TIMER_SetGPIOInterrupt 495

18.12 CMSIS USART Driver 496

18.12.1 Summary 496

18.12.2 CMSIS USART Driver Typedef Documentation 499

18.12.2.1 ARM_USART_STATUS 499

18.12.2.2 ARM_USART_MODEM_CONTROL 499

18.12.2.3 ARM_USART_MODEM_STATUS 499

18.12.2.4 ARM_USART_SignalEvent_t 499

18.12.2.5 ARM_USART_CAPABILITIES 499

18.12.2.6 ARM_DRIVER_USART 500

18.12.3 CMSIS USART Driver Data Structures Type Documentation 500

18.12.3.1 _ARM_USART_STATUS 500

18.12.3.2 _ARM_USART_MODEM_STATUS 501

18.12.3.3 _ARM_USART_CAPABILITIES 501

18.12.3.4 _ARM_DRIVER_USART 502

18.12.4 CMSIS USART Driver Enumeration Type Documentation 503

18.12.4.1 _ARM_USART_MODEM_CONTROL 503

18.12.5 CMSIS USART Driver Macro Definition Documentation 504

18.12.5.1 ARM_USART_API_VERSION 504

18.12.5.2 ARM_USART_CONTROL_Pos 504

18.12.5.3 ARM_USART_CONTROL_Msk 505

18.12.5.4 ARM_USART_MODE_ASYNCHRONOUS 505

18.12.5.5 ARM_USART_MODE_SYNCHRONOUS_MASTER 505

18.12.5.6 ARM_USART_MODE_SYNCHRONOUS_SLAVE 505

18.12.5.7 ARM_USART_MODE_SINGLE_WIRE 505

18.12.5.8 ARM_USART_MODE_IRDA 506

18.12.5.9 ARM_USART_MODE_SMART_CARD 506

18.12.5.10 ARM_USART_DATA_BITS_Pos 506

18.12.5.11 ARM_USART_DATA_BITS_Msk 506

18.12.5.12 ARM_USART_DATA_BITS_5 507

18.12.5.13 ARM_USART_DATA_BITS_6 507

18.12.5.14 ARM_USART_DATA_BITS_7 507

18.12.5.15 ARM_USART_DATA_BITS_8 507

18.12.5.16 ARM_USART_DATA_BITS_9 507

18.12.5.17 ARM_USART_PARITY_Pos 508

18.12.5.18 ARM_USART_PARITY_Msk 508

18.12.5.19 ARM_USART_PARITY_NONE 508

18.12.5.20 ARM_USART_PARITY_EVEN 508

18.12.5.21 ARM_USART_PARITY_ODD 508

18.12.5.22 ARM_USART_STOP_BITS_Pos 509

18.12.5.23 ARM_USART_STOP_BITS_Msk 509

18.12.5.24 ARM_USART_STOP_BITS_1 509

18.12.5.25 ARM_USART_STOP_BITS_2 509

18.12.5.26 ARM_USART_STOP_BITS_1_5 509

18.12.5.27 ARM_USART_STOP_BITS_0_5 510

18.12.5.28 ARM_USART_FLOW_CONTROL_Pos 510

18.12.5.29 ARM_USART_FLOW_CONTROL_Msk 510

18.12.5.30 ARM_USART_FLOW_CONTROL_NONE 510

18.12.5.31 ARM_USART_FLOW_CONTROL_RTS 510

18.12.5.32 ARM_USART_FLOW_CONTROL_CTS 511

18.12.5.33 ARM_USART_FLOW_CONTROL_RTS_CTS 511

18.12.5.34 ARM_USART_CPOL_Pos 511

18.12.5.35 ARM_USART_CPOL_Msk 511

18.12.5.36 ARM_USART_CPOL0 511

18.12.5.37 ARM_USART_CPOL1 512

18.12.5.38 ARM_USART_CPHA_Pos 512

18.12.5.39 ARM_USART_CPHA_Msk 512

18.12.5.40 ARM_USART_CPHA0 512

18.12.5.41 ARM_USART_CPHA1 513

18.12.5.42 ARM_USART_SET_DEFAULT_TX_VALUE 513

18.12.5.43 ARM_USART_SET_IRDA_PULSE 513

18.12.5.44 ARM_USART_SET_SMART_CARD_GUARD_TIME 513

18.12.5.45 ARM_USART_SET_SMART_CARD_CLOCK 513

18.12.5.46 ARM_USART_CONTROL_SMART_CARD_NACK 514

18.12.5.47 ARM_USART_CONTROL_TX 514

18.12.5.48 ARM_USART_CONTROL_RX 514

18.12.5.49 ARM_USART_CONTROL_BREAK 514

18.12.5.50 ARM_USART_ABORT_SEND 514

18.12.5.51 ARM_USART_ABORT_RECEIVE 515

18.12.5.52 ARM_USART_ABORT_TRANSFER 515

18.12.5.53 ARM_USART_ERROR_MODE 515

18.12.5.54 ARM_USART_ERROR_BAUDRATE 515

18.12.5.55 ARM_USART_ERROR_DATA_BITS 515

18.12.5.56 ARM_USART_ERROR_PARITY 516

18.12.5.57 ARM_USART_ERROR_STOP_BITS 516

18.12.5.58 ARM_USART_ERROR_FLOW_CONTROL 516

18.12.5.59 ARM_USART_ERROR_CPOL 516

18.12.5.60 ARM_USART_ERROR_CPHA 516

18.12.5.61 ARM_USART_EVENT_SEND_COMPLETE 517

18.12.5.62 ARM_USART_EVENT_RECEIVE_COMPLETE 517

18.12.5.63 ARM_USART_EVENT_TRANSFER_COMPLETE 517

18.12.5.64 ARM_USART_EVENT_TX_COMPLETE 517

18.12.5.65 ARM_USART_EVENT_TX_UNDERFLOW 518

18.12.5.66 ARM_USART_EVENT_RX_OVERFLOW 518

18.12.5.67 ARM_USART_EVENT_RX_TIMEOUT 518

18.12.5.68 ARM_USART_EVENT_RX_BREAK 518

18.12.5.69 ARM_USART_EVENT_RX_FRAMING_ERROR 518

18.12.5.70 ARM_USART_EVENT_RX_PARITY_ERROR 519

18.12.5.71 ARM_USART_EVENT_CTS 519

18.12.5.72 ARM_USART_EVENT_DSR ... 519

18.12.5.73 ARM_USART_EVENT_DCD ... 519

18.12.5.74 ARM_USART_EVENT_RI ... 519

18.12.6 CMSIS USART Driver Function Documentation ... 520

18.12.6.1 ARM_USART_GetVersion ... 520

18.12.6.2 ARM_USART_GetCapabilities ... 520

18.12.6.3 ARM_USART_Initialize ... 520

18.12.6.4 ARM_USART_Uninitialize ... 521

18.12.6.5 ARM_USART_PowerControl ... 521

18.12.6.6 ARM_USART_Send ... 522

18.12.6.7 ARM_USART_Receive ... 522

18.12.6.8 ARM_USART_Transfer ... 523

18.12.6.9 ARM_USART_GetTxCount ... 523

18.12.6.10 ARM_USART_GetRxCount ... 524

18.12.6.11 ARM_USART_Control ... 524

18.12.6.12 ARM_USART_GetStatus ... 525

18.12.6.13 ARM_USART_SetModemControl ... 525

18.12.6.14 ARM_USART_GetModemStatus ... 525

18.12.6.15 ARM_USART_SignalEvent ... 526

19. swmTrace Reference ... 527

19.1 Summary ... 527

19.2 swmTrace Reference Macro Definition Documentation ... 527

19.2.1 swmLogVerbose ... 527

19.2.2 swmLogInfo ... 527

19.2.3 swmLogWarn ... 528

19.2.4 swmLogError ... 528

19.2.5 swmLogFatal ... 528

19.2.6 swmLogTestPass ... 528

19.2.7 swmLogTestFail ... 529

19.3 swmTrace Reference Function Documentation ... 529

19.3.1 swmTrace_init ... 529

19.3.2 swmTrace_txInProgress ... 530

19.3.3 swmTrace_printf ... 530

19.3.4 swmTrace_vprintf ... 530

19.3.5 swmTrace_getch ... 531

19.3.6 swmLog ... 531

20. Bluetooth Low Energy Stack Abstraction ... 533

20.1 Summary ... 533

20.2 Detailed Description ... 538

20.3 Bluetooth Low Energy Stack Abstraction Typedef Documentation ... 538

20.3.1 MsgHandlerCallback_t ... 538

20.4 Bluetooth Low Energy Stack Abstraction Variable Documentation ... 538

20.4.1 ble_dev_params ... 538

20.5 Bluetooth Low Energy Stack Abstraction Data Structures Type Documentation ... 539

20.5.1 GAPM_ActivityStatus_t ... 539

20.5.2 GAP_Env_t ... 539

20.5.3 att_db_desc ... 540

20.5.4 cust_svc_desc ... 541

20.5.5 GATT_Env_t ... 541

20.5.6 low_power_clock ... 542

20.5.7 ble_device_parameter ... 542

20.5.8 BASC_Env_t ... 542

20.5.9 BASS_Env_t 543

20.5.10 DISS_DeviceInfoField_t 544

20.5.11 DISS_DeviceInfo_t 545

20.5.12 DISS_Env_t 545

20.6 Bluetooth Low Energy Stack Abstraction Enumeration Type Documentation 546

20.6.1 gapm_activity_state 546

20.6.2 gapm_state 547

20.6.3 cfm_msg_instr_enum 547

20.6.4 basc_app_msg_id 548

20.6.5 bass_app_msg_id 548

20.7 Bluetooth Low Energy Stack Abstraction Macro Definition Documentation 549

20.7.1 GAPM_CFG_ADDR_PUBLIC 549

20.7.2 GAPM_CFG_ADDR_PRIVATE 549

20.7.3 GAPM_CFG_HOST_PRIVACY 549

20.7.4 GAPM_CFG_CONTROLLER_PRIVACY 550

20.7.5 GAPM_DEFAULT_ROLE 550

20.7.6 GAPM_DEFAULT_RENEW_DUR 550

20.7.7 GAPM_DEFAULT_GAP_START_HDL 550

20.7.8 GAPM_DEFAULT_GATT_START_HDL 550

20.7.9 GAPM_DEFAULT_ATT_CFG 551

20.7.10 GAPM_DEFAULT_TX_OCT_MAX 551

20.7.11 GAPM_DEFAULT_TX_TIME_MAX 551

20.7.12 GAPM_DEFAULT_MTU_MAX 551

20.7.13 GAPM_DEFAULT_MPS_MAX 552

20.7.14 GAPM_DEFAULT_MAX_NB_LECB 552

20.7.15 GAPM_DEFAULT_AUDIO_CFG 552

20.7.16 GAP_ROLE_MASTER 552

20.7.17 GAP_ROLE_SLAVE 552

20.7.18 GAPM_DEFAULT_ADV_DATA 553

20.7.19 GAPM_DEFAULT_ADV_DATA_LEN 553

20.7.20 GAPM_DEFAULT_SCANRSP_DATA 553

20.7.21 GAPM_DEFAULT_SCANRSP_DATA_LEN 553

20.7.22 GAPM_DEFAULT_SCAN_INTERVAL 553

20.7.23 GAPM_DEFAULT_SCAN_WINDOW 554

20.7.24 GAPM_DEFAULT_CON_INTV_MIN 554

20.7.25 GAPM_DEFAULT_CON_INTV_MAX 554

20.7.26 GAPM_DEFAULT_CON_LATENCY 554

20.7.27 GAPM_DEFAULT_SUPERV_TO 555

20.7.28 GAPM_DEFAULT_CE_LEN_MIN 555

20.7.29 GAPM_DEFAULT_CE_LEN_MAX 555

20.7.30 GAPM_DEFAULT_ADV_INTV_MIN 555

20.7.31 GAPM_DEFAULT_ADV_INTV_MAX 556

20.7.32 GAPM_DEFAULT_ADV_CHMAP 556

20.7.33 GATTC_DEFAULT_START_HDL 556

20.7.34 GATTC_DEFAULT_END_HDL 556

20.7.35 CS_SERVICE_UUID_16 556

20.7.36 CS_SERVICE_UUID_32 557

20.7.37 CS_SERVICE_UUID_128 557

20.7.38 CS_ATT_SERVICE_128 557

20.7.39 CS_ATT_CHARACTERISTIC_128 558

20.7.40 CS_ATT_CLIENT_CHAR_CFG_128 558

20.7.41 CS_ATT_CHAR_USER_DESC_128 558

20.7.42 CS_CHAR_UUID_16 ... 558

20.7.43 CS_CHAR_UUID_32 ... 559

20.7.44 CS_CHAR_UUID_128 ... 559

20.7.45 CS_CHAR_CCC ... 559

20.7.46 CS_CHAR_USER_DESC ... 559

20.7.47 MIN ... 560

20.7.48 HCI_VS_RF_CW_ENABLE_CMD_CODE ... 560

20.7.49 HCI_VS_RF_CW_DISABLE_CMD_CODE ... 560

20.7.50 HCI_VS_RF_OUTPUT_PWR_CMD_CODE ... 560

20.8 Bluetooth Low Energy Stack Abstraction Function Documentation ... 561

20.8.1 GAP_Initialize ... 561

20.8.2 GAP_GetEnv ... 561

20.8.3 GAP_GetProfileAddedTaskId ... 561

20.8.4 GAP_IsAddrPrivateResolvable ... 562

20.8.5 GAP_AddAdvData ... 562

20.8.6 GAPM_ResetCmd ... 563

20.8.7 GAPM_SoftwareReset ... 563

20.8.8 GAPM_PlatformReset ... 564

20.8.9 GAPM_SetDevConfigCmd ... 564

20.8.10 GAPM_GetDeviceConfig ... 564

20.8.11 GAPM_ProfileTaskAddCmd ... 565

20.8.12 GAPM_GetProfileAddedCount ... 566

20.8.13 GAPM_LepsmRegisterCmd ... 566

20.8.14 GAPM_GenRandAddrCmd ... 567

20.8.15 GAPM_ResolvAddrCmd ... 567

20.8.16 GAPM_ActivityCreateAdvCmd ... 567

20.8.17 GAPM_ActivityCreateScanCmd 568

20.8.18 GAPM_ActivityCreateInitCmd 569

20.8.19 GAPM_ActivityCreatePeriodSyncCmd 569

20.8.20 GAPM_AdvActivityStart 570

20.8.21 GAPM_InitActivityStart 571

20.8.22 GAPM_ScanActivityStart 571

20.8.23 GAPM_PerSyncActivityStart 572

20.8.24 GAPM_ActivityStartCmd 573

20.8.25 GAPM_ActivityStop 573

20.8.26 GAPM_ActivityStopAll 574

20.8.27 GAPM_DeleteActivity 574

20.8.28 GAPM_DeleteAllActivities 575

20.8.29 GAPM_ActivityStopCmd 575

20.8.30 GAPM_DeleteActivityCmd 576

20.8.31 GAPM_SetAdvDataCmd 577

20.8.32 GAPM_MsgHandler 578

20.8.33 GAPC_ParamUpdateCmd 578

20.8.34 GAPC_ParamUpdateCfm 579

20.8.35 GAPC_ConnectionCfm 579

20.8.36 GAPC_DisconnectCmd 580

20.8.37 GAPC_IsConnectionActive 580

20.8.38 GAPC_DisconnectAll 581

20.8.39 GAPC_ConnectionCount 581

20.8.40 GAPC_MasterConnectionCount 582

20.8.41 GAPC_SlaveConnectionCount 582

20.8.42 GAPC_GetConnectionInfo 582

20.8.43 GAPC_GetDevInfoCfm_Name 583

20.8.44 GAPC_GetDevInfoCfm_Appearance 584

20.8.45 GAPC_GetDevInfoCfm_SlvPrefParams 584

20.8.46 GAPC_GetDevInfoCfm 585

20.8.47 GAPC_SetDevInfoCfm 585

20.8.48 GAPC_BondCfm 586

20.8.49 GAPC_EncryptCmd 587

20.8.50 GAPC_EncryptCfm 587

20.8.51 GAPC_IsBonded 588

20.8.52 GAPC_GetBondInfo 588

20.8.53 GAPC_BondCmd 589

20.8.54 GAPC_AddRecordToBondList 590

20.8.55 GAPC_MsgHandler 590

20.8.56 GAPM_ListSetWlCmd 591

20.8.57 GAPM_ListSetRalCmd 591

20.8.58 GAPM_IsIRKValid 592

20.8.59 WhiteList_ResolvableList_Update 592

20.8.60 GAPM_GetWhitelistNumDev 593

20.8.61 GAPM_SetWhitelistNumDev 593

20.8.62 GAPC_SetPhyCmd 594

20.8.63 GAPC_CteTxCfgCmd 594

20.8.64 GAPC_CteRxCfgCmd 595

20.8.65 GAPC_CteReqCtrlCmd 595

20.8.66 GAPC_CteRspCtrlCmd 596

20.8.67 GAPC_GetInfoCmd 596

20.8.68 GAPM_PerAdvCteTxCmd 597

20.8.69 GAPM_PerAdvReportCtrlCmd 597

20.8.70 GAPM_PerSyncIQSamplingCtrlCmd 598

20.8.71 GATT_Initialize 599

20.8.72 GATT_GetEnv 599

20.8.73 GATT_SetEnvData 600

20.8.74 GATT_GetMaxCustomServiceNumber 600

20.8.75 GATTM_GetServiceAddedCount 601

20.8.76 GATTM_ResetServiceAttributeDatabaseID 601

20.8.77 GATTM_AddAttributeDatabase 601

20.8.78 GATTM_GetHandle 602

20.8.79 GATTM_MsgHandler 603

20.8.80 GATTC_Initialize 603

20.8.81 GATTC_DiscByUUIDSvc 603

20.8.82 GATTC_DiscAllSvc 604

20.8.83 GATTC_DiscAllChar 605

20.8.84 GATTC_SendEvtCmd 605

20.8.85 GATTC_SendEvtCfm 606

20.8.86 GATTC_MtuExchange 606

20.8.87 GATTC_ReadReqInd 607

20.8.88 GATTC_WriteReqInd 607

20.8.89 GATTC_AttInfoReqInd 608

20.8.90 GATTC_MsgHandler 609

20.8.91 Device_BLE_Public_Address_Read 609

20.8.92 Device_BLE_Param_Get 610

20.8.93 rand_func 611

20.8.94 set_app_rand_func 611

20.8.95 srand_func 611

20.8.96 set_app_rand_seed_val 612

20.8.97 default_rf_rssi_func 612

20.8.98 set_app_rf_rssi_func 612

20.8.99 MsgHandler_GetTaskAppDesc 613

20.8.100 MsgHandler_Add 613

20.8.101 MsgHandler_Notify 614

20.8.102 BASC_Initialize 615

20.8.103 BASC_EnableReq 615

20.8.104 BASC_ReadInfoReq 616

20.8.105 BASC_BattLevelNtfCfgReq 616

20.8.106 BASC_RequestBattLevelOnTimeout 617

20.8.107 BASC_GetLastBatteryLevel 617

20.8.108 BASC_GetEnv 618

20.8.109 BASC_MsgHandler 618

20.8.110 BASS_Initialize 619

20.8.111 BASS_NotifyOnTimeout 619

20.8.112 BASS_NotifyOnBattLevelChange 620

20.8.113 BASS_ProfileTaskAddCmd 620

20.8.114 BASS_EnableReq 621

20.8.115 BASS_BattLevelUpdReq 621

20.8.116 BASS_GetEnv 622

20.8.117 BASS_IsAdded 622

20.8.118 BASS_MsgHandler 622

20.8.119 DISS_Initialize 623

20.8.120 DISS_ProfileTaskAddCmd 624

20.8.121 DISS_EnvGet 624

20.8.122 DISS_IsAdded 624

20.8.123 DISS_MsgHandler 624

20.8.124 DISS_DeviceInfoValueReqInd 625

21. BLE_ABSTRACTION 626

21.1 Summary 626

21.2 BLE_ABSTRACTION Data Structures Type Documentation 626

21.2.1 BondInfo_t 626

21.3 BLE_ABSTRACTION Macro Definition Documentation 627

21.3.1 BOND_INFO_BASE 627

21.3.2 BOND_INFO_SIZE 627

21.3.3 STATIC_ASSERT 628

21.3.4 BONDLIST_MAX_SIZE 628

21.3.5 BOND_INFO_FLASH_SECTORS_COUNT 628

21.3.6 BOND_INFO_STATE_INVALID 628

21.3.7 BOND_INFO_STATE_EMPTY 628

21.3.8 BOND_INFO_STATE_VALID 629

21.4 BLE_ABSTRACTION Function Documentation 629

21.4.1 STATIC_ASSERT 629

21.4.2 BondList_Size 629

21.4.3 BondList_GetIRKs 629

21.4.4 BondList_FindByIRK 630

21.4.5 BondList_FindByAddr 631

21.4.6 BondList_FlashDefrag 631

21.4.7 BondList_Add 631

21.4.8 BondList_Remove 632

21.4.9 BondList_RemoveAll 632

CHAPTER 1

Introduction

1.1 Purpose

IMPORTANT: onsemi acknowledges that this document might contain the inappropriate terms “white list", "master" and "slave”. We have a plan to work with other companies to identify an industry wide solution that can eradicate non-inclusive terminology but maintains the technical relationship of the original wording. Once new terminologies are agreed upon, future products will contain new terminology.

This group of topics describes the firmware included in the RSL15 System-on-Chip. It includes descriptions, function listings, and usage examples to help you understand the firmware and its parts.

The firmware described in this reference provides developers with a convenient software layer on which to build their applications. It is also responsible for system-level tasks such as booting the system and implementing portions of the security layer. The firmware consists of include files, macros, libraries, ROM code, and executables.

1.2 Intended Audience

This group of topics is intended for use by developers who are designing and implementing applications for the RSL15 System-on-Chip. Both novice and experienced developers can benefit. The descriptions and code examples can help novice users to learn the system, while experienced developers can go straight to the reference chapters that describe the available functions, macros, libraries and executables.

These topics assume that the reader has a basic understanding of:

- The architecture of the RSL15 chip
- The C programming language
- The fundamentals of the Arm® Thumb®-2 assembly language
- The integrated development environment and toolchains that form the development tools for the RSL15 SoC

1.3 Conventions

`monospace font`

Assembly code, macros, functions, defines and addresses.

italics

File and path names, or any portion of them.

`<angle brackets>`

Optional parameters and placeholders for specific information. To use an optional parameter or replace a placeholder, specify the information within the brackets; do not include the brackets themselves.

*

Wild card placeholder; typically used to indicate a place where two or more numeric constants could be used.

Note, Important, Caution, Warning

Information requiring special notice is presented in several attention-grabbing formats depending on the consequences of ignoring the information:

NOTE: Significant supplemental information, hints, or tips.

IMPORTANT: Information that is more significant than a Note; intended to help you avoid frustration.

CAUTION: Information that can prevent you from damaging equipment or software.

WARNING: Information that can prevent harm to humans.

1.4 Further Reading

The following documents are installed with the RSL15 system, in the default location *C:/Users/*`<your_user_name>`*/AppData/Local/Arm/Packs/ONSemiconductor/RSL15/*`<version_number>`*/documentation*. These manuals are available only in PDF format:

- *Arm TrustZone CryptoCell-312 Software Developers Manual*
- multiple CEVA manuals in the */ceva* folder

For even more information, consult these publicly-available documents:

- *Armv8M Architecture Reference Manual* (PDF download available from https://developer.arm.com/documentation/ddi0553/latest).
- *Arm Cortex-M33 Processor Technical Reference Manual*, revision r1p0, from https://developer.arm.com/documentation/100230/0100
- *Bluetooth Core Specification version 5.2*, available from https://www.bluetooth.com/specifications/adopted-specifications
- TrustZone documentation available from the Arm website at https://developer.arm.com/ip-products/security-ip/trustzone/trustzone-for-cortex-m
- Other ArmCortex-M33 publications, available from the Arm website at https://developer.arm.com/ip-products/processors/cortex-m/cortex-m33

For information about the Evaluation and Development Board, see the RSL15 Evaluation and Development Board Manual and the RSL15 web page on the onsemi website.

CHAPTER 2

Firmware Overview

RSL15 is supported by a set of firmware components that provide:

- A thin layer of support between the hardware and the developer. This firmware allows developers to focus on application development, reducing the number of details they need to know about the underlying RSL15 hardware.
- A support layer for common complex run time operations frequently included in user applications
- Security and lifetime provisioning elements, which help support the security elements needed throughout a device's lifecycle
- Wireless protocol support functionality for Bluetooth® Low Energy applications
- Manufacturing and debug support elements

2.1 Introduction

The system firmware provides hardware definitions and a hardware abstraction layer for common operations. These definitions and abstraction layer are easier to use and understand than low-level C or assembly code. The firmware components supporting common run time, debug, and manufacturing elements simplify incorporation of the device into a system throughout the development cycle. In addition, the security and wireless protocol firmware elements provide more advanced functionality, which can form the basis for applications developed to take advantage of the hardware that the RSL15 SoC provides.

When multiple programmers are involved in development, using the firmware leads to increased consistency, which in turn leads to increased overall robustness and correctness of code.

In some cases, depending on the particulars of the application, the firmware implementation might not be optimal; however, even in these situations, the firmware serves as an example and an advanced starting point for custom-developed functions and macros.

All firmware components execute on the Arm Cortex®-M33 processor, and all of these components are CMSIS-compatible.

2.1.1 Compliance Exceptions

The firmware provided for the Arm Cortex-M33 processor is generally compliant with the MISRA-C rules, as required by the CMSIS standard. The RSL15 firmware exceptions in compliance are the same compliance exceptions that are part of the CMSIS-Core standard.

The RSL15 firmware and CMSIS-Core violate the following MISRA-C rules:

- Required Rule 8.5, object/function definition in header file. Violated because function definitions in header files are used to allow inlining of functions.
- Required Rule 18.4, declaration of union type or object of union type: {...}. Violated because unions are used for effective representation of core registers.
- Advisory Rule 19.7, function-like macro defined. Violated because function-like macros are used to allow for more efficient code.

2.2 Supported Data Types

The "RSL15 Supported Data Types" table (Table 1) shows the data types and their sizes as supported by the RSL15 firmware.

Table 1. RSL15 Supported Data Types

Data Type	Number of Bytes
`double`[1]	8
`float`	4
`int`	4
`long int`	4
`uint8_t`	1
`uint16_t`	2
`uint32_t`	4
`uint64_t`	8

NOTE: All `uint_t` types are included under the *stdint.h* library.

2.3 Firmware Components

The firmware files consist of include files (denoted with *.h* extensions) and library binaries (denoted with *.a* extensions). Some libraries are also provided in source code format. Descriptions of the firmware components, and detailed API references, are provided in the remainder of this group of topics.

Applications that use the libraries provided must:

1. Include the associated firmware include file.
2. Link against any dependencies of the desired library.
3. Link against a version of the desired library.

The Arm CryptoCell™-312 libraries are available in Release configuration.

The event kernel support firmware and the Bluetooth Low Energy protocol stack are available as a library.

2.4 Firmware Naming Conventions

For clarity and ease of use, many firmware components follow several naming conventions for library functions and macros. These conventions are compatible with the CMSIS naming requirements.

- Macros (defined using a `#define` statement) use all capitals in the macro name. These macro names include an all-capital prefix indicating the library or other firmware element they are supporting (e.g., `SYS_`). If the macro supports a specific target component, this prefix is followed by the name of the component it supports. The rest of the macro name indicates the intended functionality of the macro.
- Inline and standard firmware functions use camel-case function names (e.g., `CalcPhaseCnt`). All functions use a prefix to indicate which library provides the function (e.g., `Sys_`). The remainder of a function's name indicates the block it affects and the function's intended functionality.

Table 2 lists the prefixes for each of the firmware libraries that use prefixes.

Table 2. Library Function Naming Convention

Library	Macro Prefix	Function Prefix
Hardware Abstraction Layer, System Library	`SYS_`	`Sys_`
Flash Support Library	`FLASH_`	`Flash_`
Event Kernel	`KE_`	`Kernel_`, `ke_`

Elements that follow other naming conventions include the following:

- The CMSIS library and drivers follow the CMSIS standard, which provides standard names for all CMSIS macros and functions.
- The Arm CryptoCell-312 libraries largely use an API defined by Arm for the Arm TrustZone® CryptoCell-312 IP.
- The Bluetooth Library uses naming and terminology from the Bluetooth Core Specification; for more information on the naming conventions for the Bluetooth Library, see the CEVA® documentation provided with your RSL15 install.
- The swmTrace library is a logging library that is paired with the Real-Time Transfer (RTT) viewer that is part of the onsemi IDE and uses the RTT interface defined by SEGGER®.

2.5 Firmware Resource Usage

The firmware uses the Arm Cortex-M33 processor system stack. It expects that the Arm Cortex-M33 processor's stack pointer points to a valid stack that grows downward (i.e., decreasing memory addresses).

Other system memory used by the system are listed in the "Reserved Resources" table (Table 3).

Table 3. Reserved Resources

Reservation	Addresses	Notes
NVR0 to NVR3	0x00080000 to 0x000803FF	CryptoCell-312 support records and tables

NVR6	0x00080600 to 0x000806FF	System custom trim records. Firmware and sample applications use these records if available when the `Sys_Trim_LoadCustom()` function is called, selecting defaults for trim settings if they are not available.
		
		Optional contents include:
		
		• 0x80600 : 4-byte signature consisting of four ASCII encoded capital letters or numbers
		  • `SIP*`, `TRR*` signatures are reserved for use by onsemi.
		• 0x80604 : 4-byte `ICH_TRIM`
		  • To be stored to `ACS_VCC_CTRL_ICH_TRIM` in the `ACS_VCC_CTRL` register (valid range is 0x00000000 to 0x0000000F).
		• 0x80608 : 4-byte `XTAL_TRIM`
		  • To be stored to `RF_REG2E_XTAL_TRIM_XTAL_TRIM` in the `RF_REG2E` register (valid range is 0x00000000 to 0x000000FF).
		• 0x806F0: 4-byte Unix-style test timestamp
		• 0x806FC : CRC32 calculated over 0x80600 to 0x806FB
		Trims are considered available if all of the following are true:
		
		• The signature is valid
		• The calculated CRC is correct
		• The provided trim is within the valid range.

NVR7	0x00080700 to 0x000807FF	Trim records; accessible using the `TRIM_Type` structure from *rsl15_map.h*
MNVR	0x00080800 to 0x000808FF	Manufacturing and manufacturing trim records; cannot be modified outside of manufacturing.
FLASH_RSVD	0x00158000 to 0x00158BFF	Arm CryptoCell-312 DEU transfer space

Table 3. Reserved Resources (Continued)

Reservation	Addresses	Notes
FLASH_BOND_RSVD	0x00158C00 to 0x001593FF	Recommended location for Bluetooth Low Energy bond information
SystemCoreClock	0x20000000 to 0x20000003	Reserved for mandatory global variable required by CMSIS.
ROM Status	0x20000004 to 0x20000013	Program ROM status variables
ROM Reserved (Flash Library)	0x20000014 to 0x20000017	Reserved for ROM-linked flash library

Other system components (DMA channels, LSAD inputs, etc.) that are used by the firmware are noted in the documentation for the specific component.

Some RSL15 product variants might have additional custom trim values stored in the NVR6 region. Sample codes in the SDK include the `SYS_TRIM_LOAD_CUSTOM()` macro function, which reads and applies those values.

In detail, the `SYS_TRIM_LOAD_CUSTOM()` function reads the custom value of the DC-DC inductor charge current trim and the custom value of the XTAL 48 MHz oscillator trim, and loads them into corresponding registers (the `ICH_TRIM` field of the `ACS_VCC_CTRL` register, the `RF_REG2E_XTAL_TRIM_XTAL_TRIM_INIT` field of the `RF_REG2E` register, and the `RF_REG2E_XTAL_TRIM_XTAL_TRIM` field of the same register) if all of the following conditions are met:

- A custom flash value identifier, `CUST` or `SIP1`, is found.
- CRC value is verified correctly.
- Valid trim values are found.

When the custom flash value identifier is not found, no custom flash value has been programmed. As a result, the `SYS_TRIM_LOAD_CUSTOM()` function returns without reading or loading any trim values as mentioned above. In other words, in this case, default trim values will be used.

NOTE: Only call the `SYS_TRIM_LOAD_CUSTOM()` function after the RF block is powered. If it is called before the RF block is powered, and the device has been programmed with valid contents in its NVR6, a hardfault results, and the device gets reset.

2.6 Versions

Version symbols are provided for each major firmware component. The version symbols can be used directly or indirectly to verify the version of the components being used to build an application. There are three types of version symbols available:

Define
: A preprocessor define or set of defines containing the version information.

Symbolic
: A compiled symbol contained within a binary library

Global Variable
: A global variable in memory containing the symbol

As an example, the available version information for the flash library firmware component is listed in the "Example Firmware Versions - Flash Library" table (Table 4).

Table 4. Example Firmware Versions - Flash Library

Type	Version Symbol	Description
Define	`FLASH_FW_VER_MAJOR`	Major component of the library version; updated for non-backward compatible changes
Define	`FLASH_FW_VER_MINOR`	Minor component of the library version; updated for backward compatible changes, reset if major version is incremented
Define	`FLASH_FW_VER_REVISION`	Revision for the library version; incremented for minor changes or bug-fixes that do not affect library use
Define	`FLASH_FW_VER`	Combined library version (16 bits): • 15:12 major version • 11:8 minor version • 7:0 revision
Global Variable	`FlashLib_Version`	Constant variable assigned to hold the combined library version definition

[1]Unlike the `float` type, the `double` type is not directly supported by the hardware. The use of the `double` type is supported through a software implementation that is computationally taxing on the system. For the best performance, avoid using doubles whenever possible.

CHAPTER 3

Hardware Definitions

Hardware definition files are integral to the system firmware. The hardware definitions apply a layer of data structures and address mappings to the underlying hardware, so that every control register, bit field in the system, memory, and interrupt vector is easily accessible from C and assembly code. This set of header files provides a system definition for the RSL15 SoC, including:

- Register and bit descriptions for registers accessible to the processor
- Memory maps for all of the memories and memory-mapped elements that are accessible to the processor
- Interrupt vector table descriptions for the processor
- Macros that support the Arm Cortex-M33 processor's basic core functionality

The format and configuration of all of these support files conform to CMSIS compatibility requirements wherever possible. As required by CMSIS, the hardware definitions are included by the top-level CMSIS include file (*rsl15.h*) alongside the other CMSIS requirements. For more information about CMSIS, and this top-level include file, see "Introduction" on page 73.

If an application includes the top-level header file, and defines `RSL15_CID` to match the chip identifier of the relevant RSL15 device, then all of the support macros and HAL functions that are available to support that processor development on the chip are made accessible to that application.

NOTE: All devices that share a chip identifier are guaranteed to be compatible with the same firmware.

3.1 Register and Register Bit-Field Definitions

Using the hardware definition files allows you to refer to system components by C structures, assembly code, and preprocessor symbols instead of by addresses and bit fields. This greatly improves the readability, reliability and maintainability of your application code. The use of hardware definitions in an application also means that some hardware changes, such as changes to addresses or bit field values, are transparent to your application code.

Hardware register descriptions for the Arm Cortex-M33 processor's private peripherals are provided by the CMSIS package from Arm. Hardware register descriptions for all other components, and bit settings that are appropriate for use with the hardware register descriptions for the private peripherals, are available in the following files:

- *rsl15_hw.h*: This generic include file selects the desired underlying header file appropriate to your hardware by using the `RSL15_CID` definition.
- *rsl15_hw_cid*.h*: This include file is the header file that is appropriate for all devices that are compatible with the defined `RSL15_CID` (i.e., devices sharing the same chip version and major revision).
- *rsl15_hw_flat_cid*.h*: An unstructured version of the *rsl15_hw_cid*.h* header file, which includes all of the same definitions but no structure typedefs (useful for application elements written in assembly).

NOTE: For applications that are intended to operate in non-secure application modes, the *rsl15_hw_cid*_ns.h* headers are used, as long as `NON_SECURE` is defined by the preprocessor.

Hardware descriptions in the register include files provide definitions supporting the SoC components using the defines and C objects listed in the "Hardware Register Components" table (Table 5).

Table 5. Hardware Register Components

Item	Example	Description
Component Register Structure	`GPIO_Type`	Provides a list of all registers that support a specified component, and the read/write types for those registers
Component Register Instance	`GPIO`	Links the component register structure to the underlying hardware or sets of hardware
Bit-Field Positions	`GPIO_CFG_DRIVE_Pos`	Defines the base position for any bit-field within a register
Bit-Field Mask	`GPIO_CFG_DRIVE_Mask`	Defines a bit mask for any bit-field of more than one bit within a register
Register Structure	`GPIO_CFG_Type`	Provides a list of all sub-registers and alias structures • Sub-registers are defined byte (8-bit) or short (16-bit) access interfaces to part of a register that includes all elements belonging to the same configuration area. • Aliases are Arm Cortex-M33 processor bitband aliases that provide bit access to individual single-bit bit-fields where the underlying hardware supports this single-bit access.
Register Instance	`GPIO_CFG`	Links the register structure to the underlying hardware or sets of hardware for sub-registers
Bit-Setting	`GPIO_MODE_GPIO_IN_0`	Defines providing human-readable equivalents to settings that can be applied to a register bit-field to obtain the desired behavior
Bit-Field Sub-Register Positions	`GPIO_CFG_IO_MODE_BYTE_Pos`	Defines the base position for any bit-field within a register's sub-register
Bit-Field Sub-Register Mask	`GPIO_CFG_IO_MODE_BYTE_Mask`	Defines a bit mask for any bit-field of more than one bit within a register's sub-register
Sub-Register Bit-Setting	`GPIO_MODE_GPIO_IN_0_BYTE`	Defines providing human-readable equivalents to settings that can be applied to a register's sub-register bit-field to obtain the desired behavior

3.2 Memory Map Definition

The *rsl15_map.h* header file contains the specific memory map definitions that provide the locations of the following structures within the memory maps of the processor:

- Instruction and data bus memory structures
- System bus memory structures
- Peripheral bus memory-mapped control registers (including the base of control register groups for each system component)
- Private peripheral bus internal memory-mapped control registers
- System variables
- Calibration trim records

For more information on the memory map and memory mapped elements accessible to the Arm Cortex-M33 processor, see the *RSL15 Hardware Reference*.

3.3 Interrupt Vector Definition

Interrupt vectors provide you with access to interrupts that facilitate orderly processing operations, for optimal application processing speed and minimal delays. Interrupt vector definitions are provided for the Arm Cortex-M33 processor in the *rsl15_vectors.h* header file.

Interrupt handling functionality in the Arm Cortex-M33 processor is provided by a nested vector interrupt controller (NVIC) implemented with the processor. The NVIC handles predefined interrupts internal to the core including a non-maskable interrupt (NMI), and interrupts external to the processor, that are linked to interfaces and peripherals. The NVIC is supported by standard firmware as part of the CMSIS library (as described in Chapter 5 "CMSIS Library" on page 73), and by the Hardware Abstraction Layer library (described in Chapter 4 "Hardware Abstraction Layer" on page 72), which offers additional supporting functions. These definitions have the form **`<interrupt_name>`**`_IRQn`.

For more information and a complete list of interrupt vectors, along with enumeration defines and values, see the Arm Cortex-M33 Processor chapter of the *RSL15 Hardware Reference*.

CHAPTER 4

Hardware Abstraction Layer

The Hardware Abstraction Layer (HAL) library provides a set of multiple register access functions for the majority of the system components included in the RSL15 SoC. These functions, and the included function headers, can easily be used to configure the hardware blocks through safe register accesses with no need to refer to register tables and corresponding settings in the RSL15.

The HAL library is distributed in source-code form in the RSL15 SDK. All the library functions are blocking and interruptible. The library does not use any interrupts or DMA channels.

4.1 USAGE

To use the HAL library functions, the HAL library must be included via the project's **.rteconfig* file, under **Libraries**. Additionally, the *rsl15.h* header file must be included in the application source.

The HAL functions follow `Sys_`**`<hardware block>_<function>`** name syntax, with:

- **`<hardware block>`** indicating the system component being configured or acted upon
  - In some cases, the hardware block is prefixed by a standard operation such as `Set_` or `Get_`
- **`<function>`** describing the functionality implemented by the HAL firmware

Simple wrapper macros, with the same names but capitalized, are also provided for many functions. These macros do not require a user application to provide a pointer to the block being configured or operated upon, as the macros default to the first instance available. For example, `SYS_I2C_CONFIG` is a macro that calls the `Sys_I2C_Config` function with I2C0 for the pointer parameter.

For the complete HAL library API, refer to Chapter 16 "Hardware Abstraction Layer Reference" on page 159.

CHAPTER 5

CMSIS Library

The Arm Cortex-M33 processor is supported by a standards-compliant CMSIS library and extensions to the CMSIS requirements.

5.1 Introduction

The Arm Cortex-M33 processor is supported by a standards-compliant CMSIS library and extensions to the CMSIS requirements. The CMSIS library performs the following functions:

- Provides access to the generic functions provided by the standard Arm CMSIS implementation
  - The functions are included in the CMSIS header files, and reference documentation is provided by the standard Arm CMSIS documentation (http://arm-software.github.io/CMSIS_5/Core/html/modules.html).
- Implements the CMSIS-required device-specific functions with an appropriate implementation for RSL15
- Adds extensions to the generic and required CMSIS functions to provide additional common support functions, including:
  - Weak definitions for all interrupt handlers for the Arm Cortex-M33 processor and associated NVIC
  - A C start-up routine for ensuring that applications reach main in a safe state
  - Dynamic memory allocation (through an implementation of `sbrk`)
  - Hardware-safe functions for updating the system clock source and operating frequency

The weakly defined interrupt handlers included in the CMSIS library provide empty implementation for the interrupt vectors, each of which simply returns from the interrupt. These interrupt handlers have the form **`<interrupt_name>`**`_IRQHandler()`. If a user application defines a function with this same name, the user application's definition of the interrupt handler replaces the default (empty) handlers.

The CMSIS library is supported by the top-level include file for RSL15, *rsl15.h*, which includes all CMSIS, Hardware Definitions, and Hardware Abstraction Layer elements. To use the CMSIS library, include *rsl15.h* and link against the *libcmsis.a* library archive.

For the complete API, refer to Chapter 15 "CMSIS Reference" on page 138.

CHAPTER 6

CMSIS Drivers

The CMSIS drivers are generic device-independent APIs for peripherals, including I^2C, SPI, and UART. These drivers are created to allow easy setup and use of peripherals and interfaces by handling most of the manual configuration work in the back-end. The drivers can be used in the sample applications by including the appropriate interface header files and the CMSIS drivers library.

6.1 Introduction

This topic presents the description of all the available CMSIS drivers and the instructions on how to use the configuration file (*RTE_Device.h*). The specifications and headers for all of the functions used in these CMSIS drivers have been provided by Arm, and the corresponding documentation can be found on the Arm Keil website, on this page: http://www.keil.com/pack/doc/CMSIS/Driver/html/index.html.

This topic assumes that the user is familiar with importing projects into the IDE, and also with the location of the sample applications. For more information about these topics, see the *RSL15 Getting Started Guide*.

A list of interfaces with CMSIS driver support is shown in the "Interfaces for Which a CMSIS Driver is Available" table (Table 6).

Table 6. Interfaces for Which a CMSIS Driver is Available

CMSIS Driver	Interfaces Supported	Sample Application
DMA	DMA channels 0 to 3	dma
I2C	I2C0, I2C1	i2c_cmsis
SPI	SPI0, SPI1	spi_cmsis
USART	USART0	uart_cmsis

The drivers used for CMSIS pack sample applications are shown in the "Types of Drivers" table (Table 7):

Table 7. Types of Drivers

CMSIS Driver
SPI, I2C, USART

6.2 Using the CMSIS Drivers

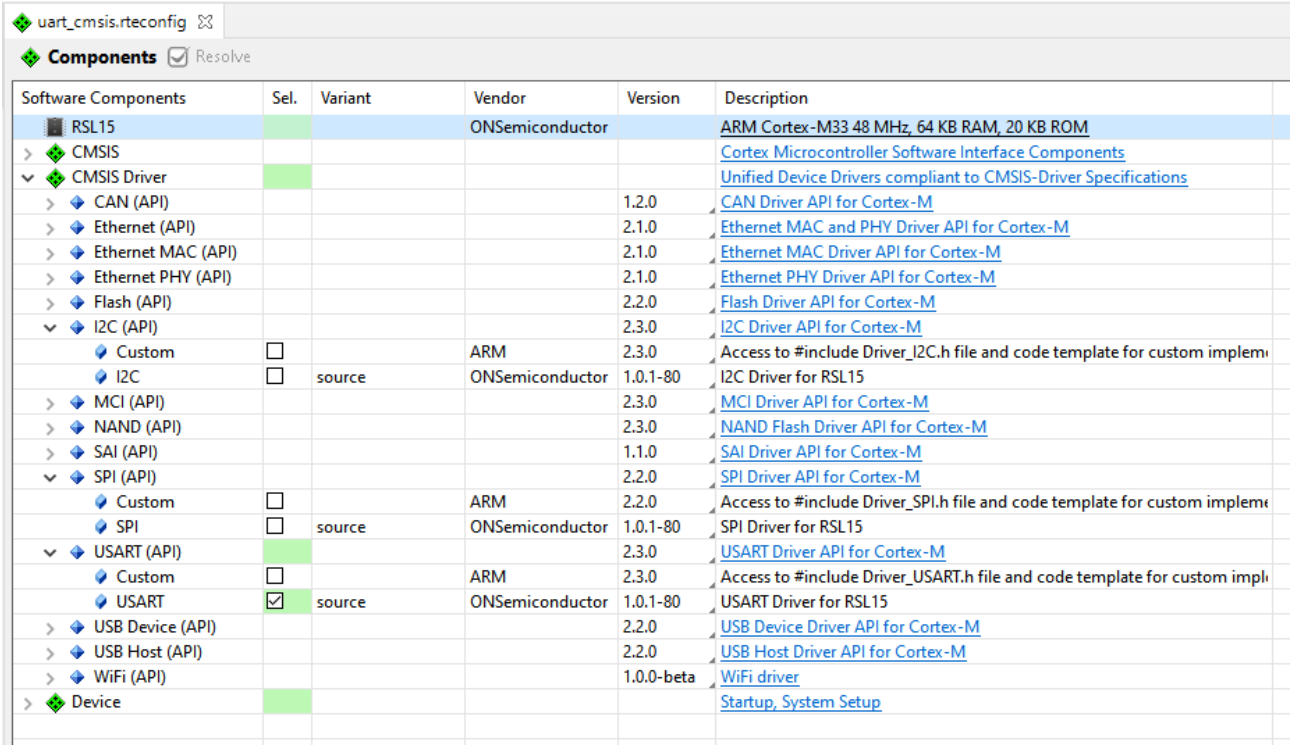
CMSIS drivers provide generic peripheral interfaces for middleware and application code. To use the CMSIS driver library, the drivers must be included in the application. Instructions for using the CMSIS drivers are shown in the following sections.

6.2.1 Adding a CMSIS Driver

To add a CMSIS driver to an application, perform the following steps:

1. Open the **`<sample_name>`**.*rtconfig* file from the project folder and select the CMSIS driver(s) that is/are required for the application. CMSIS drivers, including UART, SPI and I2C, are found under **CMSIS Drivers** in the application's *.rteconfig* file.
2. After enabling required drivers, save the **`<sample_name>`**.*rtconfig* file.

See the "Adding the UART CMSIS Driver to the uart_cmsis Sample Application" figure (Figure 1) as an example of how to add the USART CMSIS driver to the *uart_cmsis* sample application. As seen in the figure, the USART driver for RSL15 is checked, and is therefore included in the sample application.



Software Components	Sel.	Variant	Vendor	Version	Description
RSL15			ONSemiconductor		ARM Cortex-M33 48 MHz, 64 KB RAM, 20 KB ROM
CMSIS					Cortex Microcontroller Software Interface Components
CMSIS Driver					Unified Device Drivers compliant to CMSIS-Driver Specifications
CAN (API)				1.2.0	CAN Driver API for Cortex-M
Ethernet (API)				2.1.0	Ethernet MAC and PHY Driver API for Cortex-M
Ethernet MAC (API)				2.1.0	Ethernet MAC Driver API for Cortex-M
Ethernet PHY (API)				2.1.0	Ethernet PHY Driver API for Cortex-M
Flash (API)				2.2.0	Flash Driver API for Cortex-M
I2C (API)				2.3.0	I2C Driver API for Cortex-M
Custom	☐		ARM	2.3.0	Access to #include Driver_I2C.h file and code template for custom implem
I2C	☐	source	ONSemiconductor	1.0.1-80	I2C Driver for RSL15
MCI (API)				2.3.0	MCI Driver API for Cortex-M
NAND (API)				2.3.0	NAND Flash Driver API for Cortex-M
SAI (API)				1.1.0	SAI Driver API for Cortex-M
SPI (API)				2.2.0	SPI Driver API for Cortex-M
Custom	☐		ARM	2.2.0	Access to #include Driver_SPI.h file and code template for custom impleme
SPI	☐	source	ONSemiconductor	1.0.1-80	SPI Driver for RSL15
USART (API)				2.3.0	USART Driver API for Cortex-M
Custom	☐		ARM	2.3.0	Access to #include Driver_USART.h file and code template for custom impl
USART	☑	source	ONSemiconductor	1.0.1-80	USART Driver for RSL15
USB Device (API)				2.2.0	USB Device Driver API for Cortex-M
USB Host (API)				2.2.0	USB Host Driver API for Cortex-M
WiFi (API)				1.0.0-beta	WiFi driver
Device					Startup, System Setup

Figure 1. Adding the UART CMSIS Driver to the uart_cmsis Sample Application

6.2.2 Configuring CMSIS Drivers with RTE_Device.h

CMSIS drivers require I/O pin assignments and optional setup for DMA when being configured for use in a sample application. Both I/O pin assignments and DMA setup are configured in *RTE_Device.h*. In any sample application, the path to find the *RTE_Device.h* file is as follows: *RTE* > *Device* > *RSL15* > *RTE_Device.h*. To view and configure *RTE_Device.h* in an easily readable GUI, right click on *RTE_Device.h* in the Project Explorer view and select **Open With** > **CMSIS Configuration Wizard**. The Wizard window opens, as shown in the "Viewing and Configuring RTE_Device.h with the CMSIS Configuration Wizard" figure (Figure 2).

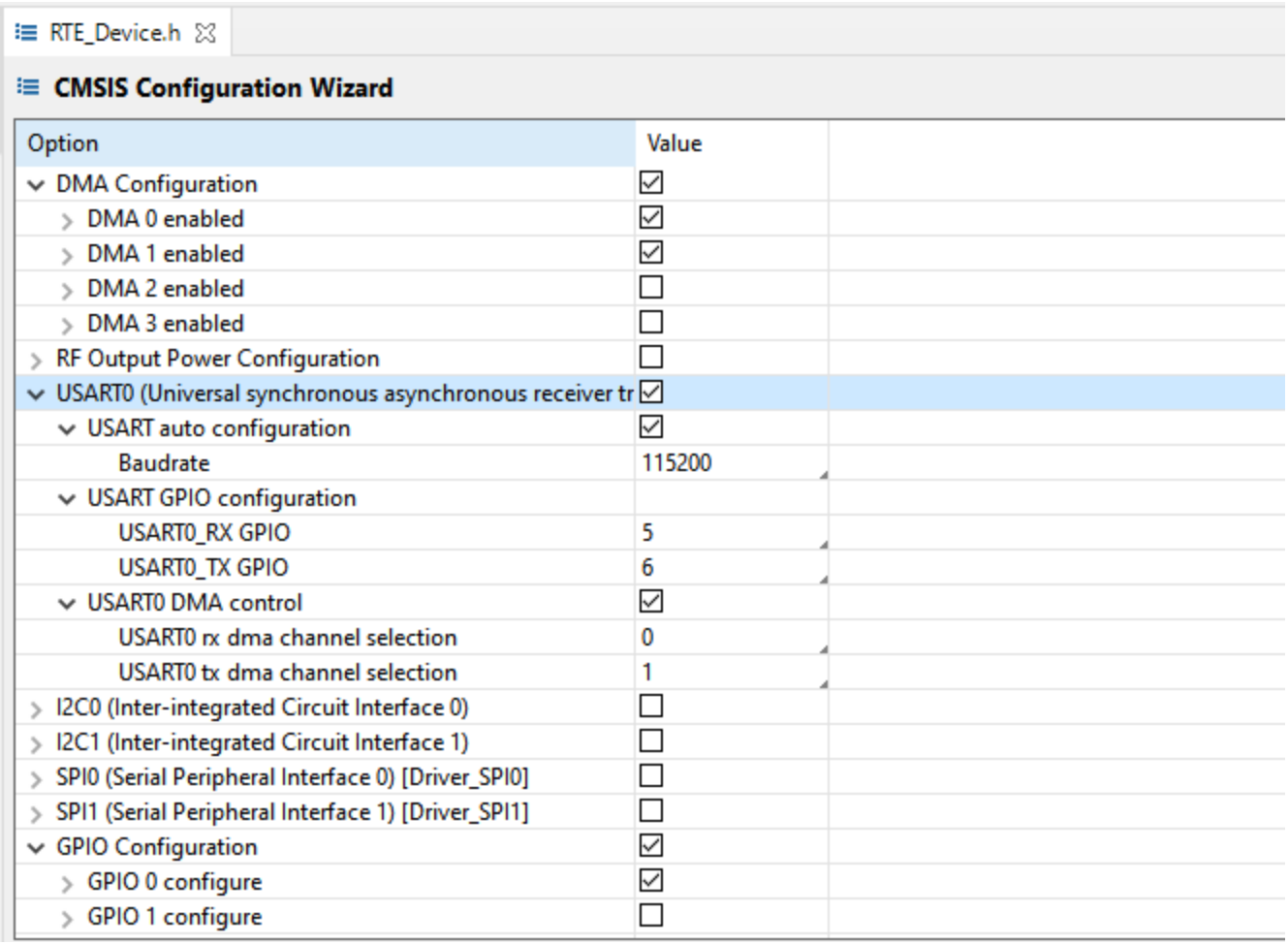


Figure 2. Viewing and Configuring RTE_Device.h with the CMSIS Configuration Wizard

NOTE: This file (*RTE_Device.h*) is best viewed using the CMSIS Configuration Wizard.

As seen in the "Viewing and Configuring RTE_Device.h with the CMSIS Configuration Wizard" figure (Figure 2), optional setup for DMA control is available in the sample application's *RTE_Device.h* file via the CMSIS Configuration Wizard. In this file, under **DMA Configuration**, the DMA channels can be configured.

DMA channel configuration can be enabled by checking the box for a channel, and disabled by unchecking it.. In the "Viewing and Configuring RTE_Device.h with the CMSIS Configuration Wizard" figure (Figure 2), the **DMA 0 enabled** and **DMA 1 enabled** boxes are checked, meaning that DMA channels 0 and 1 have their configurations enabled and are ready to be used for data transfers.

The CMSIS Configuration Wizard view of the *RTE_Device.h file* also shows options under **USART0 DMA Control** that allow channel selection for the USART0 RX DMA channel and USAR0 TX DMA channel. In the "Viewing and Configuring RTE_Device.h with the CMSIS Configuration Wizard" figure (Figure 2) the RX DMA channel for the USART0 interface is configured to use DMA channel 0, while the TX DMA channel uses DMA channel 1.

CHAPTER 7

Program ROM

The RSL15 Program ROM is responsible for ensuring that an RSL15 device correctly performs a controlled boot sequence allowing application code to execute. The Program ROM does this while taking into account the possible life cycle states of the device, the active power modes, and any required security functionality.

In addition to bringing the system up, the program ROM also ensures that the default state of a device is consistent after a cold boot sequence.

Finally, the ROM provides access to commonly-used functionality that can be called from application code, allowing these functions to be eliminated from the application footprint.

7.1 Overview

7.1.1 Versions

There are three version numbers reported by the RSL15 ROM. Each version number is defined as a 32-bit value containing major, minor and revision numbers in the form `major.minor.revision`. These items are stored at predefined locations in the ROM program memory, as shown in the "ROM Versions" table (Table 8).

Table 8. ROM Versions

Address	Version Type	Version
0x00000010	Program ROM	1.5.00
0x00000014	Secure Boot ROM Library (Arm)	1.0.00
0x00000018	Flash Library	3.0.00

7.2 ROM Initialization Sequence

The ROM is split into two main parts:

1. The low level initialization and setup, which is written in assembler: this part is responsible for ensuring that the power supplies are set up, and that memory instances are enabled for default power-up conditions.
2. The higher level functionality implemented in C dealing with the various peripherals, life cycle states and security features: these are defined in more detail in Section 7.5 “Security Subsystem” on page 84.

The ROM initialization sequence is shown graphically in the "ROM Initialization Sequence Flowchart" figure (Figure 3). The flowchart describes the various paths through the boot sequence, taking into account the possible life cycle states and power modes.

Figure 3. ROM Initialization Sequence Flowchart

The "Boot Options" table (Table 9) shows the available boot options.

Table 9. Boot Options

ACS_BOOT_CFG	CC312AO	Results
BOOT_ROT_BYPASS_ENABLE	Enable	Pass but with cold boot
BOOT_ROT_BYPASS_DISABLE	Enable	Pass but with cold boot
BOOT_ROT_BYPASS_DISABLE	Disable	Fail with CCAO_REBOOT_RESET_FLAG_SET in ACS_RESET_STATUS
BOOT_ROT_BYPASS_ENABLE	Disable	Fail with watchdog reset in boot ROM

7.2.1 ROM Basic Initialization

The basic system initialization function is called once the power supplies and memories have been set to known default states. At this stage the RAM is enabled and the C stack is available for use.

The primary sequence of this initialization is as follows:

- Set up the NVIC to allow only NMI and Hard Fault exceptions.
- Disable the MPU, bus, and usage faults, forcing them to be always promoted to hard faults.
- Disable all of the external interrupts and clear all pending status indicators.
- Assign the NMI to a constant low source.
- Stop any running flash or flash copier operations.
- Reset the GPIOs.
- Enable all JTAG pins.
- Disable Pad Retention Mode.
- Configure the watchdog timer for a maximum timeout, and refresh.
- Disable DMA.
- Configure Non Secure code zone.
- Disable the SAU and set the ALLNS to zero so that everything is secure.
- Configure the non-secure accesses to RAM and peripherals as disabled.
- Make sure that the flash is not busy before setting up the RC.
- Configure the RC for 12 MHz.
- Select the RC oscillator as the system clock.
- Set up the various clock pre-scalars.
- Disable RF access.
- Disable the baseband interface.
- Enable all memories.
- Ensure that the Arm CryptoCell-312 (CC312) is enabled and ready to run.
- Set the system core clock variable and flash delays, indicating that the device is configured to use the 12 MHz RC oscillator.

7.2.2 ROM Status Variables & Status Codes

As part of its execution, the ROM sets several status variables indicating its execution status. These variables are stored near the base of RAM, starting at 0x20000004:

ROM Status
: Variable indicating the status of the ROM execution, residing at 0x20000004.

ROM Context

Variable indicating the state of the device in which the ROM is executing. For example, this state could be energy harvesting mode or one of the secure life cycle modes. Resides at 0x20000008.

Application Status

Variable that is free to be used by the application. Resides at 0x2000000C.

Application Context

Variable that is free to be used by the application. Resides at 0x20000010.

The two tables below contain the possible values of ROM status ("ROM Status Values" table (Table 10)) and ROM context (the "ROM Context Values" table (Table 11).

Table 10. ROM Status Values

Status or Context Name	Description	Value
RS_PRODUCTION_BOOT	Indicates that the ROM is in production boot mode	0xBB000001
RS_FATAL_ERROR	Indicates a fatal boot error	0x0000FE00
RS_SE_BOOT	Indicates secure boot status values	0x5EC00000
RS_SE_BOOT_BAD_CONFIG	Indicates a bad configuration has been detected during a secure boot	0x5EC0FE01 (RS_SE_BOOT | RS_FATAL_ERROR | 0x00000001)
RS_COLD_BOOT	Indicates a cold boot state	0xCB000000
RS_COLD_BOOT_FAILED	Indicates a cold boot failure.	0xCB00FE00 (RS_COLD_BOOT | RS_FATAL_ERROR)

Table 11. ROM Context Values

Status or Context name	Description	Value
RS_EH_BOOT	Energy harvesting mode boot	0xEE000000
RS_EH_NO_KEY	Energy harvesting mode, no key provided	0xEE000001 (RS_EH_BOOT + 1)
RS_EH_INV_KEY	Energy harvesting mode, the key provided is invalid	0xEE000002 (RS_EH_BOOT + 2)
RS_EH_HAS_KEY_NO_CERT	Energy harvesting mode, the key is provided but there is no valid certificate.	0xEE000003 (RS_EH_BOOT + 3)
RS_EH_HAS_KEY_HAS_CERT	Energy harvesting mode, the key is provided and there is a valid certificate.	0xEE000004 (RS_EH_BOOT + 4)
RS_EH_NO_DCU	Energy harvesting mode, the DCU setting is missing.	0xEE000005 (RS_EH_BOOT + 5)
RS_EH_INV_DCU	Energy harvesting mode, the DCU setting is invalid.	0xEE000006 (RS_EH_BOOT + 6)
RS_EH_HAS_DCU	Energy harvesting mode, the DCU setting is valid	0xEE000007 (RS_EH_BOOT + 7)

Table 11. ROM Context Values (Continued)

Status or Context name	Description	Value
RS_CONTEXT	Indicates a context variable.	0x00008000
RS_CONTEXT_RMA_FAIL	Indicates a RMA state failure	0x00008100 (RS_CONTEXT + 0x100)
RS_CONTEXT_ROT_FAIL	Indicates a Root-Of-Trust failure	0x00008200 (RS_CONTEXT + 0x200)
RS_CONTEXT_ALL_BITS	Masking allowing modification of all context bits.	0xFFFFFFFF
RS_CONTEXT_LCS_BITS	Mask allowing modification of Life-cycle state bits.	0x000000FF
RS_CONTEXT_CTX_BITS	Mask allowing modification of Life-cycle state bits.	0x0000FF00

7.3 Application Validation and Boot

The RSL15 program ROM contains a set of functions that are used to validate and boot applications, following the completion of the security processes.

The ROM considers an application valid if it starts with its vector table, and no errors that would prevent boot are detected. Possible errors, and the error codes reported for these errors, are described in the "Application Validation" table (Table 12).

Table 12. Application Validation

Error	Error Code	Description
None	0x0	No error detected
Bad Alignment	0x1	The Arm Cortex-M33 processor requires that the application's vector table is aligned to a 512-byte boundary in memory, for a device with the number of external interrupts that are included in the RSL15 SoC. The location of the specified application is not at a valid location in memory.
Bad Stack Pointer	0x2	The initial stack pointer must point to a valid memory location on the system bus. This requires that the specified stack pointer is 32-bit aligned, and that the next address stack data will be placed at is in DRAM or BB_DRAM.

Table 12. Application Validation (Continued)

Error	Error Code	Description
Bad Reset Vector	0x3	The program ROM checks that the reset handler is located immediately after the vector table (or after a CRC located after the vector table). This check is performed indirectly by confirming that the reset vector points to a location that: • Provides space for at least the minimum number of entries in the vector table (a minimum valid vector table contains 4 entries: the stack pointer, reset vector, NMI handler, and hard fault handler) • Provides space for no more than the stack pointer, the 88 potential vectors, and a CRC (maximum of 90 words between the base of the application and the reset vector's location)
Failed to Start the Application	0x6	Indicates that the application has failed to boot or has returned with no identifiable cause.
Bad CRC	0x7	A CRC-CCITT value can be placed between the vector table and the reset handler. The boot validation step validates if a CRC calculated over the vector table matches the value written at this location. NOTE: This error code is considered to be a non-fatal error, since the inclusion of a CRC is optional. The first entry on the application's stack after boot will indicate whether no-error has occurred (0x0) or if a bad CRC has been discovered (0x7).

If the ROM determines that an application should be booted, the ROM:

1. Sets the VTOR bit-field in the Arm Cortex-M33 processor's SCB register to point to the application's vector table
2. Loads the initial stack pointer value from the application's vector table to the Arm Cortex-M33 processor's SP register
3. Pushes the application's status code to the `SYSCTRL_SYSCLK_CNT` register (valid error codes for a booted application are None and Bad CRC, as described in the "Application Validation" table (Table 12))
4. Branches to the beginning of the reset handler, as indicated by the reset vector in the application's vector table

7.4 Vector Tables

The ROM provides some access to built-in common functionality via the use of a ROM Jump Table. The features provided in this jump table are defined in the "Jump Table Functions" table (Table 13), together with a brief description of the use of each. Access to these functions is provided through *rom_vect.h* (typically included through the top-level *rsl15.h* include file) and *flash_rom.h* (used in place of *flash.h*).

Table 13. Jump Table Functions

Offset	Feature	Description
0x00	__Sys_Initialize_Base void __Sys_Initialize_Base(void);	Base device initialization
0x04	__Sys_Delay void __Sys_Delay(unsigned int cycles);	Delay a number of cycles
0x08	__ProgramROM_ValidateApp uint32_t __ProgramROM_ValidateApp(uint32_t* app_addr);	Validate an application
0x0C	__ProgramROM_StartApp uint32_t __ProgramROM_StartApp(uint32_t* app_addr);	Validate an application, and if valid, start execution of that application
0x10	Flash_Initialize FlashStatus_t Flash_Initialize(unsigned no, FlashClockFrequency_t freq);	Initialize a flash bank
0x14	Flash_WriteWord FlashStatus_t Flash_WriteWord(bool enb_endurance);	Write 32-bit word to flash
0x18	Flash_ReadWord FlashStatus_t Flash_ReadWord(uint32_t addr, uint32_t *word);	Read 32-bit word from flash
0x1C	Flash_WriteDouble FlashStatus_t Flash_WriteDouble(uint32_t addr, const uint32_t *word, bool enb_endurance);	Write two 32-bit words to flash
0x20	Flash_ReadDouble FlashStatus_t Flash_ReadDouble(uint32_t addr, uint32_t *word);	Read two 32-bit words from flash

Table 13. Jump Table Functions (Continued)

Offset	Feature	Description
0x24	Flash_WriteBuffer FlashStatus_t Flash_WriteBuffer(uint32_t addr, uint32_t word_length, const uint32_t *words, bool enb_endurance);	Write an array of 32-bit words
0x28	Flash_ReadBuffer FlashStatus_t Flash_ReadBuffer(uint32_t flash_address, uint32_t dram_address, unsigned word_length);	Read an array of 32-bit words
0x2C	Flash_EraseFlashBank FlashStatus_t Flash_EraseFlashBank(uint32_t no);	Erase a specific flash bank
0x30	Flash_EraseChip FlashStatus_t Flash_EraseChip(void);	Erase all flash
0x34	Flash_EraseSector FlashStatus_t Flash_EraseSector(uint32_t addr, bool enb_endurance);	Erase Flash Sector
0x38	Flash_BlankCheck FlashStatus_t Flash_BlankCheck(uint32_t addr, unsigned word_length);	Verify area of flash is empty
0x3C	__ProgramROM_Read_MNVR void __ProgramROM_Read_MNVR(uint32_t addr, uint32_t *word, uint8_t *readECC);	Read data from MNVR

7.5 Security Subsystem

RSL15 includes security IP provided by Arm, specifically the Arm CryptoCell-312 (CC312) security processor. In addition, the Arm Cortex-M33 processor itself includes support for TrustZone.

The ROM ensures that the hardware Root-of-Trust (RoT) is verified when running as a secure device. This ensures that no untrusted code can be executed by the ROM.

RSL15 also supports deployment as a non-secure device, allowing much lower power utilization. This mode is provided primarily for energy harvesting applications. However, this is not limited by design; any application can be executed in the lower security model if required.

IMPORTANT: For detailed information about the device states, the secure Root of Trust (RoT), and tools provided to support the security subsystem, see the *RSL15 Security User's Guide***.**

7.5.1 Secure Boot Process

There are a limited number of steps involved when evaluating the secure Root of Trust (RoT) in the secure state:

- Verify the peripheral ID.
- Get the current life cycle state.
- Secure debug authentication:
    - If debug certificates are available, they must be validated.
    - If valid, debug facilities need to be enabled.
- Image verification phase:
    - Verify the certificate chain.
    - Handle key certificates.
    - Handle content certificates and content.
    - Lock resources.
- In the case of an error occurring during this flow, abort the boot sequence in a secure state.
- If validation completes and the image is authenticated, execute the validated image.

7.6 ROM Life Cycle States (LCS) & Operational Modes

The standard Arm CryptoCell-312 secure life cycle model allows for four life cycle states:

- CM - Chip Manufacture
- DM - Device Manufacture
- SE - Secure

The transitions between these states are strictly defined by the life cycle model, and we have maintained this in our system.

In addition to the standard Arm lifecycle states, we have added an additional production state before CM which reflects the possibility of a device coming from manufacture with an unconfigured OTP. In this case, the ROM does not allow application code to be run, but it does allow the device to be configured to enter CM state.

In parallel to the secure life cycle model, we also provide for a non-secure model in which the device is defined to be in Energy-Harvesting (EH) state. In this case, the Root of Trust security features are turned off and security is provided using a security mechanism dependent on valid applications unlocking the device.

A device in Production state can be configured to be in EH state. If this is not explicitly configured, then the device is treated as a secure device.

- A completely unconfigured device is defined to be in Production state.
    - In this state, the device can be configured as either a secure device or a non-secure device.
    - In this state, no executable code is run on power-up.
    - In manufacturing, we expect all devices to be set as non-secure devices.
- A non-secure device has a signature indicating its state in OTP.
    - In order to transition from the non-secure state to a secure state, this signature needs to be removed.
    - As the signature resides in OTP, it cannot be re-instated once it has been deleted. This provides a oneway transition from non-secure to secure state.
- Once a device is set to secure, there is no mechanism to bring it back to non-secure.

IMPORTANT: If a device is in the Production state, it is possible to go directly to a secure state without transitioning through the non-secure state. In this case, care must be taken to clear the non-secure signature locations, as otherwise it can be possible to revert the device to non-secure. See the *RSL15 Security User's Guide* **for more information on device transitions.**

7.7 Data Exchange Unit (DEU) Support

In order to enable the debug ports and any other features controlled by the Debug Control Unit, certificates must be provided to the device. These can reside in RAM for one time only use, or can reside in flash to allow the port enablement to survive a cold reboot.

The provision of these certificates is handled through the DEU. All interfacing with the DEU is performed via the ROM code. Application code has no access to this device.

The ROM supports the following features when communicating with the DEU:

- `LOAD` - Allows a certificate to be loaded to a device and stored in RAM
- `ERASE` - Erases any certificates which are currently stored in flash
- `WRITE` - Writes the current set of certificates from RAM to flash
- `SOCID` - Requests the SOC ID from the device
- `CONNECT` - Connects to a device via the DEU port
- `COMPLETE` - Completes the connection to the device

7.8 Reserved Flash Sectors

On RSL15, the application configuration and debug certificates are held in data flash in consecutive sectors. (Data flash in RSL15 has a sector size of 256 bytes.)

- One sector for the application configuration
- One sector for the EH_STATE debug certificate
- Ten sectors for the ROT_STATE debug certificates

Therefore, a total of 3 KB are reserved for use by the different security mechanisms.

The layout of this is shown in the "Reserved Space for Security Mechanisms" figure (Figure 4).

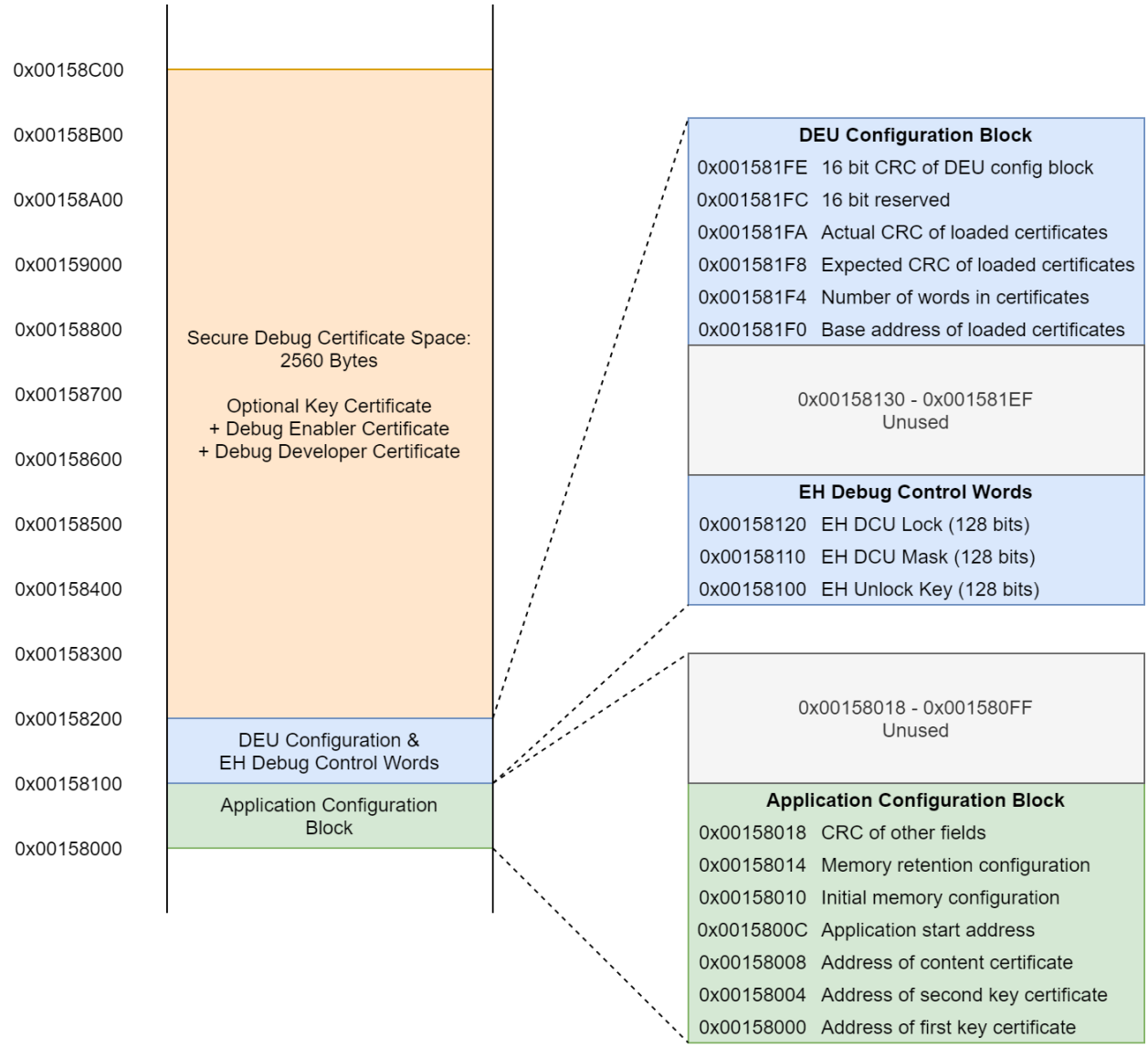


Figure 4. Reserved Space for Security Mechanisms

If you are not using the secure Root of Trust, then it is generally considered safe to repurpose the 10 sectors allocated to the debug certificates. The other two sectors need to be left reserved.

CHAPTER 8

Flash Library

The flash library provides support for erasing and programming parts of the built in flash memory. This library provides an API that abstracts the details needed properly handle the flash.

8.1 Library Forms

The flash library is available in two forms:

1. As a static library, accessed by including the *flash.h* header file and linking against the *libflashlib.a* library object
2. As a Program ROM component, accessed as memory mapped elements using the *flash_rom.h* header file

For the complete flash library API, refer to Chapter 1 "Flash Library Reference" on page 1.

IMPORTANT: A copy of the flash library has been built into the ROM, as described in Section 7.4 “Vector Tables” on page 81, for use in applications. All functions provided by the flash library must be executed from RAM or ROM, as executing them from flash can result in hidden, flash-access-related failures.

8.2 Flash Library Usage

The flash library can be used to program and erase the built-in flash, with the following considerations:

- A minimum SYSCLK frequency of 1 MHz is required for safe operation of the flash library.
- The MNVR section of flash, outside of the user-defined redundancy sector pointers, cannot be written using the flash library.
- The endurance of flash is limited by the total time flash cells are subjected to program and erase currents. As a result, a number of flash operations provide options around the endurance of the flash cells:
    - Flash writes can use a one stage normal write or a two stage endurance write. For use cases where an area of flash is written many times, a two stage write is recommended.
    - Use of mass erase can speed up flash operations, as it is the quickest way to erase the whole flash array - but use of mass erase subjects the whole array to an erase current for significantly longer than seen with any sector erase operation.
    - Flash sector erases can use a endurance erase that attempts to erase the flash sector up to four times, using a short erase pulse to maximize the number of program/erase cycles that a flash sector can be subjected to. For use cases that require maximum retention time, the normal flash sector erase needs to be used.

For more information about the built-in flash memory, see the *RSL15 Hardware Reference.*

CHAPTER 9

Security and Life Cycle Provisioning Elements

Information about the security and lifetime provisioning elements for RSL15 is primarily found in the *RSL15 Security User's Guide*.

9.1 Overview

The Security and Life Cycle Provisioning process can be managed using RSLSec, which is a software utility available for RSL15. Its command line interface enables users to perform the transitions and access other security features through parameters, as described in the *RSL15 Security User's Guide*. This utility works to leverage the underlying security system for RSL15, including the ROM and security IP.

9.2 Third Party Documentation

The Arm TrustZone CryptoCell-312 Software Developers Manual (SW Revision r0p0) is provided with the RSL15 SDK, in PDF form in the *Arm_documentation* folder.

Additional third-party information can be found on the https://developer.arm.com/ip-products/security-ip/trustzone/trustzone-for-cortex-m webpage.

CHAPTER 10

Arm CryptoCell-312 Security IP

The RSL15 system includes the Arm CryptoCell-312 IP, which is used to maintain security of the device, allow secure operation of the application firmware, and permit related user applications to access security features.

IMPORTANT: The `CC312` functions, when using hardware accelerators, use a WFI instruction while waiting for the hardware accelerator to complete its processing. Therefore, these functions must not be called in the context of any interrupt routines that have a priority higher than, or equal to, the `CC312` interrupt.

IMPORTANT: The `CC312` functions are not reentrant. These functions cannot be called again until their execution has been completed.

A significant part of the Arm CryptoCell-312's operation is based on the use of the Mbed TLS libraries.

10.1 Arm CryptoCell-312 Mbed TLS

RSL15 provides a standardized interface to the cryptographic features on the device using a customized version of Mbed TLS, which has been tuned to make efficient use of the cryptographic acceleration hardware provided by the Arm CryptoCell-312.

At present, the Mbed TLS variant supported by RSL15 is version 2.16.2. Some features of this have been replaced by alternative versions to make the best use of the underlying hardware. The interface has been maintained so that standard Mbed TLS applications can work as expected.

For further information on developing cryptographic applications with RSL15, refer to the *Arm CryptoCell-312 Runtime Software Developers Manual*, Issue 01, revision r1p3.

CHAPTER 11

Event Kernel

NOTE: Some of the material in this topic has been adapted with permission from CEVA documentation. This chapter is intended to complement the CEVA API documentation included with your RSL15 install.

The RSL15 event kernel is a small and efficient event and message handling system that can be used as a Real Time Operating System (RTOS) or as a process executed under an RTOS, offering the following features:

- Exchange of messages
- Message saving
- Timer functionality
- The kernel also provides an event functionality used to defer actions

The purpose of the event kernel is to provide messages (such as the ones in Section 11.2 “Messages” on page 90) and timed tasks to keep RF traffic on schedule and aligned with the specification requirements.

11.1 Overview

11.1.1 Include Files

The "Kernel File List" table (Table 14) shows a list of the kernel include files, with descriptions.

Table 14. Kernel File List

File	Description
ke.h	Contains the kernel environment definition
ke_event.h	Contains the event handling primitives
ke_mem.h	Contains the implementation of the heap management module
ke_msg.h	This file contains the scheduler primitives called to create or delete a task. It also contains the scheduler itself
ke_task.h	Contains the implementation of the kernel task management
ke_timer.h	Contains the scheduler primitives called to create or delete a timer task. It also contains the timer scheduler itself

11.1.2 Kernel Environment

The kernel environment structure contains the queues used for event, timer and message management, including:

- A queue of sent messages that have not yet been delivered to the receiver
- A queue of messages delivered to the receiver, but not yet consumed
- A queue for timer events

11.2 Messages

11.2.1 Overview

Message queues provide a mechanism to transmit one or more messages to a task. (Queue names and purposes are defined in Section 11.1.2 “Kernel Environment” on page 90.)

Transmission of messages is performed in three steps:

- Sender task allocates a message structure
- Message parameters are filled
- Message structure is pushed in the kernel

A message is identified by a unique ID composed of the task type and an increasing number. A message has a list of parameters that is defined in a structure (see Section 11.2.2 “Message Format” on page 91).

11.2.2 Message Format

The structure of the message contains:

- `id`: Message identifier
- `dest_id`: Destination kernel identifier
- `src_id`: Source kernel identifier
- `param_len`: Parameter embedded structure length
- `param`: Parameter embedded structure. Must be word-aligned.

The source and destination tasks describe which task the message is coming from/to. As a task can implement multiple instances, these source and destination task identifiers are constructed with both the task index and the task type, as shown in the "Destination or source task identifier construction" figure (Figure 5).

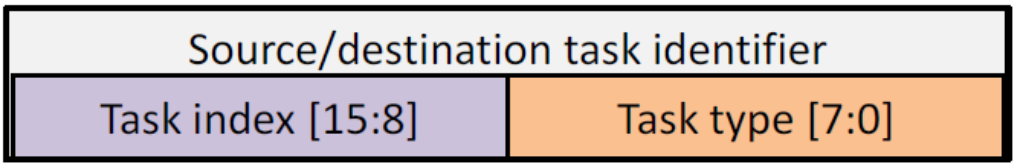


Figure 5. Destination or source task identifier construction

11.2.3 Parameter Management

During message allocation, the size of the parameter is passed and memory is allocated in the kernel heap. To store this data, the pointer on the parameters is returned. The scheduler frees this memory after the transition completion.

11.2.4 Message Queue Primitives

11.2.4.1 Message Allocation

`ke_msg_alloc`

Prototype:

```
void *ke_msg_alloc(ke_msg_id_t const id, ke_task_id_t const dest_id,
                   ke_task_id_t const src_id, uint16_t const param_str)
```

Parameters:

Table 15. Message Allocation Parameters

Type	Parameters	Description
`ke_msg_id_t`	`id`	Message Identifier
`ke_task_id_t`	`dest_id`	Destination Task Identifier
`ke_task_id_t`	`src_id`	Source Task Identifier
`uint16_t`	`param_len`	Size of the message parameters to be allocated

Return:

Pointer to the parameter member of the `ke_msg`. If the parameter structure is empty, the pointer points to the end of the message and must not be used (except to retrieve the message pointer or to send the message).

Description:

This primitive allocates memory for a message that has to be sent. The memory is allocated dynamically on the heap, and the length of the variable parameter structure must be provided in order to allocate the correct size.

11.2.4.2 Message Allocation Macro (Static Structure)

```
KE_MSG_ALLOC
```

Prototype:

```
KE_MSG_ALLOC(id, dest_id,
             src_id, param_str)
```

Parameters:

Table 16. Message Allocation Macro Parameters for Static Structures

Type	Parameters	Description
`ke_msg_id_t`	`id`	Message Identifier
`ke_task_id_t`	`dest_id`	Destination Task Identifier
`ke_task_id_t`	`src_id`	Source Task Identifier
`void const *`	`param_str`	Structure tag for the message data

Return:

Pointer to the allocated structure cast to the correct type.

Description:

This macro calls `ke_msg_alloc()` and cast the returned pointer to the appropriate structure. Can only be used if a parameter structure exists for this message (otherwise, use `ke_msg_send_basic()`).

11.2.4.3 Message Allocation Macro (Variable Structure)

KE_MSG_ALLOC_DYN

Prototype:

```
KE_MSG_ALLOC_DYN(id, dest_id,
                src_id, param_str, length)
```

Parameters:

Table 17. Message Dynamic Allocation Parameters

Type	Parameters	Description
`ke_msg_id_t`	`id`	Message Identifier
`ke_task_id_t`	`dest_id`	Destination Task Identifier
`ke_task_id_t`	`src_id`	Source Task Identifier
`void const *`	`param_str`	Structure tag for the message data
`uint16_t`	`length`	Length of the variable portion of the data structure

Return:

Pointer to the allocated structure of variable length, cast to the correct type.

Description:

This macro calls `ke_msg_alloc()` and cast the returned pointer to the appropriate structure. Can only be used if a parameter structure exists for this message (otherwise, use `ke_msg_send_basic()`).

NOTE: This function can only be used if the variable data array is located at the end of the structure to be allocated.

11.2.4.4 Message Free

ke_msg_free

Prototype:

```
void ke_msg_free(struct ke_msg const *param)
```

Parameters:

Table 18. Message Free Parameters

Type	Parameters	Description
`struct ke_msg const *`	`param`	Pointer to the message to be freed

Return:

None

Description:

Free allocated message.

11.2.4.5 Message Free Macro

```
KE_MSG_FREE
```

Prototype:

```
KE_MSG_FREE(param_ptr)
```

Parameters:

Table 19. Message Free Parameters

Type	Parameters	Description
`void const *`	`param_ptr`	Structure tag for the message data

Return:

None

Description:

This macro calls `ke_msg_free()` to free a previously allocated message.

11.2.4.6 Message Send

```
ke_msg_send
```

Prototype:

```
void ke_msg_send(void const *param_ptr)
```

Parameters:

Table 20. Message Send Parameters

Type	Parameters	Description
`void const *`	`param_ptr`	Pointer to the parameter member of the message that is to be sent

Return:

None

Description:

Send a message previously allocated with any `ke_msg_alloc()`-like functions. The kernel takes care of freeing the message memory. Once the function has been called, it is not possible to access the message's data any more as the kernel might have copied the message and freed the original memory.

11.2.4.7 Message Send Basic

```
ke_msg_send_basic
```

Prototype:

```
void ke_msg_send_basic(ke_msg_id_t const id, ke_task_id_t const dest_id, ke_task_id_t
const src_id)
```

Parameters:

Table 21. Message Send Basic Parameters

Type	Parameters	Description
ke_msg_id_t	id	Message Identifier
ke_task_id_t	dest_id	Destination Task Identifier
ke_task_id_t	src_id	Source Task Identifier

Return:

None

Description:

Send a message that has a zero length parameter member. No allocation is required as this is performed internally.

11.2.4.8 Message Forward

```
ke_msg_forward
```

Prototype:

```
void ke_msg_forward(void const *param_ptr, ke_task_id_t const dest_id, ke_task_id_t
const src_id)
```

Parameters:

Table 22. Message Forward Parameters

Type	Parameters	Description
void const *	param_ptr	Pointer to the parameter member of the message that is to be sent
ke_task_id_t	dest_id	Destination Task Identifier
ke_task_id_t	src_id	Source Task Identifier

Return:

None

Description:

Forward a message to another task by changing its destination and source task IDs.

11.3 SCHEDULER

11.3.1 Overview

The scheduler is called in the main loop of the user application using the `BLE_Kernel_Process()` function.

IMPORTANT: For maximum stability, ensure that the user application calls `BLE_Kernel_Process()` at least once during each Bluetooth connection interval. For more information, see Section 13.2 “Baseband and Kernel Functions” on page 119.

In the user application’s main loop, the kernel checks if the event field is non-null, and executes the event handlers for which the corresponding event bit is set.

11.3.2 Requirements

11.3.2.1 Scheduling Algorithm

The "Scheduling Algorithm" figure (Figure 6) shows how the scheduler handles messages. The message handler pops messages from the message queue, passes them to the pre-defined message handler, and then handles either releasing or saving those messages based on the responses from those handlers.

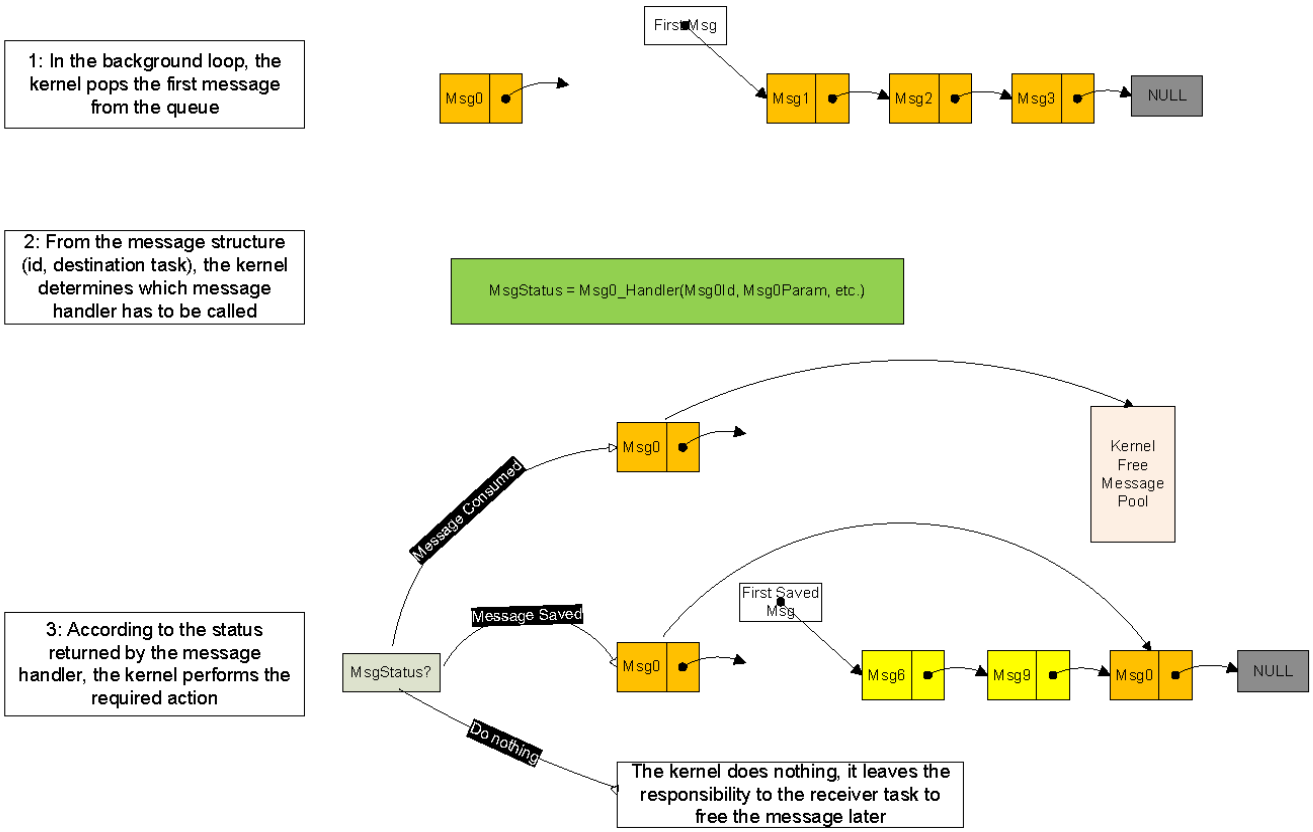


Figure 6. Scheduling Algorithm

11.3.2.2 Save Service

The Save service can save a message (i.e., store it in memory without it being consumed). If the task state changes after a message is received, the scheduler tries to handle the saved message before scheduling any other signals.

11.4 Tasks

A kernel task is defined by:

- Its task type (i.e., a constant value defined by the kernel, unique for each task)
- Its task descriptor, which is a structure as shown in the "Task Descriptor Construction" figure (Figure 7) containing all the information about the task:
    - The messages handlers table
    - The states table
    - The number of instances of the task
    - The number of messages it can handle

The kernel keeps a pointer to each task descriptor, which is used to handle the scheduling of the messages transmitted from one task to another.

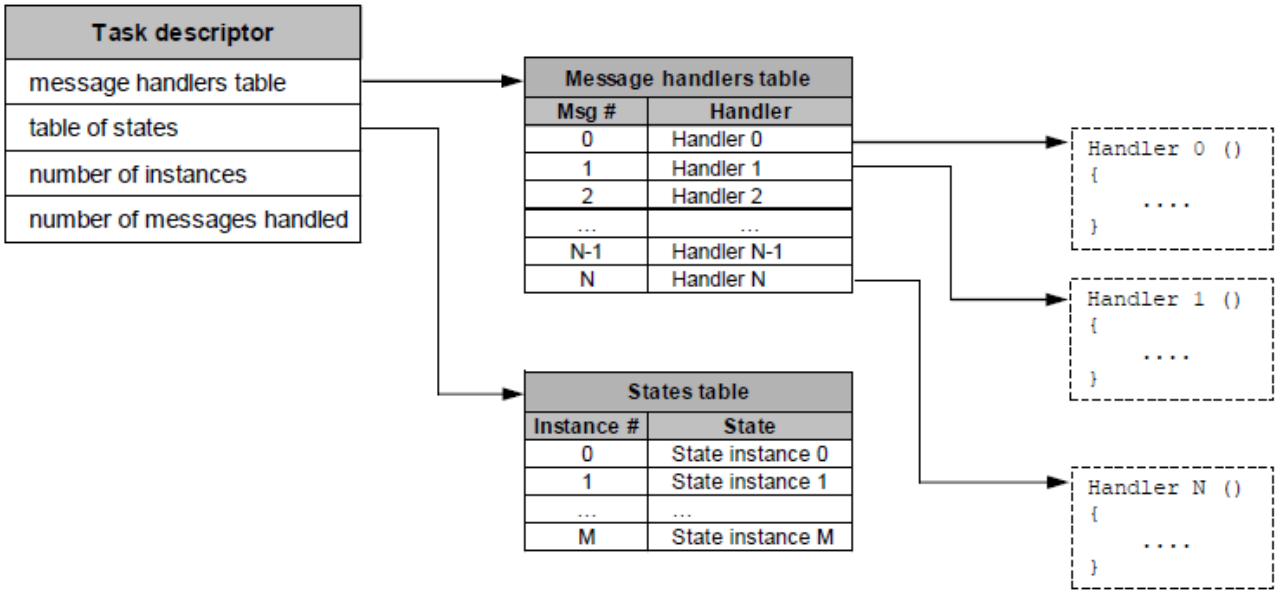


Figure 7. Task Descriptor Construction

11.5 Kernel Timer

11.5.1 Overview

The RW Kernel provides:

- A time reference (absolute time counter)
- Timer services to start and stop the timer

Timers are implemented by means of a reserved queue of delayed messages. Timer messages do not have parameters.

IMPORTANT: Kernel timer functions (`ke_timer_*`) must not be used in an interrupt context. Do not call them within any interrupt service routines. The kernel timer is not interrupt-safe.

11.5.2 Time Definition

Time is defined as duration; the minimum step is a multiple of 1 ms.

11.5.3 Timer Object

The structure of the timer message contains:

- `*next`: Pointer on the next timer
- `id`: Message identifier
- `task`: Destination task identifier
- `time`: Duration

11.5.4 Timer Setting

The "Timer Setting Flow" figure (Figure 8) shows the flow for setting up timer messages.

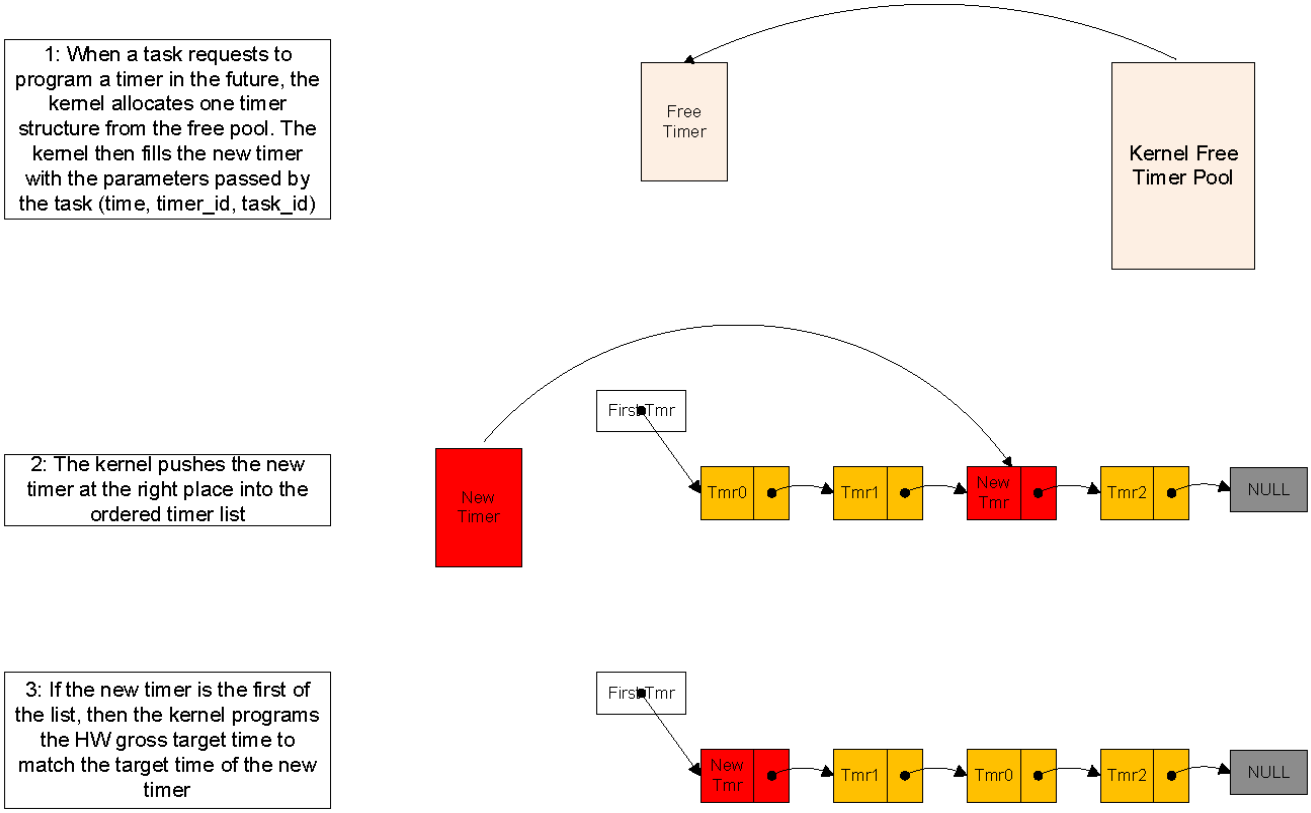


Figure 8. Timer Setting Flow

11.5.5 Time Primitives

11.5.5.1 Timer Set

Set a timer.

Prototype:

```
void ke_timer_set(ke_msg_id_t const timer_id, ke_task_id_t const task, uint32_t const delay);
```

Parameters:

Table 23. Timer Set Parameters

Type	Parameters	Description
`ke_msg_id_t`	`timer_id`	Timer identifier
`ke_task_id_t`	`task_id`	Task identifier
`uint16_t`	`delay`	Timer duration (multiple of 1 ms)

Return:

None

Description:

The function first cancels the timer if it exists; then it creates a new one.

When the timer expires, a message is sent to the task provided as argument, with the timer id as message id.

11.5.5.2 Timer Clear

Remove a registered timer.

Prototype:

```
void ke_timer_clear(ke_msg_id_t const timer_id, ke_task_id_t const task);
```

Parameters:

Table 24. Timer Clear Parameters

Type	Parameters	Description
`ke_msg_id_t`	`timer_id`	Timer identifier
`ke_task_id_t`	`task_id`	Task identifier

Return:

None

Description:

This function searches for the timer element identified by its timer and task identifiers. If found, it is stopped and freed; otherwise an error message is returned.

11.5.5.3 Timer Activity

Check if a requested timer is active.

Prototype:

```
bool ke_timer_active(ke_msg_id_t const timer_id, ke_task_id_t const task);
```

Parameters:

Table 25. Timer Activity Parameters

Type	Parameters	Description
`ke_msg_id_t`	`timer_id`	Timer identifier
`ke_task_id_t`	`task_id`	Task identifier

Return:

`TRUE` if the timer identified by Timer ID is active for the Task ID; `FALSE` otherwise

Description:

This function pops the first timer from the timer queue and notifies the appropriate task by sending a kernel message. If the timer is periodic, it is set again; if it is one-shot, the timer is freed. The function also checks the next timers, and processes them if they have expired or are about to expire. The "Timer Expiry Flow" figure (Figure 9) shows the process flow for handling expired timers.

11.5.5.4 Timer Adjust

Adjust all kernel timers by specified adjustment delay.

Prototype:

```
void ke_timer_adjust_all(uint32_t delay);
```

Parameters:

Table 26. Timer Adjust Parameters

Type	Parameters	Description
`uint16_t`	`delay`	Adjustment delay is a multiple of 1 ms

Return:

None

Description:

This function updates all timers to align to a new `SYSCLK` after a system clock adjustment.

11.5.5.5 Timer Expiry

The "Timer Expiry Flow" figure (Figure 9) shows the flow of timer messages when the timer expires.

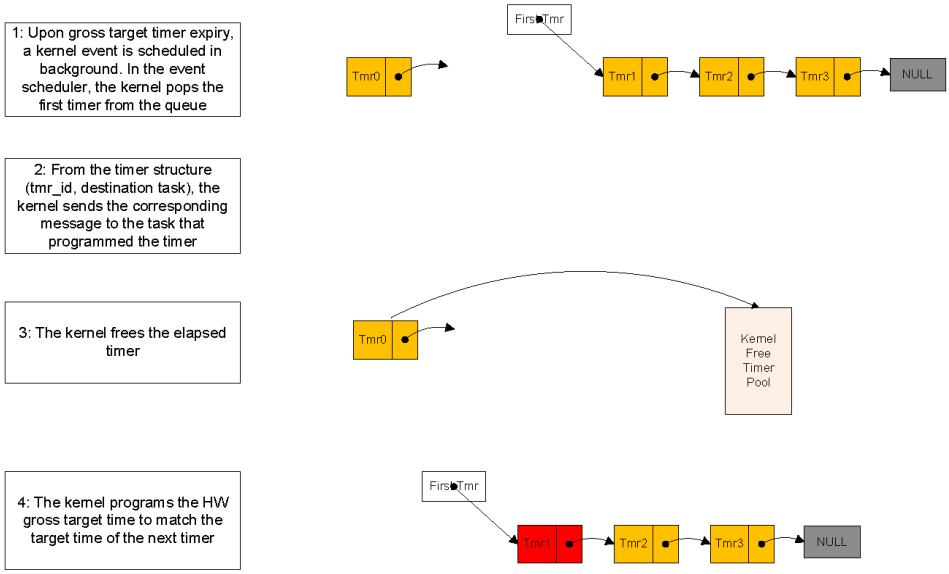


Figure 9. Timer Expiry Flow

11.6 Useful Macros

- Builds the task identifier from the type and the index of that task:

```
#define KE_BUILD_ID(type, index) ( (ke_task_id_t)(((index) << 8)|(type)) )
```

- Retrieves task type from task id:

```
#define KE_TYPE_GET(ke_task_id) ((ke_task_id) & 0xFF)
```

- Retrieves task index number from task id:

```
#define KE_IDX_GET(ke_task_id) (((ke_task_id) >> 8) & 0xFF)
```

CHAPTER 12

Bluetooth Stack

NOTE: Some of the material in this topic has been adapted with permission from CEVA documentation. This chapter is intended to complement the CEVA API documentation included with your RSL15 install. Consult this documentation when API function information cannot be found in the CEVA documentation.

This topic explains how the Bluetooth stack, including the HCI (host/controller interface), GATT (generic attribute profile), and GAP (generic access profile), is implemented for RSL15. This chapter also provides a description of the Bluetooth profile libraries that are provided with the RSL15 system to support standard use cases.

12.1 Introduction

12.1.1 Include and Object Files

The Bluetooth stack is accessed through the *ble.h header* file. This header is supported by a set of include files located in the *include\ble* folder of the installation.

The stack is extended by a number of GATT-based profiles and services. The headers for these profiles are located in the *include\ble\profiles* folder, and a list of the available supporting objects is provided in the "Bluetooth GATT-Based Profile and Service Object Files" table (Table 27).

Table 27. Bluetooth GATT-Based Profile and Service Object Files

Profile Name	Profile	Profile Library Name	Profile Description
Battery Service	BAS	libbasc libbass	The Battery Service exposes the state of a battery within a device, or it reads the battery state of a peer device.
Device Information Service	DIS	libdisc libdiss	This service exposes manufacturer and/or vendor information about their own device or discovers peer device information.
Glucose Service	GLS	libglps	This service exposes glucose and other data from a personal glucose sensor for use in consumer healthcare applications.

All of the individual profile libraries use the Bluetooth Low Energy stack through the profile's specified interfaces. These interfaces are documented in the interface specifications. Because the Bluetooth Low Energy stack itself requires a reciprocal link in order to find all of the profile components, the stack library has been built with an object factory that instantiates calls to each of the profiles. If a profile is used by an application, the Bluetooth stack needs to use the specified profile library.

12.1.2 Bluetooth Stack

The RSL15 device supports a Bluetooth stack through a combination of hardware and firmware resources. The hardware components of the Bluetooth stack are described in the *RSL15 Hardware Reference*. The firmware components of the Bluetooth stack are accessible through a Bluetooth library and associated header files.

The Bluetooth stack is optionally accessible through HCI over UART or through the host (GAP, GATT, L2CAP) and profile APIs.

The "Bluetooth Stack and Kernel Object Files" table (Table 28) describes the Bluetooth stacks provided with RSL15 and their associated object files.

Table 28. Bluetooth Stack and Kernel Object Files

Stack Type	Library Name	Description
Standard stack	*\ib\ble_core\Release* *\ib\ble_profiles\Release*	These libraries provide the complete standard stack and support for all RSL15 Bluetooth features.
HCI stack	*\ib\ble_core\Release_HCI* *\ib\ble_profiles\Release_HCI*	These libraries can be used with an HCI interface over UART.

12.1.3 Stack Support Functions

The Bluetooth stack library includes a set of support functions that augment the stack firmware, as described in Chapter 13 "Bluetooth and Kernel Library" on page 117. All other stack APIs are described in their reference documentation, with support for specific Bluetooth layers described in the following documents:

GAP
: *RW-BLE-GAP-IS.pdf*

GATT
: *RW-BLE-GATT-IS.pdf*

L2CAP
: *RW-BLE-L2C-IS.pdf*

Profiles
: *RW-BLE-PRF-*-IS.pdf*

12.1.4 Managing Bluetooth Low Energy Stack RAM Usage

The Bluetooth Low Energy stack is provided as a complied library and cannot be modified by the user. However, some of the stack variables are defined at the application level, and the user has some control over the size of these variables. This is a guide to optimizing stack RAM memory usage by adjusting the application level variable defines as dependent upon the application use case.

The application level variables are defined to support the maximum number of connections and activities, which is 10 and 11 respectively, as described in the CEVA documentation provided with RSL15. If the application use case does not require the maximum number of connections and activities, application level variable sizes can be reduced to save memory.

NOTE: It is not recommended to adjust application level variable sizes in the HCI stack, as this can adversely affect Bluetooth certification.

The simplest method and recommended starting point to reduce stack memory usage is to reduce the value of `APP_MAX_NB_CON`. This scales most of the other application level variables as well, resulting in a reduction of memory usage. This is usually sufficient for most users. If you are an advanced user requiring further optimization, read on.

The memory allocated by the stack is divided into different parts. The first part comprises the required environment and global variables that are independent of the number of connections or activities. This part cannot be changed by developers.

The second part is heap memory, which the kernel takes as a global variable (*.bss* section), and later the kernel and the Bluetooth Low Energy stack use it as the heap memory for their procedures. It is divided into four parts:

- Environment variables
- Message heap memory
- Data base memory
- Non-retention memory

By default, all parameters are set to address the maximum number of connections and activities, which are 10 and 11 respectively. If definition `APP_HEAP_SIZE_DEFINED` is defined in the *app.h* file of any Bluetooth Low Energy application, then the heap sizes can be customized in the *ble_protocol_support.c* file (where they are defined and allocated). The following code example shows how the required memory sizes can be calculated (defined in *ble_protocol_support.c*).

```
/// Memory allocated for environment variables
uint32_t rwip_heap_env[RWIP_CALC_HEAP_LEN(APP_RWIP_HEAP_ENV_SIZE)];
/// Memory allocated for Attribute database
uint32_t rwip_heap_db[RWIP_CALC_HEAP_LEN(APP_RWIP_HEAP_DB_SIZE)];
/// Memory allocated for kernel messages
uint32_t rwip_heap_msg[RWIP_CALC_HEAP_LEN(APP_RWIP_HEAP_MSG_SIZE)];
/// Non Retention memory block
uint32_t rwip_heap_non_ret[RWIP_CALC_HEAP_LEN(APP_RWIP_HEAP_NON_RET_SIZE)];
```

We assume:

`APP_MAX_NB_ACTIVITY` and `APP_MAX_NB_CON` are defined in the application, based on the application's use case. Please note that `PP_BLE_CONNECTION_MAX` needs to be at least one value bigger than `APP_MAX_NB_CON`.

```
APP_RWIP_HEAP_ENV_SIZE =
(600 + (APP_MAX_NB_ACTIVITY) * 230)
     + APP_MAX_NB_CON * ((sizeof(struct gapc_env_tag)
     + KE_HEAP_MEM_RESERVED)
     + (sizeof(struct gattc_env_tag)  + KE_HEAP_MEM_RESERVED)
     + (sizeof(struct l2cc_env_tag)   + KE_HEAP_MEM_RESERVED))
     + ((APP_MAX_NB_ACTIVITY ) * (sizeof(struct gapm_actv_scan_tag)
                                        + KE_HEAP_MEM_RESERVED))
```

`APP_RWIP_HEAP_DB_SIZE`: This depends on how much data base memory (GATT services) is added. The default value is 3072 bytes, but developers can choose a smaller value based on their use case (see Section 13.3 "Managing Bluetooth Low Energy Stack RAM Usage" on page 123 for more information).

```
APP_RWIP_HEAP_MSG_SIZE =
(1650 + 2 * ((16 + (APP_MAX_NB_ACTIVITY - 1) * 56)
      + (58 + (APP_MAX_NB_ACTIVITY - 1) * 26)
      + ((APP_MAX_NB_ACTIVITY) * 66)
      + ((APP_MAX_NB_ACTIVITY) * 100)
      +((APP_MAX_NB_ACTIVITY) * 12)))
      +(((BLEHL_HEAP_MSG_SIZE_PER_CON * APP_MAX_NB_CON) > BLEHL_HEAP_DATA_THP_SIZE)
                                                        ?
       (BLEHL_HEAP_MSG_SIZE_PER_CON * APP_MAX_NB_CON) : BLEHL_HEAP_DATA_THP_SIZE)
```

`APP_RWIP_HEAP_NON_RET_SIZE`: By default, this is set to 656 bytes. It is used for security algorithm calculations. This part of memory does not need to be in retention mode when the use case calls for keeping the

Bluetooth Low Energy link active and going to sleep with VDDM in retention. If enough memory is allocated — for example, in the database — the kernel can allocate this part of memory from another heap. Developers need to make sure that this default amount of memory is available, in non-ret heap or in another heap.

In this way, users can set their own `APP_MAX_NB_CON` and `APP_MAX_NB_ACTIVITY` values, and decrease the memory size allocated by default settings.

12.2 HCI

The role of the HCI is to provide a uniform interface method of accessing a Bluetooth Low Energy controller's capabilities from the host. The HCI layer is part of the Bluetooth Low Energy protocol stack, as shown in the "Bluetooth Low Energy Protocol Stack" figure (Figure 10).

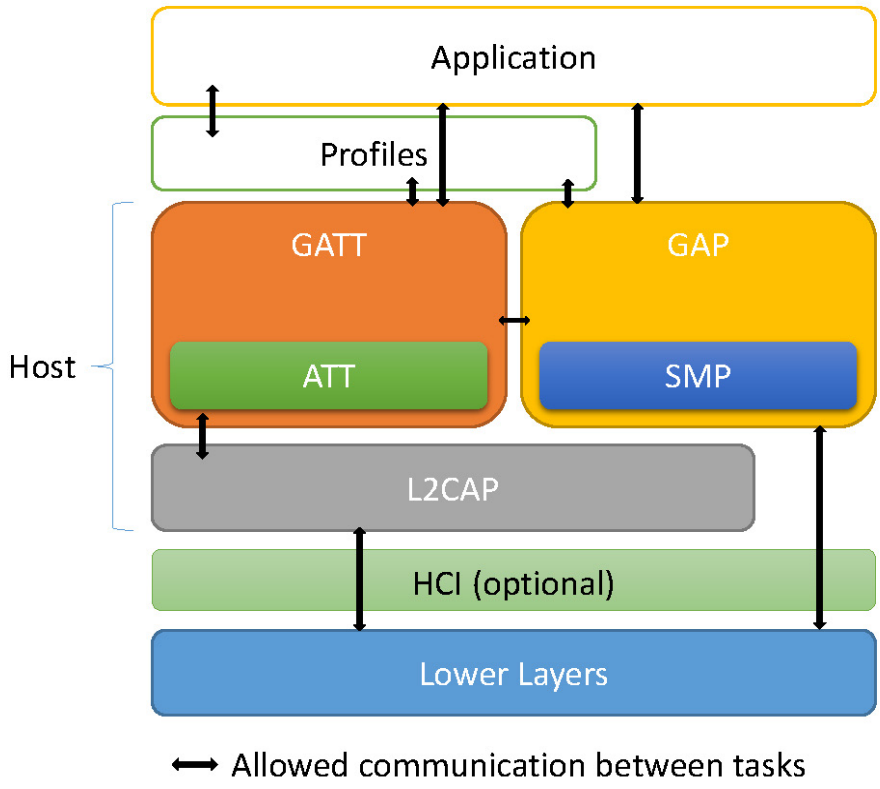


Figure 10. Bluetooth Low Energy Protocol Stack

The Bluetooth stack optionally provides an HCI layer, which provides direct access to the stack at an interface between the host and controller layers. The role of the HCI is to convey the information from one layer to the other by following the rules defined in the HCI portion of the Bluetooth standard. As shown in the sample code, the RSL15 HCI layer implementation can be used to interface with a transport layer that manages the reception and transmission of messages over a UART physical interface.

As shown in the "HCI" figure (12.2), the two main configurations are supported by the HCI software.

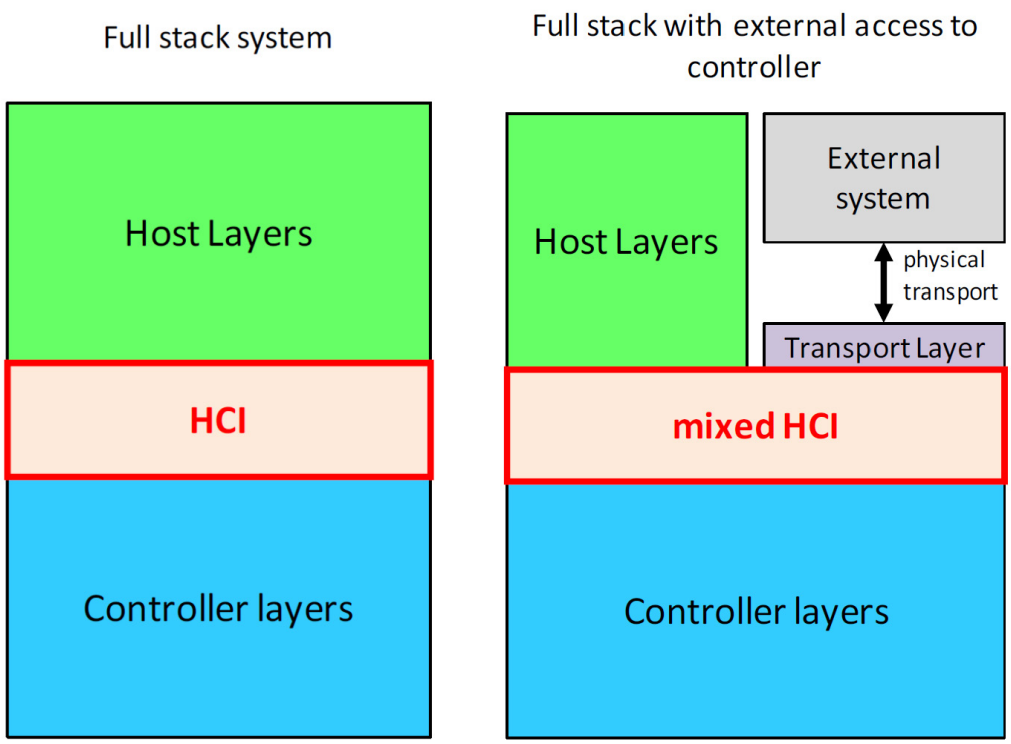


Figure 11. HCI Working Modes

RSL15 has both a full stack system, and compatibility providing an external system with aaccess to the Bluetooth Controller.

12.2.1 HCI Software Architecture

The HCI software is an interface communication block (depicted in the "HCI Software Interfaces" figure (Figure 12)) that can be used for three main purposes:

1. Communication between internal controller and external host
   - In this case, RSL15 can be used only as a controller and can communicate to an external host for verification and certification purposes or as a standalone Bluetooth Low Energy controller device (for example, connecting to a PC where an open host stack is running). It is demonstrated using the *hci* sample application from the RSL15 SDK.
2. Communication between internal controller and internal host
   - In this case, a fully embedded host and application is used. HCI cannot be used with an external UART interface.
3. Communication between internal host and external application
   - In this case, an external application communicates to the RSL15 host over UART. This interface is not based on the HCI standard (because there is no such use case or standard defined in the Bluetooth core specification). However, an external application can use the same *hci* sample application and use the same kernel messaging and format specified in GAP, GATT, L2CAP, and profile API documentation.

The only difference is that the first byte of any message sent or received over UART needs to be 0x05 (`AHI_KE_MSG_TYPE` definition of the RSL15 Bluetooth Low Energy stack).

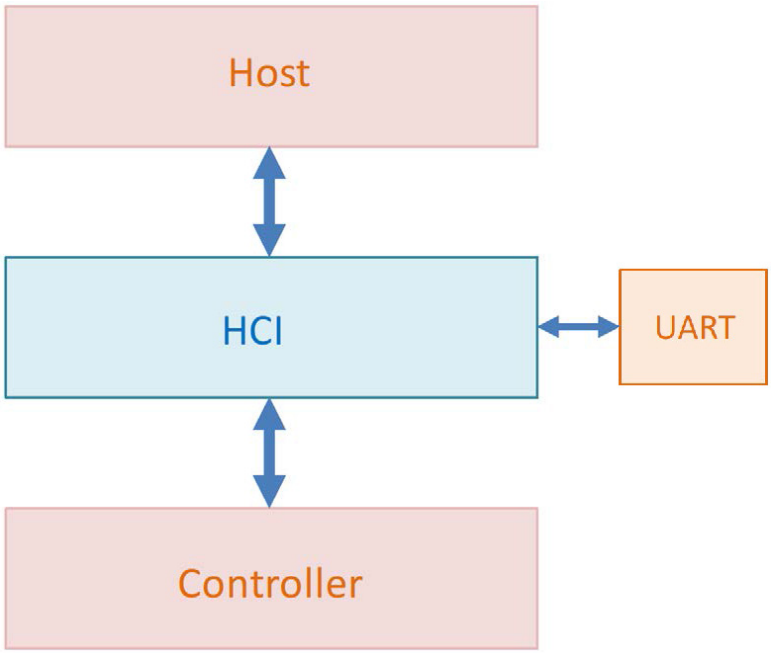


Figure 12. HCI Software Interfaces

12.3 GATT

The GATT is the gateway used by the Attribute Protocol to discover, read, write and obtain indications of the attributes present in the server attribute, and to configure the broadcasting of attributes. The GATT lies above the Attribute Protocol and communicates with the Generic Access Profile (GAP), higher layer profiles, and applications. The architecture of the GATT is shown in 12.3.

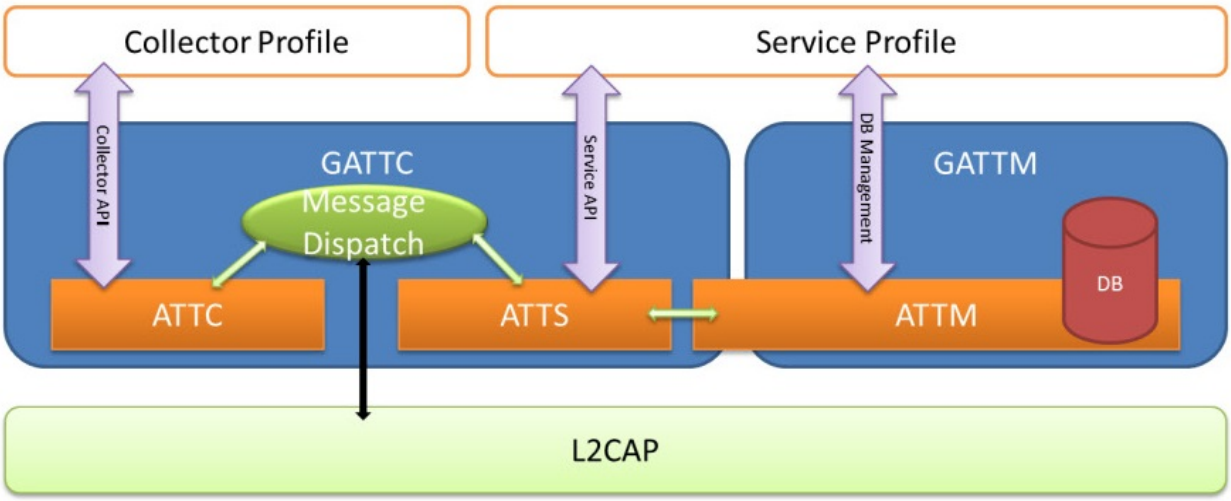


Figure 13. GATT Architecture

For more information about the GATT, and the Bluetooth Stack implementation of GATT, see the *Bluetooth Core Specification* (Volume 3, part G) and the provided CEVA *GATT Interface Specification API* document (*RW-BLE-GATT-IS.pdf*).

12.4 GAP

The Generic Access Profile (GAP) describes the generic procedures related to discovery of Bluetooth devices and link management aspects of connecting to Bluetooth devices. It also defines procedures related to use of different Bluetooth security modes.

The GAP module deals with four features:

1. Management of non-connected activities
2. Management of connected activities
3. Handling of Bluetooth Low Energy security, including pairing, bonding, encryption, and privacy.
4. Handling of life cycle of upper layer profiles

IMPORTANT: The Bluetooth standard for Bluetooth Low Energy provides several pairing schemes that can be used. Use of legacy pairing is not recommended due to known security concerns. We recommend that applications use secure connections for pairing, as per the *Bluetooth® Security and Privacy Best Practices Guide*, due to secure connection's improved overall security including substantially better MITM protection.

For more information about the GAP, and the Bluetooth Stack implementation of GAP, see the *Bluetooth Core Specification* (Volume 3, part C), the *Bluetooth® Security and Privacy Best Practices Guide*, and the provided CEVA *GAP Interface Specification API* document (*RW-BLE-GAP-IS.pdf*).

12.4.1 Non-Connected Procedures

This section describes the support provided to an application that uses non-connected procedures.

Management of these non-connected procedures is based on creation of activities representing the different available procedures. Four kind of activities can be created:

- Advertising activity
- Scanning activity
- Initiating activity
- Periodic Synchronization activity

12.4.1.1 Activity Overview

GAP API provides a set of command messages allowing to:

- Create an activity (`GAPM_ACTIVITY_CREATE_CMD`)
- Start a created activity (`GAPM_ACTIVITY_START_CMD`)
- Stop a started activity (`GAPM_ACTIVITY_STOP_CMD`)
- Delete a created activity (`GAPM_ACTIVITY_DELETE_CMD`)

A descriptions of these commands can be found in the CEVA *Gap Interface Specification*.

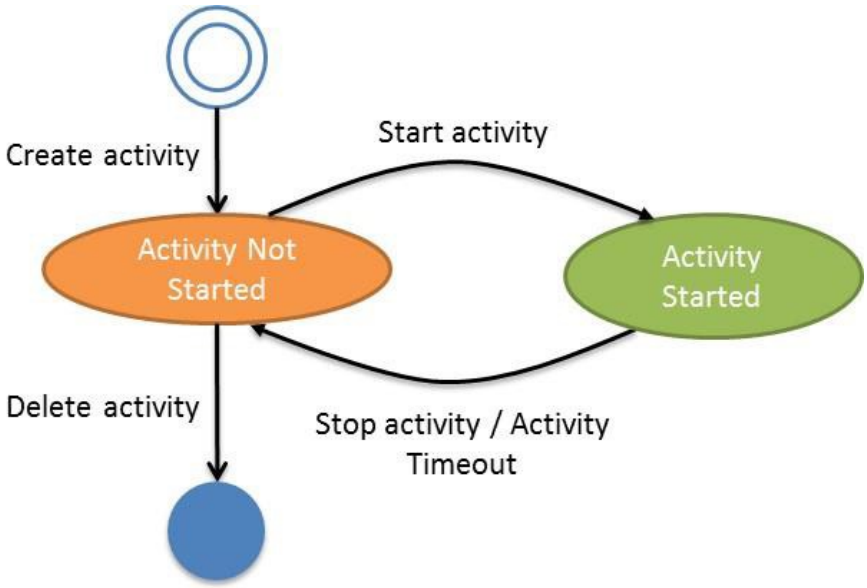


Figure 14. Activity Life Cycle

Figure 14 shows an overview of the life cycle of an activity.

- Before being usable, an activity must first be created.
- The activity must then be started.
- The activity can be considered finished after one of several events has occurred:
  - Request is received from application
  - Timeout
  - End of requested operation (after connection, synchronization,…)
- An activity can either be started again, if the application needs to perform the same procedure, or it can be deleted so that the allocated structure can be reused by another activity.

NOTE: It is not possible to directly update an activity's parameters. Instead, the application must create another activity with different parameters.

The number of activities that can be created in parallel depends on how many activities are supported by the upper layers. However, a few rules exist about the activities that can be started in parallel:

- It is possible to create and start several advertising activities in parallel.
- It is not possible to start two scanning or two initiating activities in parallel, due to HCI commands not allowing management of such operations, because management of such operations is not supported by HCI commands. However, it is possible to create two such activities in parallel.
- It is possible to start one scanning and one initiating activity in parallel with each other.

12.4.1.2 Advertising Activity

An advertising activity can be run on a device that is configured as a broadcaster device.

An advertising activity is defined by its discoverable and connectable modes, as described in Table 29.

Table 29. Advertising Activity Modes

Discoverable Mode	Connectable Mode
Non-discoverable mode: Procedure that can be limited in time. A device in this mode cannot be found by a general or limited discovery procedure. Filtering policy or targeted address can be used in this procedure. In AD_TYPE flag of advertising data, LE general and LE limited discoverable flag are set to zero.	Non-connectable mode: A device in this mode cannot be connected by a central device.
General discoverable mode: Procedure without duration limit. A device in this mode can be found by a general discovery procedure. Whitelist shall not be involved in this mode. In AD_TYPE flag of advertising data, LE general is set to 1 and LE limited discoverable flag is set to zero.	Undirected-connectable mode: A device in this mode can accept connection from any device or from device present in the whitelist.
Limited discoverable mode: Procedure with a limited duration. A device in this mode can be found either by a general or limited discovery procedure. Whitelist is not involved in this mode. In AD_TYPE flag of advertising data, LE general is set to zero and LE limited discoverable flag is set to 1.	Directed-connectable mode: A device in this mode can accept connection only by the targeted address.

In this table, the Periodic Advertising Synchronizability mode and Broadcast mode are missing.

- Periodic Advertising Synchronizability mode is neither a connectable mode nor a discoverable mode. A device in this mode sends synchronization information about periodic advertising.
- Broadcast mode is a non-connectable and non-discoverable mode.

Creation of advertising activity is possible using the `GAPM_ACTIVITY_CREATE_CMD` message, which allows creation of three different types of advertising:

- Legacy advertising activity
- Extended advertising activity
- Periodic advertising activity

NOTE: The discoverability modes such as non-discoverable mode, general discoverable mode and limited discoverable mode are considered as a property of the advertising mode.

12.4.1.2.1 Advertising Properties

The advertising properties are used to describe the content of advertising packets or behavior of the advertising activity. This section provides a small description for each of the configurable properties.

Directed

This property means that a specific device is targeted by the advertiser; the targeted device address is present in the advertising packet. For legacy advertising this applies only in connectable mode, but this is not the case for extended advertising.

High Duty Cycle

This property applies only in legacy direct advertising. The controller advertises the direct connectable packet for 1.28 s, with an advertising interval ≤ 3.75 ms.

Scannable

Advertising activity opens an RX window to receive a scan request packet, and sends in reply a scan response packet.

Connectable

Advertising activity opens an RX window to receive a connect request packet. This property is mandatory to start a connection as slave.

Anonymous

This applies only in extended advertising that is neither connectable nor scannable. The device address is not present at all in the advertised packet, but the packet can contain a targeted address.

TX Power

This applies only in extended advertising, and means that transmit power is present in an advertising packet.

Scan Request Notification

When a scan request packet is received over the air, a scan report is triggered to inform the application about observer device present over the air.

Filter Policy

This indicates if the whitelist is involved, or not to accept scan requests or connect requests. This property applies only for non-discoverable mode advertising.

12.4.1.2.2 Legacy Advertising Activity

The create legacy advertising activity operation allows an application to start legacy advertising on primary advertising channels (37, 38, and 39) at 1 Mb/s. Table 30 shows properties available for each kind of advertising mode supported for legacy advertising.

Table 30. Advertising Properties for Legacy Advertising (Primary Channel Only)

Modes/Properties **0: Must be Disabled** **1: Must be Active** **X: Can be Either Active or Disabled**	**Disc Mode**	**High Duty Cycle**	**Directed**	**Scannable**	**Connectable**	**TX Packet Type**
Non-connectable	Non-disc (Broadcaster mode)	0	0	X	0	
	Limited	0	0	X	0	
	General	0	0	X	0	
Undirected connectable	Non-disc	0	0	1	1	ADV_IND
	Limited	0	0	1	1	
	General	0	0	1	1	
Directed connectable	Non-disc	X	1	0	1	ADV_DIRECT_IND

12.4.1.2.3 Extended Advertising Activity

The create extended advertising activity operation allows an application to start extended advertising on secondary advertising channels (0 to 36) using 1 Mb/s, 2 Mb/s, or LE Coded PHY. Table 31 shows properties available for each kind of advertising mode supported for extended advertising.

NOTE: Advertising extension does not support to have both connectable and scannable modes (scannable means that advertiser replies to `AUX_SCAN_REQ`). So it is possible to set the scan response data only for non-connected modes if the scan property is set.

Table 31. Advertising Properties for Extended Advertising (Secondary Channel Usage)

Modes/Properties 0: Must be Disabled 1: Must be Active X: Can be Either Active or Disabled	Disc Mode	Anonymous	Directed	Scannable	Connectable
Non-connectable	General	0	0	X	0
	Limited	0	0	X	0
	Non-disc (Broadcaster Mode)	0	X	X	0
	Non-disc (Broadcaster Mode)	1	X	0	0
Undirected connectable	Non-disc	0	0	0	1
	General	0	0	0	1
	Limited	0	0	0	1
Directed connectable	Non-disc	0	1	0	1

12.4.1.2.4 Periodic Advertising Activity

The create periodic advertising activity operation allows an application to start a periodic advertising on secondary advertising channels (0 to 36) using 1 Mb/s, 2 Mb/s or LE Coded PHY. In order for a scanner to get information about position of the periodic advertising, a non-connectable and non-scannable extended advertising activity is started.

With this activity it is possible to set advertising data (limited to one fragment) and periodic advertising data.

Table 32 shows properties available for periodic advertising.

Table 32. Advertising Properties for Periodic Advertising (Secondary Channel Usage)

Modes/Properties 0: Must be Disabled 1: Must be Active X: Can be Either Active or Disabled	Disc Mode	Anonymous	Directed	Scannable	Connectable
Periodic Advertising Synchronizability	General	0	0	0	0
	Limited	0	0	0	0
	Non-disc	0	X	0	0

12.4.1.3 Scanning Activity

The purpose of the scanning activity is the reception of advertising packets. The Observer or Central role is mandatory for the creation of a scanning activity. Scanning activities are managed by the GAPM SCAN module.

Six scanning modes are available:

Observer Mode
A passive or an active scan procedure with non-limited duration. In this mode, the application is notified about any received advertising data, whatever its type.

Selective Observer Mode
A passive or active scan procedure with non-liminted duration using whitelist filtering.

General Discovery
A passive or an active scan procedure with a limited duration. In this mode, a device is able to discover advertiser devices broadcasting data in limited or general discoverable mode. Do not use the whitelist.

Limited Discovery
Passive or an active scan procedure with a limited duration. In this mode, a device is able to discover advertiser devices broadcasting data in limited discoverable mode. Do not use the whitelist.

General Connectable Discovery
Discover all connectable devices.

Selective Connectable Discovery
Discover connectable devices using whitelist filtering.

NOTE: Due to the content of HCI LE Extended Set Scan Param/Enable commands, a scanning activity cannot be started in parallel with another scanning activity.

Scan can be performed on LE 1M PHY or LE Coded PHY, or on both PHYs in parallel.

12.4.1.4 Initiating Activity

The purpose of initiating activity is the establishment of a Bluetooth Low Energy connection as master. This implies that support of the Central role is mandatory for creation of an initiating activity. Initiating activities are managed by GAPM INIT module.

Three connection modes are available:

Direct Connection Establishment
This procedure initiates a connection with a specific device.

Automatic Connection Establishment
This procedure makes use of the whitelist to establish a connection with known devices. The application needs to set the whitelist before starting this activity. As soon as connection with one of the whitelisted devices is established, the activity autonomously restarts the connection establishment procedure for the next device, until all desired connections have been established.

Name Discovery
As soon as connection is established, this procedure performs a read of the device name characteristic (GATT UUID 0x2A00) and then disconnects. The device name is sent to the application by using the `GAPM_PEER_NAME_IND` message.

NOTE: Based on the content of the HCI LE Extended Creation Connection command, only one initiating activity can be started at a given time.

12.4.1.5 Periodic Synchronization Activity

The periodic synchronization activity is used to perform a periodic advertising synchronization establishment procedure. Periodic synchronization activity is managed by the `GAPM PER SYNC` module. Observer mode needs to be supported for creation of a periodic synchronization activity.

To establish a synchronization, there are two possible options: waiting information from an existing link using a periodic advertising sync transfer from a peer device, or establishing synchronization without a connection. When not using a connection, it is mandatory to start a scan activity in parallel. This scan activity can be started before or after the periodic synchronization activity.

In the case of a periodic sync transfer, activity must be started with the connection index before expecting sync information from a peer device.

Once the activity has been created, it can be started using the `GAPM_ACTIVITY_START_CMD` message to synchronize with either one specific device (general type) or any of the devices present in the periodic advertising list (selective type).

NOTE: It is not allowed for two periodic synchronization activities to be waiting for synchronization at a same time.

CHAPTER 13

Bluetooth and Kernel Library

This topic describes:

- The custom application programming interface (API) for the event kernel and Bluetooth stack library
- The Bluetooth abstraction API that provides a simplified access to the Bluetooth stack library
- Using the event kernel and Bluetooth stack in an application

13.1 Use of the Event Kernel and Bluetooth Stack

Applications using the event kernel and Bluetooth Stack make use of both the custom API, described in Section 13.2 “Baseband and Kernel Functions”, and standard event kernel and Bluetooth Stack APIs, as described in the provided CEVA/RivieraWaves documentation.

IMPORTANT: To ensure Bluetooth connection stability, the Event Kernel and Bluetooth stack require that their interrupts be handled in a timely manner. For maximum Bluetooth stability, their interrupts must remain enabled throughout a connection interval and must be configured as the highest priority interrupts in the system to avoid preemption. For more information, see Section 13.2 “Baseband and Kernel Functions” on page 119.

To assist in understanding the event kernel and Bluetooth stack, the following sections outline the customary usage of these components within the context of a typical application use case.

13.1.1 The Kernel Scheduler

The kernel scheduler is responsible for constantly checking messages communicated by different tasks. Applications might use as few as one task, although more complex applications typically include several tasks. Additionally, applications can contain multiple instances of the same task.

Different application tasks control different Bluetooth Low Energy functionalities. An application sometimes needs to communicate with different tasks, such as GAPM, GAPC, GATTM, GATTC, and different standard and custom profiles (services). The kernel scheduler is responsible for handling these messages.

When the application needs to send a message, it allocates memory in the kernel buffer, fills in the message with any parameter that it wants to send, and then fills in the source address and destination address as the message identifier. The application calls a function from the kernel asking to send this message to the destination. The kernel scheduler checks that buffer. If it is filled, the scheduler sends the message to the destination task by automatically calling a pre-defined message handler. The kernel scheduler also handles all timers used by the stack, services, or application. When a timer expires, the kernel scheduler automatically calls a function or message handler allocated to the timer’s identifier.

13.1.2 Message Handlers

Coordinating with the event scheduler, the message handlers manage any request or command associated with a scheduled event.

For example, any message, request or command sent from the application to the GAPM has an associated `GAPM_COMPLETE` event. This event message is sent from the stack to the application. Based on its message identifier, this message is called automatically by the kernel.

NOTE: We recommend that users read the CEVA documentation for GATT and GAP before attempting to add their own message handler for a specific API or message.

To add a message handler, you first need to add `DEFINE_MESSAGE_HANDLER` in the corresponding *.h* file, and define a function in the corresponding file, which can be *ble_standard.h*, *application.h*, or *standard_profile.h*.

To add a message handler, users must:

1. Add a task message ID for the handler to the message enumeration used by the application.
2. Add a callback function and function prototype that executes when an event with the task message ID added in step 1 occurs.
3. Register the task message ID and callback function using:

```
MsgHandler_Add(ke_msg_id_t const msg_id, MsgHandlerCallback_t callback);
```

13.1.3 Core Bluetooth Profiles

Core Bluetooth profiles provide the standard communications structures needed to communicate between devices and to organize Bluetooth data. These core profiles are listed here:

- The Generic Access Profile (GAP) contains standard compliant implementations of the broadcasting and connecting mechanisms by which a Bluetooth Low Energy device can communicate with the outside world. The GAP implementation is divided into GAPM (GAP Manager) and GAPC (GAP Controller) services, and is described in the GAP Interface Specification.
- The Generic Attribute Profile (GATT) contains rules for how attributes (data) are formatted, packaged, and sent between Bluetooth Low Energy devices once they have a dedicated connection. The GATT implementation is divided into GATTM (GATT Manager) and GATTC (GATT Controller) services, and is described in CEVA's GATT Interface Specification.

13.1.4 Standard Profiles and Services

Standard services are pre-defined data structures and methods, which Bluetooth Low Energy devices can use to communicate Bluetooth-specific data between devices in a standardized way. Standard profiles define how a group of one or more services interact and are used.

The attributes of standard services are already recognized by the Bluetooth Low Energy stack, so you do not need to list the attributes when adding standard services to an application. Each standard service has a 16-bit UUID (unique numeric identifier).

13.1.5 Custom Profiles and Services

Custom services, like standard services, are data structures, and the methods which Bluetooth Low Energy devices can use to communicate with, and work with, your Bluetooth Low Energy applications to add functionality. Unlike standard services, custom services are not pre-defined. Users control what custom services do, creating them to meet the needs of their own applications. For custom services, a list all of the attributes needed for the service must be provided to the stack, because the stack cannot automatically tell which attributes to add. Each custom service needs a 128-bit UUID.

Custom profiles can be defined that use both standard and custom services in similar ways to standard profiles.

13.1.6 Event Kernel and Bluetooth Low Energy Library Initialization and Execution

When using the event kernel and Bluetooth stack in a user application, the application must:

- Initialize the event kernel and Bluetooth stack using the `BLE_Initialize()` function, described in Section 13.2 “Baseband and Kernel Functions”.
- Periodically execute the kernel scheduler using the `BLE_Kernel_Process()` function, described in Section 11.3 “Scheduler”.
    - For proper operation, this function must be called in the application’s main loop prior to placing the core into a state waiting for an interrupt.
    - If using low power modes, the `BLE_Baseband_Sleep()` function needs to be called after running the event scheduler before going to sleep. Only transition the device to Sleep Mode or Standby Mode if this function returns `RWIP_DEEP_SLEEP`.

NOTE: While connected to a remote device, the application must allow Bluetooth- and RF-related interrupts to be handled regularly, by not disabling and blocking interrupts for multiple Bluetooth connection periods.

13.2 Baseband and Kernel Functions

To maximize the Bluetooth Low Energy connection stability, the stack must be provided sufficient processing time. We recommended that you:

1. Set the stack interrupts to the highest priority to prevent preemption or ISR processing delays.
2. Minimize the continuous time when interrupts are disabled
3. Have the application call `BLE_Kernel_Process()` at least once per Bluetooth connection interval.

In the current sample code, all Bluetooth and non-Bluetooth interrupt priorities are set to 0 (highest priority). In future SDK updates, the priority of non-Bluetooth interrupts will be lowered. We recommend that customers make this change to their own code to lower the priority of all non-Bluetooth interrupts to maximize the Bluetooth Low Energy connection stability.

Use of the event kernel and baseband are primarily supported by the functions described in the "Event Kernel Support Functions" table (Table 33). These functions initialize and execute these components, while supporting transitions between different power modes. Function prototypes for these functions are included through *ble.h*, and these prototypes are defined in *ble/rwip.h*.

Table 33. Event Kernel Support Functions

Function	Description
BLE_Initialize	Initialize the event kernel and Bluetooth baseband for use within an application.
BLE_Baseband_Sleep	Place the Bluetooth baseband to sleep if the current Bluetooth Low Energy stack and the event kernel state indicate it is safe.
BLE_Kernel_Process	Execute any pending events that have been scheduled with the event kernel.
BLE_Baseband_Is_Awake	Check if the Bluetooth baseband is awake.

13.2.1 BLE_Initialize

Prototype:

```
void BLE_Initialize(uint8_t * param_ptr)
```

Parameters:

Table 34. BLE_Initialize Parameters

Type	Parameters	Description
`uint8_t *`	`param_ptr`	Reserved for future use (input/output parameter)

Return:

None

Description:

Initializes the Bluetooth Low Energy stack and the event kernel for use within an application.

Example

```
/* Initialize the kernel and Bluetooth stack */
uint8_t param;
BLE_Initialize(&param);
```

13.2.2 BLE_Baseband_Sleep

Prototype:

```
uint8_t BLE_Baseband_Sleep(struct ble_sleep_api_param_tag *param_ptr)
```

Parameters:

Table 35. BLE_Baseband_Sleep Parameters

Type	Parameters	Description
`struct ble_sleep_api_param_tag *`	`param_ptr`	param_ptr->app_sleep_request = Application sleep request; set to 1 if the Bluetooth baseband is intended to go to sleep when possible, 0 if the system is meant to stay awake param_ptr->max_sleep_duration = Requested maximum sleep duration in multiples of 312.5 µs (if set to zero, no maximum sleep duration is used) param_ptr->min_sleep_duration = Requested minimum sleep duration in µs param_ptr->calculated_sleep_duration = Return value, in low power baseband clock cycles, providing the calculated sleep duration

NOTE: Minimum sleep duration that `BLE_Baseband_Sleep()` sets is the maximum value of `min_sleep_duration` and the `BB_ENBPRESET_TWOSC` bit field from the `BB_ENBPRESET` register.

Return:

Allowed sleep state.

- `RWIP_DEEP_SLEEP` if the baseband controller has been put to sleep, the baseband timer is set, and the core can switch to Sleep Mode (after saving the baseband and RF registers) or Standby Mode.
- `RWIP_CPU_SLEEP` if the core can be put to sleep, but the system is not meant to enter Sleep Mode or Standby Mode.
- `RWIP_ACTIVE` otherwise.

Description:

A user application calls this function to put the system into Sleep Mode. This function performs the following operations:

1. Checks the event kernel and Bluetooth Low Energy stack to decide whether it is the right time for the system to go into Sleep Mode
2. Programs and starts the low-power BB timer (if requested)
3. Puts the BB and RF hardware into Sleep Mode (if requested)
4. Returns the calculated sleep duration and sleep decision to the application

The application uses input parameter `param_ptr->app_sleep_request` to indicate whether it requests this function to put the BB and RF hardware into Sleep Mode when possible (`0x0`: not requesting; `0x1`: requesting). In detail:

When the application calls this function with `param_ptr->app_sleep_request` set to `0x0`:

- If Sleep Mode is possible, this function writes the expected baseband sleep duration (in low-power clock cycles) in the `param_ptr->calculated_sleep_duration` field, and returns `RWIP_DEEP_SLEEP`.

  NOTE: In this case, `BLE_Baseband_Sleep()` does not put the BB and RF hardware into Sleep Mode even if itis possible, because this has not been requested.

- If Sleep Mode is not possible, this function returns `RWIP_CPU_SLEEP` or `RWIP_ACTIVE`.

When the application calls this function with `param_ptr->app_sleep_request` set to `0x1`:

- If Sleep Mode is possible, this function writes the expected baseband sleep duration (in low-power clock cycles) in the `param_ptr->calculated_sleep_duration` field, programs and starts the low-power BB timer, puts the BB and RF hardware into Sleep Mode, and finally returns `RWIP_DEEP_SLEEP`.
- If Sleep Mode is not possible, this function returns `RWIP_CPU_SLEEP` or `RWIP_ACTIVE`.

In other words, the application calls `BLE_Baseband_Sleep()` with `param_ptr->app_sleep_request` set to `0x0` when it simply wants to check if Sleep Mode is possible and to get the expected baseband sleep duration. If Sleep Mode is possible, the application then might want to check the expected baseband sleep duration and other application-specific contexts to decide if it wants to enter Sleep Mode or not. If the application decides to enter Sleep Mode, it needs to call `BLE_Baseband_Sleep()` again with `param_ptr->app_sleep_request` set to `0x1`.

Example:

```
/* Attempt to place the device to sleep */
struct ble_sleep_api_param_tag param;
param.app_sleep_request = 1;
param.max_sleep_duration = MAX_SLEEP_DURATION;
param.min_sleep_duration = MIN_SLEEP_DURATION;

switch(BLE_Baseband_Sleep(&param))
{
    case RWIP_DEEP_SLEEP:
    {
        /* Save the baseband and RF registers, then go to sleep */
        BB_Sleep(POWER_MODE);
        break;
    }
    case RWIP_CPU_SLEEP:
    {
        /* Wait for interrupt */
        __WFI();
        break;
    }
    case RWIP_ACTIVE:
    default:
    {
    }
}
```

13.2.3 BLE_Kernel_Process

Prototype:

```
void BLE_Kernel_Process(void)
```

Parameters:

None

Return:

None

Description:

Execute any pending events that have been scheduled with the event kernel.

Example:

```
/* Main application loop:
 * - Run the kernel scheduler
 * - Refresh the watchdog and wait for an interrupt before continuing */
while (1)
{
    BLE_Kernel_Process();

    /* Refresh the watchdog timer */
```

```
    SYS_WATCHDOG_REFRESH();

    /* Wait for an event before executing the scheduler again */
    __WFI();
}
```

13.2.4 BLE_Baseband_Is_Awake

Prototype:

```
bool BLE_Baseband_Is_Awake(void)
```

Parameters:

None

Return:

True if the Bluetooth baseband is awake, false otherwise.

Description:

Check if the Bluetooth baseband is awake.

Example:

```
/* Check if the Bluetooth baseband is still awake */
if (BLE_Baseband_Is_Awake())
{
    BLE_Baseband_Sleep(&param);
}
```

13.3 Managing Bluetooth Low Energy Stack RAM Usage

The Bluetooth Low Energy stack is provided as a complied library and cannot be modified by the user. However, some of the stack variables are defined at the application level, and the user has some control over the size of these variables. This is a guide to optimizing stack RAM memory usage by adjusting the application level variable defines as dependent upon the application use case.

The application level variables are defined to support the maximum number of connections and activities, which is 10 and 11 respectively. (See the CEVA documentation provided with your RSL15 download, in the default location *C:\Users***<user_name>***\ON_Semiconductor\PACK\ONSemiconductor\RSL15***<version>***\documentation\cev*a.) If the application use case does not require the maximum number of connections and activities, application level variable sizes can be reduced to save memory.

NOTE: It is not recommended to adjust application level variable sizes in the HCI stack, as this can adversely affect Bluetooth certification.

The simplest method and recommended starting point to reduce stack memory usage is to reduce the value of `APP_MAX_NB_CON`. This scales most of the other application level variables as well, resulting in a reduction of memory usage. This is usually sufficient for most users. If you are an advanced user requiring further optimization, read on.

The memory allocated by the stack is divided into different parts. The first part comprises the required environment and global variables that are independent of the number of connections or activities. This part cannot be changed by developers.

The second part is heap memory, which the kernel takes as a global variable (*.bss* section), and later the kernel and the Bluetooth Low Energy stack use it as the heap memory for their procedures. It is divided into four parts:

- Environment variables
- Message heap memory
- Data base memory
- Non-retention memory

By default, all parameters are set to address the maximum number of connections and activities, which are 10 and 11 respectively. If definition `APP_HEAP_SIZE_DEFINED` is defined in the *app.h* file of any Bluetooth Low Energy application, then the heap sizes can be customized in the *ble_protocol_support.c* file (where they are defined and allocated). The following code example shows how the required memory sizes can be calculated (defined in *ble_protocol_support.c*).

```
/// Memory allocated for environment variables
uint32_t rwip_heap_env[RWIP_CALC_HEAP_LEN(APP_RWIP_HEAP_ENV_SIZE)];
/// Memory allocated for Attribute database
uint32_t rwip_heap_db[RWIP_CALC_HEAP_LEN(APP_RWIP_HEAP_DB_SIZE)];
/// Memory allocated for kernel messages
uint32_t rwip_heap_msg[RWIP_CALC_HEAP_LEN(APP_RWIP_HEAP_MSG_SIZE)];
/// Non Retention memory block
uint32_t rwip_heap_non_ret[RWIP_CALC_HEAP_LEN(APP_RWIP_HEAP_NON_RET_SIZE)];
```

We assume:

`APP_MAX_NB_ACTIVITY` and `APP_MAX_NB_CON` are defined in the application, based on the application's use case. Please note that `PP_BLE_CONNECTION_MAX` needs to be at least one value bigger than `APP_MAX_NB_CON`.

```
APP_RWIP_HEAP_ENV_SIZE =
(600 + (APP_MAX_NB_ACTIVITY) * 230)
     + APP_MAX_NB_CON * ((sizeof(struct gapc_env_tag)
     + KE_HEAP_MEM_RESERVED)
     + (sizeof(struct gattc_env_tag)  + KE_HEAP_MEM_RESERVED)
     + (sizeof(struct l2cc_env_tag)   + KE_HEAP_MEM_RESERVED))
     + ((APP_MAX_NB_ACTIVITY ) * (sizeof(struct gapm_actv_scan_tag)
                                     + KE_HEAP_MEM_RESERVED))
```

`APP_RWIP_HEAP_DB_SIZE`: This depends on how much data base memory (GATT services) is added. The default value is 3072 bytes, but developers can choose a smaller value based on their use case. This stack memory heap (section) is used for several purposes:

- GAP and GATT database: total size 158 B
- Database of Bluetooth Low Energy Profiles. Considering that only BAS and DIS are available, the required sizes are as follows, providing an application requests adding them:
    - DISC: 694 B
    - DISS: 154 B
    - BASC: 392B

- BASS: 70 B

  NOTE: BASC and BSS can be one instance or two instances. Depending how many an application adds, the used memory size increases by two if two instances are configured.

- Database for custom services. Each service needs 22 B for service itself. For each attribute it needs a maximum of 22 B in addition, depending on attribute types. We recommend that you consider the maximum value.
- L2CAP credit-based connection-oriented channels environment (LECB): this needs 36 B per channel.

```
APP_RWIP_HEAP_MSG_SIZE =
(1650 + 2 * ((16 + (APP_MAX_NB_ACTIVITY - 1) * 56)
      + (58 + (APP_MAX_NB_ACTIVITY - 1) * 26)
      + ((APP_MAX_NB_ACTIVITY) * 66)
      + ((APP_MAX_NB_ACTIVITY) * 100)
      +((APP_MAX_NB_ACTIVITY) * 12)))
      +(((BLEHL_HEAP_MSG_SIZE_PER_CON * APP_MAX_NB_CON) > BLEHL_HEAP_DATA_THP_SIZE)
                                                        ?
       (BLEHL_HEAP_MSG_SIZE_PER_CON * APP_MAX_NB_CON) : BLEHL_HEAP_DATA_THP_SIZE)
```

`APP_RWIP_HEAP_NON_RET_SIZE`: By default, this is set to 656 bytes. It is used for security algorithm calculations. This part of memory does not need to be in retention mode when the use case calls for keeping the Bluetooth Low Energy link active and going to sleep with VDDM in retention. If enough memory is allocated — for example, in the database — the kernel can allocate this part of memory from another heap. Developers need to make sure that this default amount of memory is available, in non-ret heap or in another heap.

In this way, users can set their own `APP_MAX_NB_CON` and `APP_MAX_NB_ACTIVITY` values, and decrease the memory size allocated by default settings.

13.4 Bluetooth Low Energy Link Layer Metrics

A user application can enable or disable the tracking of metrics for Bluetooth Low Energy links. This is accomplished by calling the following function, with the `en` input parameter set to `0x1` or `0x0`, respectively:

```
 uint8_t BLE_Link_Metrics(uint8_t en, uint8_t actidx, struct ble_link_metrics
*metricsPtr)
```

We recommend using this function to enable the link metrics tracking when a Bluetooth Low Energy link is established (i.e., when `GAPC_CONNECTION_REQ_IND` is received), and disable the tracking at Bluetooth Low Energy disconnection (i.e., when `GAPC_DISCONNECT_IND` is received).

Example code for the `BLE_Link_Metrics()` function:

```
struct ble_link_metrics bleLinkMetrics[APP_MAX_NB_CON];
...

/* At receiving GAPC_CONNECTION_REQ_IND */

memset(&bleLinkMetrics[conidx], 0, sizeof(struct ble_link_metrics));
ble_metrics_ret_val = BLE_Link_Metrics(1, p->conhdl, &bleLinkMetrics[conidx]);
...

/* At receiving GAPC_DISCONNECT_IND */
```

```
 ble_metrics_ret_val = BLE_Link_Metrics(0, gap_env.connection[conidx].conhdl,
&bleLinkMetrics[conidx]);
 ...
```

This example shows data structure `ble_link_metrics` used to track the metrics for a Bluetooth Low Energy link:

```
 struct ble_link_metrics
 {
      uint8_t enable;
      uint32_t noerr_pkt_non_empty_cnt;
      uint32_t noerr_pkt_empty_cnt;
      uint32_t sync_err_cnt;
      uint32_t rxtime_err_cnt;
      uint32_t len_err_cnt;
      uint32_t crc_err_cnt;
      uint32_t mic_err_cnt;
      uint32_t llid_err_cnt;
      uint32_t sn_err_cnt;
      uint32_t nsen_err_cnt;
      uint32_t rxcte_err_cnt;
 };
```

The "ble_link_metrics Field Names and their Meanings" table (Table 36) shows the meaning of each field in the `ble_link_metrics` data structure:

Table 36. ble_link_metrics Field Names and their Meanings

Field Name	Contents
enable	Enable (`0x1`) or disable (`0x0`) the tracking of link metrics
noerr_pkt_non_empty_cnt	The number of received non-empty packets without any errors
noerr_pkt_empty_cnt	The number of received empty packets without any errors
sync_err_cnt	The number of received packets with Sync Word not correct or not detected
rxtime_err_cnt	The number of received packets with RX time greater than a threshold configured by the stack till the last CRC bit
len_err_cnt	The number of received packets with lengths greater than the maximum value allowed by the event configured by the stack
crc_err_cnt	The number of received packets with incorrect CRC
mic_err_cnt	The number of received packets with incorrect MIC
llid_err_cnt	The number of received packets with invalid LLID
sn_err_cnt	The number of received packets with incorrect sequence number (SN)
nsen_err_cnt	The number of received packets with incorrect next expected sequence number (NESN)
rxcte_err_cnt	The number of received packets with incomplete reception of CTE field to indicate an abort of the reception during the CTE field

When a packet is received at the link layer of the Bluetooth Low Energy stack, if the tracking of the associated link is enabled (whether the received packet is erroneous or not), the counters in `ble_link_metrics` data structure associated with the link are updated using the logic below:

```
if(rxstatus & EM_BLE_SYNC_ERR_BIT)
{
    sync_err_cnt++;
}
else if(rxstatus & EM_BLE_RXTIME_ERR_BIT)
{
    rxtime_err_cnt++;
}
else if(rxstatus & EM_BLE_LEN_ERR_BIT)
{
    len_err_cnt++;
}
else if(rxstatus & EM_BLE_CRC_ERR_BIT)
{
    crc_err_cnt++;
}
else if(rxstatus & EM_BLE_MIC_ERR_BIT)
{
    mic_err_cnt++;
}
else if(rxstatus & EM_BLE_LLID_ERR_BIT)
{
    llid_err_cnt++;
}
else if(rxstatus & EM_BLE_SN_ERR_BIT)
{
    sn_err_cnt++;
}
else if(rxstatus & EM_BLE_NESN_ERR_BIT)
{
    nsen_err_cnt++;
}
else if(rxstatus & EM_BLE_RXCTEERR_BIT)
{
    rxcte_err_cnt++;
}
else
{
    if(rxLength == 0)
    {
        noerr_pkt_empty_cnt++;
    }
    else
    {
        noerr_pkt_non_empty_cnt++;
    }
}
```

13.4.1 BLE_Link_Metrics

```
 uint8_t BLE_Link_Metrics(uint8_t en, uint8_t actidx, struct ble_link_metrics
*metricsPtr)
```

Location: lld_con.c: 3643

This function enables or disables the tracking of Bluetooth Low Energy link metrics.

Returns:

Return value from this function indicates if the requested action (enable or disable) has been performed successfully.

Parameters:

Direction	Name	Description
in	en	0x1to enable the tracking of link metrics, 0x0 to disable it
in	actidx	Activity identifier of Bluetooth Low Energy link
in	metricsPtr	Pointer to the ble_link_metrics data structure, which maintains link metric counters
out	Return value	Returns status and indicates if requested action (enable/disable) has been performed successfully • 0x0 if requested action has been performed successfully • 0x1 if activity identifier of Bluetooth Low Energy link is invalid • 0x2 if the pointer to ble_link_metrics data structure is NULL

13.5 Bluetooth Stack Supporting API Functions

13.5.1 Device_BLE_Param_Get

Prototype:

```
uint8_t Device_BLE_Param_Get(uint8_t param_id, uint8_t *lengthPtr, uint8_t *buf)
```

This is a callback function that is called by the Bluetooth stack whenever the associated data is required. If the application returns PARAM_OK, it means that requested parameter is provided by the application; if it returns PARAM_FAIL, it means the stack can use the default settings/parameters.

In this function there is a switch/case, described below:

- PARAM_ID_BD_ADDRESS: the global public address of the device
- PARAM_ID_LPCLK_DRIFT: the clock accuracy in ppm for the low power clock. There are three clock sources: XTAL32, RC OSC 32, and clock input from a GPIO[0- 3]. By default it is 500 ppm, but the application can provide another value based on the clock accuracy used. For the central role it is not expected to be bigger than 500.
- PARAM_ID_ACTCLK_DRIFT: the clock accuracy of XTAL 48 MHz in ppm that depends on the XTAL component used in PCB. If it is not provided by application maximum, 50 ppm will be used.
- PARAM_ID_OSC_WAKEUP_TIME: the time that, after wakeup from a low power mode, XTAL48 and other blocks of need time to power up and be ready. The value is in µs.
- PARAM_ID_CH_ASS_EN: can be used in a central role to indicate if the channel assessment mechanism needs to be executed by the stack or not. It has True and False values.

- PARAM_ID_LPCLK_NO_XTAL32K: determines if a XTAL 32768 Hz is used as the low power clock source or another option of clock source/value is used. If 32768 Hz with maximum 500 ppm is not used, then the actual clock value can be provided by calling LPCLK_PeriodValue_Set(LPCLK_PERIOD_VALUE); refer to sample application code.
- PARAM_ID_LE_PRIVATE_KEY_P256 and PARAM_ID_LE_PUBLIC_KEY_P256: if ECC public and private keys are provided by the application, this can be used for debugging purposes or for use cases where an application needs to generate the keys by itself (using the Arm CryptoCell-312 hardware accelerator or any other way) rather than having them generated by the Bluetooth stack.
- PARAM_ID_LE_DBG_FIXED_P256_KEY: values are True or False, indicating if the stack should use forced keys provided by application or not.
- PARAM_ID_CH_ASSESS_PARAMS: when a channel assessment algorithm is enabled by the application, the related parameters can be provided through this case that need to follow struct channel_map_assess_tag.
- PARAM_ID_DTM_ANT_ID_TO_PTRN_PARAMS: for antenna-switching use cases, the antennal pattern table can be provided through this case.
- PARAM_ID_CUSTOMIZED_HEAP_SIZE: if an application needs to control and define the memory size used by the Bluetooth stack (stack heap), the four heap sizes values can be provided; otherwise the stack will use default sizes. (Refer to "Introduction" on page 103 for details).
- PARAM_ID_DFT_SLAVE_MD: When a peripheral device sends a packet, it sets the MD bit to make sure that an acknowledge can be received from the central peer device in the same connection event. It has true and false values. By default, it is set to true, which means that the MD bit is set.
- PARAM_ID_ACTIVITY_MOVE_CONFIG: When there is more than one Bluetooth Low Energy connection, events anchor points might overlap and a peripheral device can send a connection parameter update request to peer central devices asking for an anchor point move. By default, it is enabled (true). It can be disabled if the application sets it to false.
- PARAM_ID_LE_CODED_PHY_500: In the RSL15 *peripheral_server* sample project, setting ADV_EXTENSION =1 causes the application to use GAPM_PHY_TYPE_LE_CODED for advertising at a default rate of 250 Kbps. But if 500 Kbps is required instead, then this case is used.

13.5.2 platform_reset

Prototype:

```
void platform_reset(uint32_t error)
```

This function is called whenever the stack memory is full, so that the application can be notified of the situation and handle it accordingly.

13.5.3 srand_func

Prototype:

```
void srand_func(uint32_t seed)
```

This provides a seed value for the rand() function used in the stack.

13.5.4 rand_func

Prototype:

```
int rand_func(void)
```

Instead of the C `rand` function, the application can provide its own implementation or use the TRNG accelerator from the Arm CryptoCell-312.

13.5.5 Device_RF_RSSI_Convert

Prototype:

```
int8_t Device_RF_RSSI_Convert(uint8_t rssi_reg)
```

This function can be used by application to convert or readthe RF RSSI register to an actual dBm value. A general function is provided, but RSSI needs calibration for use cases that need an more accurate RSSI.

13.5.6 Device_RF_TxPwr_Get_dBm

Prototype:

```
int8_t Device_RF_TxPwr_Get_dBm(uint8_t txpwr_idx)
```

This callback function is called whenever the stack needs to convert the `RF0_REG1A_PA_PWR_PA_PWR_BYTE` field of register `RF0_REG1A` to a dBm value.

13.5.7 Device_RF_TxPwr_Get_Idx

Prototype:

```
uint8_t Device_RF_TxPwr_Get_Idx(int8_t txpwr_dbm)
```

This callback function is used whenever the stack is converting output power in dBm to a value used to set the `RF0_REG1A_PA_PWR_PA_PWR_BYTE` field of register `RF0_REG1A`.

13.5.8 BLE_Set_RxStatusCallBack

Prototype:

```
void App_RxStatus_Callback(uint8_t *actidx, uint16_t *status, uint8_t *rssi, uint8_t
*chnl, uint16_t *length)
```

Parameters:

- *actidx
- *status
- *rssi
- *chnl
- *length

Return:

The returned RSSI is a raw RSSI register read value and needs to be converted to dBm.

Description:

This API can register an application callback function, so that whenever a packet is received, the Bluetooth Low Energy stack calls that callback function and returns the listed parameters as arguments of that function.

Examples:

```
 /* registering a callback function any time in application after Bluetooth Low Energy
initialization */

 BLE_Set_RxStatusCallBack(App_RxStatus_Callback);

 /*callback function:*/

 void App_RxStatus_Callback(uint8_t *actidx, uint16_t *status, uint8_t *rssi, uint8_t
*chnl, uint16_t *length)
 {

 int8_t calculated_rssi = ((0.328 * (*rssi)) - 108);

     if(*status == 0)

 {         swmTrace_printf("\n\r actidx = %d, status = %d, rssi = %d, chnl= %d, length =
%d", *actidx, *status, calculated_rssi, *chnl, *length);    }

 }

 /* When this functionality is required another API needs to be called before BLE_
Initialize() is called, as shown below: */

 BLE_Set_RFOffSeqMode(1);
 uint8_t param_ptr;

 BLE_Initialize(&param_ptr);
```

This information is only valid if the status of the received packet is zero (no error). The callback function does not consume a great deal of processing time, as it is called in the Bluetooth Low Energy RX ISR.

A rough estimation formula can be used as shown below, but in general the RSSI needs to be calibrated to take into account sample-to-sample variation, and channel variation that depends on PCB, XTAL48 trimming, and antenna/matching circuit design. Changes in temperature can also cause variation.

13.5.9 Device_RF_SetMaxPwrIdx

Prototype:

```
 uint8_t Device_RF_SetMaxPwrIdx(void)
```

This API/callback function is called by the Bluetooth Low Energy stack to set a maximum power index, which is later used for non-advertisement activities powered by TX output.

The return value is based on the format that function `Device_RF_TxPwr_Get_Idx(desired dBm value)` returns.

Setting `adv_param.max_tx_pwr` to any value between -127 and 126 causes the stack to call `Device_RF_TxPwr_Get_Idx()`, and the return index to this function determines the actual ADV TX power.

However, for connections, scan response, and initiating activities, the stack uses the value that is provided in advance by the application during the stack initialization when `Device_RF_SetMaxPwrIdx()` is called.

Thus, applications can control ADV TX powers through `adv_param.max_tx_pwr`, and output power levels for connections are controlled by `Device_RF_SetMaxPwrIdx()`.

13.5.10 BLE_Set_ScanConIndStatusCallBack

Prototype:

```
BLE_Set_ScanConIndStatusCallBack(App_ScanConIndStatus_Callback);
```

Parameters:

none

Return:

The returned RSSI is a raw RSSI register read value and needs to be converted to dBm.

Description:

Through this function, it is possible to register an application callback function to access the channel number and RSSI for any successfully received scan request and connection indication. It requires that `BLE_Set_RFOffSeqMode(1)` be called in advance.

Example:

```
 BLE_Set_ScanConIndStatusCallBack(App_ScanConIndStatus_Callback);

 void App_ScanConIndStatus_Callback(uint8_t *type, uint8_t *rssi, uint8_t *chnl)

 {

    int8_t calculated_rssi = ((0.328 * (*rssi)) - 108);
    if(*type == BLE_SCAN_REQ)

 {        swmTrace_printf("\n\r  SCAN_REQ, rssi = %d, chnl= %d", calculated_rssi,
*chnl);      }

     else if(*type == BLE_CONNECT_IND)

 {          swmTrace_printf("\n\r  CONNECT_IND, rssi = %d, chnl= %d", calculated_rssi,
*chnl);      }

 }
```

13.5.11 Hci_Vs_Cmd_App_Func

Prototype:

```
 uint8_t Hci_Vs_Cmd_App_Func(uint8_t cmd_code,
uint8_t length, uint8_t *data_buf,
uint8_t *result_length, uint8_t *result_data)
```

Parameters:

- cmd_code
- length
- *data_buf
- *result_length
- *result_data

Return:

Status.

Description:

When running an hci application that uses the HCI variant of the Bluetooth Low Energy stack to support HCI commands over UART, `HCI_DBG_VS_APP_CMD_OPCODE` needs to be implemented in the application. This allows users to develop any desired vendor-specific command. The format of the message is handled by the stack, and the application needs to have a function implemented such that it can interpret the command code, input parameters length, and data, and sends a response with the status, length and data. Some of the command codes are used by RF tools, so users can add unused command codes.

Example:

```
 uint8_t Hci_Vs_Cmd_App_Func(uint8_t cmd_code,
 uint8_t length, uint8_t *data_buf,
 uint8_t *result_length, uint8_t *result_data)
 {
 uint8_t status = CO_ERROR_NO_ERROR;
 *result_length = 0;
 uint16_t freq_Mhz;
 int8_t pwr_dBm, txOrRx; /* 0: Tx, 1:Tx */

 switch(cmd_code)
 { case HCI_VS_RF_CW_ENABLE_CMD_CODE: txOrRx = data_buf[0]; freq_Mhz = (data_buf[1] +
(data_buf[2] << 8)); //TODO break;
 case HCI_VS_RF_CW_DISABLE_CMD_CODE: //TODO break; case HCI_VS_RF_OUTPUT_PWR_CMD_CODE:
pwr_dBm =
 (int8_t)data_buf[0]; //TODO To be replace with HAL RF set output power function break;
default: status =
 CO_ERROR_INVALID_HCI_PARAM; break; }

 return(status);

 }
```

The above commands are implemented and can be found in the *hci* sample code.

13.5.12 BLE_Set_MaxRFConnectionPwr

Prototype:

```
 void BLE_Set_MaxRFConnectionPwr(void)
```

Parameters:

None

Return:

None

Description:

This API can be used by an application to set the maximum RF transmission output power for Bluetooth Low Energy connections. It must be invoked before a connection is established (RF transmission power cannot be changed during a connection and therefore calling this API during a connection does not change the power). This API calls `Device_RF_SetMaxPwrIdx()`, where it uses the `rf_tx_power_dbm` field of the `ble_dev_params` data structure; therefore, this `rf_tx_power_dbm` field must be set to the desired output power (dBm value) in advance by the application before a new connection is made.

NOTE: Setting the VDDPA Enable or Disable configuration is the application's responsibility The effect that the `RF0_REG1A_PA_PWR_PA_PWR_BYTE` field of the `RF0_REG1A` register has on output power is this API's responsibility.

Example:

```
/* Before a new Bluetooth Low Energy connection is established */

/* Set RF TX power to -5 dBm */
ble_dev_params.rf_tx_power_dbm = -5;
BLE_Set_MaxRFConnectionPwr();
```

13.5.13 uint32_t ke_get_max_mem_usage

Prototype:

```
uint32_t ke_get_max_mem_usage(void)
```

Parameters:

None

Return:

Maximum memory used by the Bluetooth stack in the application

Description:

The maximum memory that the Bluetooth stack has used in the application at any time can be returned using this API.

Example:

```
/* Get the Maximum memory used by the Bluetooth stack in the application */
uint32_t max_mem_usage_stack = ke_get_max_mem_usage();
```

13.5.14 uint16_t ke_get_mem_usage

Prototype:

```
uint32_t ke_get_max_mem_usage(void)
```

Parameters:

type : Kernel memory heap type which can be set using one of the following arguments

- `KE_MEM_ENV`: Memory allocated for environment variables
- `KE_MEM_ATT_DB`: Memory allocated for Attribute database
- `KE_MEM_KE_MSG`: Memory allocated for kernel messages
- `KE_MEM_NON_RETENTION`: Non-retention memory block

Return:

The amount of memory allocated for the specified Bluetooth stack heap type at the time of calling this API

Description:

The amount of memory that is allocated for the Bluetooth stack heap type of environment variables, attribute database, kernel messages, or non-retention memory block can be returned using this API.

Example:

```
/* Memory heap allocated for environment variables */
Uint32_t mem_env = ke_get_mem_usage(KE_MEM_ENV);
/* Memory heap allocated for Attribute database */
Uint32_t mem_env = ke_get_mem_usage(KE_MEM_ATT_DB);
/* Memory heap allocated for kernel messages*/
Uint32_t mem_env = ke_get_mem_usage(KE_MEM_KE_MSG);
/* Memory heap allocated for non-retention memory block */
Uint32_t mem_env = ke_get_mem_usage(KE_MEM_NON_RETENTION);
```

13.6 Bluetooth Low Energy Abstraction

The Bluetooth Low Energy abstraction is a wrapper for the Bluetooth APIs provided by CEVA for the RSL15 device. This provides a simplified API to access the Bluetooth components, which can be included through the *ble_abstraction.h* header. Component wrappers include:

Generic Access Profile (GAP)
: This profile defines the generic procedures related to discovery of Bluetooth devices (idle mode procedures), link management aspects of connecting to Bluetooth devices (connection mode procedures), and management of security procedures.

  As defined in *ble_gap.h* / *ble_gap.c*, this portion of the Bluetooth abstraction API extends the GAP support described in *RW-BLE-GAP-IS.pdf.*

Generic Attribute Profile (GATT)
: This profile provides a service framework using the Bluetooth attribute protocol for discovering services, and for reading and writing characteristic values on a peer device.

As defined in *ble_gatt.h* / *ble_gatt.c*, this portion of the Bluetooth abstraction API extends the GATT support described in *RW-BLE-GATT-IS.pdf.*

Protocol Support

As defined in *ble_protocol_support.h* / *ble_protocol_support.*c, this portion of the Bluetooth abstraction API provides access to support components that are used with the Bluetooth stack. This includes:

- The Bluetooth address
- The RF front-end transmit power configuration and received signal strength indication (RSSI)
- Bluetooth parameters
- Random number generation

Bondlist

Support for managing bonding information used by the Bluetooth GAP interface. This includes storage and access to the device addresses and privacy-related identity resolving keys (IRKs) for any devices bonded to this RSL15 device.

This support is defined in *bondlist.h* / *bondlist.c*, and is used with the Bluetooth GAP support.

Message Handler

This block extends the event kernel message handling, described in Section 11.2 “Messages” on page 90. The Bluetooth abstraction API for this support is defined in *msg_handler.h* / *msg_handler.c*.

The Bluetooth Low Energy abstraction is configured by adding the `ble_abstraction` to an application using the *RTE_Device.h* file, as described in Section 1.1 “Configuring The Driver Run-time Environment” on page 1.

CHAPTER 14

swmTrace Library

The swmTrace library is a complement to the RTT Viewer plugin described in the Diagnostic Strategies chapter of the *RSL15 Developer's Guide*. The swmTrace library is a logging utility, intended to provide debugging capabilities through an application running on the Arm Cortex-M33 core. When using the swmTrace library's functions with the RTT viewer, the logging information shown by the RTT Viewer is color-coded, based on the level of logging used and the particular function calls to the swmTrace library.

14.1 Introduction

Although this library's expected primary use case is with the SEGGER RTT technology through the SDK's RTT Viewer, you can also use the library with UART simply by linking to a different library variant. There are four library variants that can be selected between without changing the underlying code:

- RTT variants include both blocking and non-blocking debug.
- UART variants include support for DMA and user defined pins for UART, TX and RX, where both variants are non-blocking.

14.2 onsemi IDE Setup

In the onsemi IDE, the swmTrace Library requires that you set at least one of the following configurations in **Properties** > **C/C++ Build** > **Settings** > **Cross ARM C Linker** > **Miscellaneous**:

- Check the **newlib-nano** checkbox to enable newlib-nano.
- Check the **Do not use syscalls** checkbox.
- In the text-box labelled **Other linker flags**, enter `–specs=rdimon.specs`

We strongly recommend that you set up the first configuration enabling newlib-nano, regardless of whether you set the other two. newlib-nano is enabled by default in all RSL15 sample projects. Enabling newlib-nano in the onsemi IDE also minimizes the flash and RAM requirements of applications.

14.3 Usage

Various levels of logging are available. See *swmTrace_api.h* for full details, but examples include `SWM_LOG_LEVEL_VERBOSE` and `SWM_LOG_TEST_PASS`. All levels of logging use the `SWM_LOG_` prefix.

Sample applications are available specifically to illustrate the usage of the swmTrace library. See *swmTrace_logger* under the sample application folder. Other sample applications also make use of the swmTrace library.

CHAPTER 15

CMSIS Reference

Hardware register abstraction layer for the SOC.

15.1 SUMMARY

Variables

- RSL15_Sys_Version : RSL15 firmware version (variable)
- __Heap_Begin__ : Start location for the heap.
- __Heap_Limit__ : Top limit for the heap.
- __stack_limit : Bottom limit for the stack.
- __stack : Start location for the stack.
- __data_init__ : Pointer to the data to used to initialize volatile memory.
- __data_start__ : Start address of the initialized data area in volatile memory.
- __data_end__ : End address of the initialized data area in volatile memory.
- __bss_start__ : Start address of the cleared data area in volatile memory.
- __bss_end__ : End address of the cleared data area in volatile memory.
- __preinit_array_start__ : Weakly defined function list pointer for pre-initialization functions.
- __preinit_array_end__ : Weakly defined pointer to the end of the pre-initialization function list.
- __init_array_start__ : Weakly defined function list pointer for initialization functions.
- __init_array_end__ : Weakly defined pointer to the end of the initialization function list.
- flash_layout : Flash layout for the RSL15 device.
- SystemCoreClock : Contains the current SYS_CLK frequency, in Hz.

Data Structures

- flash_region : Structure used to define flash regions.

Macros

- RSL15_SYS_VER_MAJOR : RSL15 header file major version.
- RSL15_SYS_VER_MINOR : RSL15 header file minor version.
- RSL15_SYS_VER_REVISION : RSL15 header file revision version.
- RSL15_SYS_VER : RSL15 firmware version.
- __ARMv8MML_REV : Arm v8 architecture revision.
- __CM33_REV : Core revision r0p4.
- __FPU_PRESENT : FPU present.
- __DSP_PRESENT : DSP extension present.
- __SAUREGION_PRESENT : SAU regions present.
- __MPU_PRESENT : MPU present.
- __VTOR_PRESENT : VTOR present.
- __NVIC_PRIO_BITS : 3 bits used for interrupt priority levels
- __Vendor_SysTickConfig : Standard SysTick configuration is used.
- I2C_REF_VALID : Validation of I2C register block pointer reference for assert statements.

- LIN_REF_VALID : Validation of LIN register block pointer reference for assert statements.
- PCM_REF_VALID : Validation of PCM register block pointer reference for assert statements.
- PWM_REF_VALID : Validation of PWM register block pointer reference for assert statements.
- SPI_REF_VALID : Validation of SPI register block pointer reference for assert statements.
- UART_REF_VALID : Validation of UART register block pointer reference for assert statements.
- TIMER_REF_VALID : Validation of TIMER register block pointer reference for assert statements.
- DMA_REF_VALID : Validation of DMA register block pointer reference for assert statements.
- FLASH_REF_VALID : Validation of FLASH register block pointer reference for assert statements.
- GPIO_PAD_COUNT : GPIO peripheral definitions.
- GPIO_GROUP_LOW_PAD_RANGE : Number of GPIO pads in the lowest group (all)
- GPIO_EVENT_CHANNEL_COUNT : Number of available GPIO interrupts.
- GPIO0 : GPIO pads definitions
- GPIO1 : GPIO 1.
- GPIO2 : GPIO 2.
- GPIO3 : GPIO 3.
- GPIO4 : GPIO 4.
- GPIO5 : GPIO 5.
- GPIO6 : GPIO 6.
- GPIO7 : GPIO 7.
- GPIO8 : GPIO 8.
- GPIO9 : GPIO 9.
- GPIO10 : GPIO 10.
- GPIO11 : GPIO 11.
- GPIO12 : GPIO 12.
- GPIO13 : GPIO 13.
- GPIO14 : GPIO 14.
- GPIO15 : GPIO 15.
- SYS_DUMMY_READ : Register that always reads back as 0x00000000.
- SYS_DUMMY_WRITE : Register to which writes are ineffective.
- ERRNO_NO_ERROR : No error.
- ERRNO_GENERAL_FAILURE : General error.
- DEFAULT_FREQ : High speed main RC oscillator default frequency set by boot ROM application Default value is 3 MHz uncalibrated.
- STANDBYCLK_DEFAULT_FREQ : Low speed standby RC oscillator default frequency.
- RFCLK_BASE_FREQ : Frequency of the 48 MHz crystal used for the RF front-end.
- EXTCLK_MAX_FREQ : Maximum frequency supported by using an external clock.
- SWCLK_MAX_FREQ : Maximum frequency supported by the SWJ-DP interface.
- RCOSC_MAX_FREQ : Maximum frequency supported by the internal RC oscillator.

Functions

- _start : Initialize the application data and start execution with main.
- _sbrk : Increment (or decrement) the top of the heap.
- SystemInit : Initializes the system by clearing and disabling interrupts, updating the SystemCoreClock variable and updating flash timing registers based on the read SYS_CLK.
- SystemCoreClockUpdate : Reads system registers to determine the current system clock frequency, update the SystemCoreClock variable and update the flash timing registers accordingly.

15.2 CMSIS Reference Variable Documentation

15.2.1 RSL15_Sys_Version

```
const short RSL15_Sys_Version
```

Location: rsl15.h:58

RSL15 firmware version (variable)

15.2.2 __Heap_Begin__

```
uint8_t __Heap_Begin__
```

Location: rsl15_start.h:42

Start location for the heap.

15.2.3 __Heap_Limit__

```
uint8_t __Heap_Limit__
```

Location: rsl15_start.h:43

Top limit for the heap.

15.2.4 __stack_limit

```
uint32_t __stack_limit
```

Location: rsl15_start.h:48

Bottom limit for the stack.

15.2.5 __stack

```
uint32_t __stack
```

Location: rsl15_start.h:49

Start location for the stack.

15.2.6 __data_init__

```
uint32_t __data_init__
```

Location: rsl15_start.h:54

Pointer to the data to used to initialize volatile memory.

15.2.7 __data_start__

```
uint32_t __data_start__
```

Location: rsl15_start.h:55

Start address of the initialized data area in volatile memory.

15.2.8 __data_end__

```
uint32_t __data_end__
```

Location: rsl15_start.h:56

End address of the initialized data area in volatile memory.

15.2.9 __bss_start__

```
uint32_t __bss_start__
```

Location: rsl15_start.h:58

Start address of the cleared data area in volatile memory.

15.2.10 __bss_end__

```
uint32_t __bss_end__
```

Location: rsl15_start.h:59

End address of the cleared data area in volatile memory.

15.2.11 __preinit_array_start__

```
void(* __preinit_array_start__[])(void)
```

Location: rsl15_start.h:65

Weakly defined function list pointer for pre-initialization functions.

15.2.12 __preinit_array_end__

```
void(* __preinit_array_end__[])(void)
```

Location: rsl15_start.h:68

Weakly defined pointer to the end of the pre-initialization function list.

15.2.13 __init_array_start__

```
void(* __init_array_start__[])(void)
```

Location: rsl15_start.h:71

Weakly defined function list pointer for initialization functions.

15.2.14 __init_array_end__

```
void(* __init_array_end__[])(void)
```

Location: rsl15_start.h:74

Weakly defined pointer to the end of the initialization function list.

15.2.15 flash_layout

```
 const struct flash_region flash_layout[] =
{
    {
        .start = FLASH0_CODE_BASE,
        .end = FLASH0_CODE_TOP,
        .flash = FLASH,
        .IRQn = FLASH0_COPY_IRQn,
    },
    {
        .start = FLASH0_DATA_BASE,
        .end = FLASH0_DATA_TOP,
        .flash = FLASH,
        .IRQn = FLASH0_COPY_IRQn,
    },
}
```

Location: rsl15_start.h:104

Flash layout for the RSL15 device.

15.2.16 SystemCoreClock

```
 uint32_t SystemCoreClock
```

Location: system_rsl15.h:74

Contains the current SYS_CLK frequency, in Hz.

15.3 CMSIS Reference Data Structures Type Documentation

15.3.1 flash_region

Location: rsl15_start.h:110

Structure used to define flash regions.

Data Fields

Type	*Name*	Description
uint32_t	*start*	First address in the flash region.
uint32_t	*end*	Last address in the flash region.
FLASH_Type *	*flash*	Flash interface for this region.
uint32_t	*IRQn*	Interrupt supporting this flash region.

15.4 CMSIS Reference Macro Definition Documentation

15.4.1 RSL15_SYS_VER_MAJOR

```
#define RSL15_SYS_VER_MAJOR 0x01
```

Location: rsl15.h:44

RSL15 header file major version.

15.4.2 RSL15_SYS_VER_MINOR

```
#define RSL15_SYS_VER_MINOR 0x00
```

Location: rsl15.h:47

RSL15 header file minor version.

15.4.3 RSL15_SYS_VER_REVISION

```
#define RSL15_SYS_VER_REVISION 0x01
```

Location: rsl15.h:50

RSL15 header file revision version.

15.4.4 RSL15_SYS_VER

```
#define RSL15_SYS_VER ((RSL15_SYS_VER_MAJOR << 12) | \
                       (RSL15_SYS_VER_MINOR << 8)  | \
                       (RSL15_SYS_VER_REVISION))
```

Location: rsl15.h:53

RSL15 firmware version.

15.4.5 __ARMv8MML_REV

```
#define __ARMv8MML_REV 0x0000U
```

Location: rsl15.h:100

Arm v8 architecture revision.

15.4.6 __CM33_REV

```
#define __CM33_REV 0x0000U
```

Location: rsl15.h:101

Core revision r0p4.

15.4.7 __FPU_PRESENT

```
#define __FPU_PRESENT 1U
```

Location: rsl15.h:102

FPU present.

15.4.8 __DSP_PRESENT

```
#define __DSP_PRESENT 1U
```

Location: rsl15.h:103

DSP extension present.

15.4.9 __SAUREGION_PRESENT

```
#define __SAUREGION_PRESENT 1U
```

Location: rsl15.h:104

SAU regions present.

15.4.10 __MPU_PRESENT

```
#define __MPU_PRESENT 1U
```

Location: rsl15.h:105

MPU present.

15.4.11 __VTOR_PRESENT

```
#define __VTOR_PRESENT 1U
```

Location: rsl15.h:106

VTOR present.

15.4.12 __NVIC_PRIO_BITS

```
#define __NVIC_PRIO_BITS 3U
```

Location: rsl15.h:107

3 bits used for interrupt priority levels

15.4.13 __Vendor_SysTickConfig

```
#define __Vendor_SysTickConfig 0U
```

Location: rsl15.h:108

Standard SysTick configuration is used.

15.4.14 I2C_REF_VALID

```
#define I2C_REF_VALID (((uint32_t)(ref) == (uint32_t)I2C)  | \
                                        ((uint32_t)(ref) == (uint32_t)I2C0) | \
                                        ((uint32_t)(ref) == (uint32_t)I2C1))
```

Location: rsl15.h:165

Validation of I2C register block pointer reference for assert statements.

15.4.15 LIN_REF_VALID

```
#define LIN_REF_VALID (((uint32_t)(ref) == (uint32_t)LIN) | \
                                        ((uint32_t)(ref) == (uint32_t)LIN0))
```

Location: rsl15.h:170

Validation of LIN register block pointer reference for assert statements.

15.4.16 PCM_REF_VALID

```
#define PCM_REF_VALID (((uint32_t)(ref) == (uint32_t)PCM) | \
                                        ((uint32_t)(ref) == (uint32_t)PCM0))
```

Location: rsl15.h:174

Validation of PCM register block pointer reference for assert statements.

15.4.17 PWM_REF_VALID

```
#define PWM_REF_VALID (((uint32_t)(ref) == (uint32_t)PWM) | \
                                        ((uint32_t)(ref) == (uint32_t)PWM0))
```

Location: rsl15.h:178

Validation of PWM register block pointer reference for assert statements.

15.4.18 SPI_REF_VALID

```
#define SPI_REF_VALID (((uint32_t)(ref) == (uint32_t)SPI)  | \
                                        ((uint32_t)(ref) == (uint32_t)SPI0) | \
                                        ((uint32_t)(ref) == (uint32_t)SPI1))
```

Location: rsl15.h:182

Validation of SPI register block pointer reference for assert statements.

15.4.19 UART_REF_VALID

```
#define UART_REF_VALID (((uint32_t)(ref) == (uint32_t)UART) | \
                                        ((uint32_t)(ref) == (uint32_t)UART0))
```

Location: rsl15.h:187

Validation of UART register block pointer reference for assert statements.

15.4.20 TIMER_REF_VALID

```
#define TIMER_REF_VALID (((uint32_t)(ref) == (uint32_t)TIMER) | \
                                        ((uint32_t)(ref) == (uint32_t)TIMER0) | \
                                        ((uint32_t)(ref) == (uint32_t)TIMER1) | \
                                        ((uint32_t)(ref) == (uint32_t)TIMER2) | \
                                        ((uint32_t)(ref) == (uint32_t)TIMER3))
```

Location: rsl15.h:191

Validation of TIMER register block pointer reference for assert statements.

15.4.21 DMA_REF_VALID

```
#define DMA_REF_VALID (((uint32_t)(ref) == (uint32_t)DMA) | \
                                        ((uint32_t)(ref) == (uint32_t)DMA0) | \
                                        ((uint32_t)(ref) == (uint32_t)DMA1) | \
                                        ((uint32_t)(ref) == (uint32_t)DMA2) | \
                                        ((uint32_t)(ref) == (uint32_t)DMA3))
```

Location: rsl15.h:198

Validation of DMA register block pointer reference for assert statements.

15.4.22 FLASH_REF_VALID

```
#define FLASH_REF_VALID (((uint32_t)(ref) == (uint32_t)FLASH) | \
                                        ((uint32_t)(ref) == (uint32_t)FLASH0))
```

Location: rsl15.h:205

Validation of FLASH register block pointer reference for assert statements.

15.4.23 GPIO_PAD_COUNT

```
#define GPIO_PAD_COUNT 16
```

Location: rsl15.h:218

GPIO peripheral definitions.

Maximum number of GPIO pads

15.4.24 GPIO_GROUP_LOW_PAD_RANGE

```
#define GPIO_GROUP_LOW_PAD_RANGE 16
```

Location: rsl15.h:219

Number of GPIO pads in the lowest group (all)

15.4.25 GPIO_EVENT_CHANNEL_COUNT

```
#define GPIO_EVENT_CHANNEL_COUNT 4
```

Location: rsl15.h:220

Number of available GPIO interrupts.

15.4.26 GPIO0

```
#define GPIO0 0
```

Location: rsl15.h:223

GPIO pads definitions

GPIO 0

15.4.27 GPIO1

```
#define GPIO1 1
```

Location: rsl15.h:224

GPIO 1.

15.4.28 GPIO2

```
#define GPIO2 2
```

Location: rsl15.h:225

GPIO 2.

15.4.29 GPIO3

```
#define GPIO3 3
```

Location: rsl15.h:226

GPIO 3.

15.4.30 GPIO4

```
#define GPIO4 4
```

Location: rsl15.h:227

GPIO 4.

15.4.31 GPIO5

```
#define GPIO5 5
```

Location: rsl15.h:228

GPIO 5.

15.4.32 GPIO6

```
#define GPIO6 6
```

Location: rsl15.h:229

GPIO 6.

15.4.33 GPIO7

```
#define GPIO7 7
```

Location: rsl15.h:230

GPIO 7.

15.4.34 GPIO8

```
#define GPIO8 8
```

Location: rsl15.h:231

GPIO 8.

15.4.35 GPIO9

```
#define GPIO9 9
```

Location: rsl15.h:232

GPIO 9.

15.4.36 GPIO10

```
#define GPIO10 10
```

Location: rsl15.h:233

GPIO 10.

15.4.37 GPIO11

```
#define GPIO11 11
```

Location: rsl15.h:234

GPIO 11.

15.4.38 GPIO12

```
#define GPIO12 12
```

Location: rsl15.h:235

GPIO 12.

15.4.39 GPIO13

```
#define GPIO13 13
```

Location: rsl15.h:236

GPIO 13.

15.4.40 GPIO14

```
#define GPIO14 14
```

Location: rsl15.h:237

GPIO 14.

15.4.41 GPIO15

```
#define GPIO15 15
```

Location: rsl15.h:238

GPIO 15.

15.4.42 SYS_DUMMY_READ

```
#define SYS_DUMMY_READ SYSCTRL->PROD_STATUS
```

Location: rsl15.h:251

Register that always reads back as 0x00000000.

15.4.43 SYS_DUMMY_WRITE

```
#define SYS_DUMMY_WRITE SYSCTRL->CC_DCU_EN0
```

Location: rsl15.h:254

Register to which writes are ineffective.

15.4.44 ERRNO_NO_ERROR

```
#define ERRNO_NO_ERROR 0x0000
```

Location: rsl15.h:310

No error.

15.4.45 ERRNO_GENERAL_FAILURE

```
#define ERRNO_GENERAL_FAILURE 0x0001
```

Location: rsl15.h:313

General error.

15.4.46 DEFAULT_FREQ

```
#define DEFAULT_FREQ 5000000
```

Location: system_rsl15.h:56

High speed main RC oscillator default frequency set by boot ROM application Default value is 3 MHz uncalibrated.

Assuming a worse case of 5 MHz

15.4.47 STANDBYCLK_DEFAULT_FREQ

```
#define STANDBYCLK_DEFAULT_FREQ 32768
```

Location: system_rsl15.h:59

Low speed standby RC oscillator default frequency.

15.4.48 RFCLK_BASE_FREQ

```
#define RFCLK_BASE_FREQ 48000000
```

Location: system_rsl15.h:62

Frequency of the 48 MHz crystal used for the RF front-end.

15.4.49 EXTCLK_MAX_FREQ

```
#define EXTCLK_MAX_FREQ 48000000
```

Location: system_rsl15.h:65

Maximum frequency supported by using an external clock.

15.4.50 SWCLK_MAX_FREQ

```
#define SWCLK_MAX_FREQ 48000000
```

Location: system_rsl15.h:68

Maximum frequency supported by the SWJ-DP interface.

15.4.51 RCOSC_MAX_FREQ

```
#define RCOSC_MAX_FREQ 12000000
```

Location: system_rsl15.h:71

Maximum frequency supported by the internal RC oscillator.

15.5 CMSIS Reference Function Documentation

15.5.1 _start

```
void _start()
```

Location: rsl15_start.h:91

Initialize the application data and start execution with main.

Should be called from the reset vector.
Returns:
None

Assumptions

The symbols **data_init**, **data_start**, **data_end**, **bss_start**, **bss_end**, and **stack_limit** are defined when the application is linked.

Assumptions

The symbol flash_layout exists, and is an array of structures containing the start, end, and FLASH_Type* of all the banks of flash, in ascending memory address order, that the data region could be present in.

Example Code for _start

```
// Initialize static application data
_start();
```

Returns:

None

Assumptions

The symbols **data_init**, **data_start**, **data_end**, **bss_start**, **bss_end**, and **stack_limit** are defined when the application is linked.

Assumptions

The symbol flash_layout exists, and is an array of structures containing the start, end, and FLASH_Type* of all the banks of flash, in ascending memory address order, that the data region could be present in.

Example Code for _start

```
// Initialize static application data
_start();
```

15.5.2 _sbrk

```
int8_t * _sbrk(int increment)
```

Location: rsl15_start.h:104

Increment (or decrement) the top of the heap.

Returns:

The prior value of the heap top (points to the base of the newly allocated data if the heap was incremented); returns -1 if the function was unable to allocate the requested memory

Assumptions

The symbols **Heap_Begin**, **Heap_Limit** are defined when the application is linked.

Example Code for _sbrk

```
// Initialize the heap with no increment
_sbrk(0);
```

Parameters

Direction	Name	Description
in	*increment*	Increment to be applied to the top of the heap

15.5.3 SystemInit

```
void SystemInit()
```

Location: system_rsl15.h:48

Initializes the system by clearing and disabling interrupts, updating the SystemCoreClock variable and updating flash timing registers based on the read SYS_CLK.

Example Code for SystemInit

```
// Setup the system core clock variable
SystemInit();
```

15.5.4 SystemCoreClockUpdate

```
void SystemCoreClockUpdate()
```

Location: system_rsl15.h:82

Reads system registers to determine the current system clock frequency, update the SystemCoreClock variable and update the flash timing registers accordingly.

Example Code for SystemCoreClockUpdate

```
// Update the declared system clock (SYSCLK) frequency
// used by the Arm Cortex-M3 firmware
SystemCoreClockUpdate();
```

CHAPTER 16

Hardware Abstraction Layer Reference

Simple hardware abstraction layer library reference.

16.1 Summary

16.2 Activity Counter

Activity counter hardware abstraction layer.

16.2.1 Summary

Functions

- Sys_ACNT_Start : Start activity counter.
- Sys_ACNT_Stop : Stop activity counter.
- Sys_ACNT_Clear : Clear activity counter values.

16.2.2 Activity Counter Function Documentation

16.2.2.1 Sys_ACNT_Start

```
void Sys_ACNT_Start()
```

Location: acnt.h:42

Start activity counter.

Example Code for Sys_ACNT_Start

```
// Start the activity counter
Sys_ACNT_Start();
```

16.2.2.2 Sys_ACNT_Stop

```
void Sys_ACNT_Stop()
```

Location: acnt.h:52

Stop activity counter.

Example Code for Sys_ACNT_Stop

```
// Stop the activity counter
Sys_ACNT_Stop();
```

16.2.2.3 Sys_ACNT_Clear

```
void Sys_ACNT_Clear()
```

Location: acnt.h:62

Clear activity counter values.

Example Code for Sys_ACNT_Clear

```
// Clear the current counter value in the activity counters.
Sys_ACNT_Clear();
```

16.3 Analog Control System

Analog Control System hardware abstraction layer.

16.3.1 Summary

Functions

- Sys_ACS_WriteRegister : Safely write an ACS register, without corrupting the RTC counts.

16.3.2 Analog Control System Function Documentation

16.3.2.1 Sys_ACS_WriteRegister

```
void Sys_ACS_WriteRegister(volatile uint32_t * p_register, uint32_t value)
```

Location: acs.h:45

Safely write an ACS register, without corrupting the RTC counts.

Example Code for Sys_ACS_WriteRegister

```
// Disable the buck converter
Sys_ACS_WriteRegister(&ACS->VCC_CTRL,
                      ((ACS->VCC_CTRL & ~VCC_BUCK) | VCC_LDO));
```

Parameters

Direction	Name	Description
in	*p_register*	Address of the register to write.
in	*value*	Value to be written to the register.

16.4 Asynchronous Clock Counter

Asynchronous Clock Counter (ASCC) hardware abstraction layer.

16.4.1 Summary

Functions

- Sys_ASCC_GPIOConfig : Configure the sync pulse and asynchronous clock sources.
- Sys_ASCC_Config : Configure the asynchronous clock counter and set initial phase and period count values.
- Sys_ASCC_StartCounters : Reset the counter, and start the phase and period counter mechanisms.

16.4.2 Asynchronous Clock Counter Function Documentation

16.4.2.1 Sys_ASCC_GPIOConfig

```
void Sys_ASCC_GPIOConfig(uint32_t cfg, uint32_t sync, uint32_t async)
```

Location: ascc.h:48

Configure the sync pulse and asynchronous clock sources.

Example Code for Sys_ASCC_GPIOConfig

```
// Configure GPIO pads to connect to the asynchronous clock counter
Sys_ASCC_GPIOConfig(GPIO_MODE_INPUT | GPIO_WEAK_PULL_UP, GPIO0, GPIO1);
```

Parameters

Direction	Name	Description
in	*cfg*	GPIO pin configuration for the ASCC pads
in	*sync*	GPIO pad input for the synchronization pulse
in	*async*	GPIO pad input for the asynchronous clock source

16.4.2.2 Sys_ASCC_Config

```
void Sys_ASCC_Config(ASCC_Type * ascc, uint32_t cfg, uint32_t phasecnt, uint32_t
periodcnt)
```

Location: ascc.h:69

Configure the asynchronous clock counter and set initial phase and period count values.

Example Code for Sys_ASCC_Config

```
// Configure the ASCC to measure 16 periods
// Set the inital count of the phase and period counts to 0
Sys_ASCC_Config(ASCC, ASCC_PERIODS_16, 0, 0);
```

Parameters

Direction	Name	Description
in	*ascc*	Pointer to the asynchronous clock counter instance
in	*cfg*	The number of clock periods the period counter measures; use ASCC_PERIODS_*
in	*phasecnt*	The clock phase counter initial value
in	*periodcnt*	The clock period counter initial value

16.4.2.3 Sys_ASCC_StartCounters

```
void Sys_ASCC_StartCounters(ASCC_Type * ascc)
```

Location: ascc.h:83

Reset the counter, and start the phase and period counter mechanisms.

Example Code for Sys_ASCC_StartCounters

```
// Reset the ASCC counters and start counting again
Sys_ASCC_StartCounters(ASCC);
```

Parameters

Direction	Name	Description
in	*ascc*	Pointer to the asynchronous clock counter instance

16.5 Baseband Interface

Baseband interface hardware abstraction layer.

16.5.1 Summary

Macros

- BLE_NONE : No Bluetooth Low Energy event.
- BLE_RISING_EDGE : Bluetooth Low Energy rising edge event.
- BLE_FALLING_EDGE : Bluetooth Low Energy falling edge event.
- BLE_TRANSITION : Bluetooth Low Energy transition event.

Functions

- Sys_BBIF_CoexIntConfig : Configure the coexistence interrupts to monitor for Bluetooth and other RF activity.

16.5.2 Baseband Interface Macro Definition Documentation

16.5.2.1 BLE_NONE

```
#define BLE_NONE 0x0U
```

Location: bbif.h:38

No Bluetooth Low Energy event.

16.5.2.2 BLE_RISING_EDGE

```
#define BLE_RISING_EDGE 0x1U
```

Location: bbif.h:41

Bluetooth Low Energy rising edge event.

16.5.2.3 BLE_FALLING_EDGE

```
#define BLE_FALLING_EDGE 0x2U
```

Location: bbif.h:44

Bluetooth Low Energy falling edge event.

16.5.2.4 BLE_TRANSITION

```
#define BLE_TRANSITION 0x3U
```

Location: bbif.h:47

Bluetooth Low Energy transition event.

16.5.3 Baseband Interface Function Documentation

16.5.3.1 Sys_BBIF_CoexIntConfig

```
void Sys_BBIF_CoexIntConfig(uint32_t edge, uint32_t types, uint32_t cf_ant_delay, uint32_
t cf_powerup)
```

Location: bbif.h:66

Configure the coexistence interrupts to monitor for Bluetooth and other RF activity.

Example Code for Sys_BBIF_CoexIntConfig

```
// Enable co-exsistence interrupts on BLE rising edge events and while
// receiving on the BLE communication
Sys_BBIF_CoexIntConfig(BLE_RISING_EDGE, BLE_RX_BUSY, 0, 0);
```

Parameters

Direction	Name	Description
in	*edge*	The edge used for all coexistence interrupts; use BLE_[NONE | RISING_EDGE | FALLING_EDGE | TRANSITION]
in	*types*	The types of coexistence interrupts that are relevant; use BLE_RX_BUSY, BLE_TX_BUSY, BLE_IN_PROCESS, and/or EVENT_IN_PROCESS
in	*cf_ant_delay*	Correction factor to be applied to account for antenna delays; used to advance the coexistence interrupts by the specified number of uS.
in	*cf_powerup*	Correction factor to be applied to account for power-up delays on TX, RX; used to advance the coexistence interrupts by the specified number of uS.

16.6 Clock Configuration

Clock configuration hardware abstraction layer.

16.6.1 Summary

Variables

- SystemCoreClock : CMSIS required system core clock variable.

Functions

- Sys_Clocks_XTALClk_Wait : Waits for the XTAL clock to start with a timeout using system clock counter.
- Sys_Clocks_RCSystemClkConfig : Configure the RC oscillator and system clock.
- Sys_Clocks_SystemClkConfig : Configure the system clock.
- Sys_Clocks_XTALClkConfig : Configure the 48 MHz XTAL Oscillator.
- Sys_Clocks_DividerConfig : Configure the clock divisors for a standard configuration:

16.6.2 Clock Configuration Variable Documentation

16.6.2.1 SystemCoreClock

```
uint32_t SystemCoreClock
```

Location: clock.h:46

CMSIS required system core clock variable.

16.6.3 Clock Configuration Function Documentation

16.6.3.1 Sys_Clocks_XTALClk_Wait

```
void Sys_Clocks_XTALClk_Wait()
```

Location: clock.h:49

Waits for the XTAL clock to start with a timeout using system clock counter.

16.6.3.2 Sys_Clocks_RCSystemClkConfig

```
void Sys_Clocks_RCSystemClkConfig(uint32_t cfg, uint32_t rc_cfg)
```

Location: clock.h:114

Configure the RC oscillator and system clock.

Example Code for Sys_Clocks_RCSystemClkConfig

```
// Enable RC oscillators, set 12 MHz multiplier for start RC oscillator
// Set nominal trim settings for RC oscillators
Sys_Clocks_RCSystemClkConfig(SYSCLK_CLKSRC_RCCLK, RC_OSC_12MHZ |
                             RC_OSC_ENABLE  |
                             RC32_OSC_NOM   |
                             RC_OSC_NOM);
```

Parameters

Direction	Name	Description
in	*cfg*	Configuration of the system clock source and prescale value; use SYSCLK_CLKSRC_[RCCLK | STANDBYCLK | RFCLK | JTCK], and JTCK_PRESCALE_*
in	*rc_cfg*	Configuration for the RC oscillator

16.6.3.3 Sys_Clocks_SystemClkConfig

```
void Sys_Clocks_SystemClkConfig(uint32_t cfg)
```

Location: clock.h:138

Configure the system clock.

Assumptions

The flash delay configuration is correct for the previously selected system clock source and frequency; if also changing the RC oscillator frequency, use Sys_Clocks_RCSystemClkConfig()

Example Code for Sys_Clocks_SystemClkConfig

```
// Set SYSCLK source to the RF clock.
Sys_Clocks_SystemClkConfig_Example(SYSCLK_CLKSRC_RFCLK)
```

Parameters

Direction	Name	Description
in	*cfg*	Configuration of the system clock source and prescale value; use SYSCLK_CLKSRC_[RCCLK | STANDBYCLK | RFCLK | JTCK], and JTCK_PRESCALE_*

16.6.3.4 Sys_Clocks_XTALClkConfig

```
void Sys_Clocks_XTALClkConfig(uint32_t xtal_prescaler)
```

Location: clock.h:151

Configure the 48 MHz XTAL Oscillator.

Example Code for Sys_Clocks_XTALClkConfig

```
// Configure the RFCLK to 8 MHz, using the 48 MHz external crystal.
Sys_Clocks_XTALClkConfig(CK_DIV_1_6_PRESCALE_6)
```

Parameters

Direction	Name	Description
in	*xtal_prescaler*	Configuration of the 48MHz XTAL Oscillator as clock and its prescale value; use CK_DIV_1_6_PRESCALE_[NO_CLOCK_BYTE | 1_BYTE | 2_BYTE | 3_BYTE | 4_BYTE | 5_BYTE | 6_BYTE],

16.6.3.5 Sys_Clocks_DividerConfig

```
void Sys_Clocks_DividerConfig(uint32_t uartclk_freq, uint32_t sensorclk_freq, uint32_t
userclk_freq)
```

Location: clock.h:231

Configure the clock divisors for a standard configuration:

- SLOWCLK (1 MHz)
- BBCLK (8 MHz)
- DCCLK (4 MHz)
- CPCLK (166 kHz)
- UARTCLK as per uartclk_freq
- SENSOR_CLK as per sensorclk_freq
- USERCLK as per userclk_freq

If an exact configuration cannot be found for the desired frequency, the clock divisor will be set to ensure the divided clock does not exceed the specified target frequency.

Assumptions

The system clock has previously been configured.

Example Code for Sys_Clocks_DividerConfig

```
// Set UARTCLK to 115200 Hz
// Set SENSORCLK to 1000 Hz
// Set USERCLK to 1000000 Hz
// Set other clocks to typical values:
// SLOWCLK set to 1 MHz
// BBCLK set to 8 MHz
// DCCLK set to 4 MHz
// CPCLK set to 125 kHz
Sys_Clocks_DividerConfig(115200, 1000, 1000000)
```

Parameters

Direction	Name	Description
in	*uartclk_freq*	Target frequency for the UART clock
in	*sensorclk_freq*	Target frequency for the sensor clock
in	*userclk_freq*	Target frequency for user clock; if the target frequency exceeds the system clock frequency and the RF clock is available, USERCLK will be sourced from RF clock.

16.7 Arm Cortex-M Processor

Arm Cortex-M processor support hardware abstraction layer.

16.7.1 Summary

Macros

- SYS_WAIT_FOR_EVENT : Wait for an event.
- SYS_WAIT_FOR_INTERRUPT : Wait for an interrupt.

16.7.2 Arm Cortex-M Processor Macro Definition Documentation

16.7.2.1 SYS_WAIT_FOR_EVENT

```
#define SYS_WAIT_FOR_EVENT __ASM("wfe")
```

Location: cortex.h:44

Wait for an event.

Example Code for SYS_WAIT_FOR_EVENT

```
// Stop executing code, put core into low power state, waiting for an event.
SYS_WAIT_FOR_EVENT;
```

16.7.2.2 SYS_WAIT_FOR_INTERRUPT

```
#define SYS_WAIT_FOR_INTERRUPT __ASM("wfi")
```

Location: cortex.h:50

Wait for an interrupt.

Example Code for SYS_WAIT_FOR_INTERRUPT

```
// Stop executing code, put core into low power state, waiting for an
// interrupt.
SYS_WAIT_FOR_INTERRUPT;
```

16.8 Cyclic Redundancy Check

Cyclic Redundancy Check (CRC) hardware abstraction layer.

16.8.1 Summary

Macros

- SYS_CRC_CONFIG : Macro wrapper for Sys_Set_CRC_Config() Configure the CRC generator type, endianness of the input data, and standard vs non-standard CRC behavior.
- SYS_CRC_32INITVALUE : Macro wrapper for Sys_CRC_32InitValue() Initialize CRC for CRC-32.
- SYS_CRC_CCITTINITVALUE : Macro wrapperr for Sys_CRC_CCITTInitValue() Initialize CRC for CRC-CCITT.
- SYS_CRC_GETCURRENTVALUE : Macro wrapper for Sys_CRC_GetCurrentValue() Initialize CRC for CRC-CCITT.
- SYS_CRC_GETFINALVALUE : Macro wrapper for Sys_CRC_GetFinalValue() Initialize final CRC value.
- SYS_CRC_ADD : Macro wrapper for Sys_CRC_Add() Add data to the current CRC calculation, based on size.

Functions

- Sys_Set_CRC_Config : Configure the CRC generator type, endianness of the input data, and standard vs non-standard CRC behavior.
- Sys_CRC_32InitValue : Initialize CRC for CRC-32.
- Sys_CRC_CCITTInitValue : Initialize CRC for CRC-CCITT.
- Sys_CRC_GetCurrentValue : Retrieve current value from CRC.

- Sys_CRC_GetFinalValue : Initialize final CRC value.
- Sys_CRC_Add : Add data to the current CRC calculation, based on size.

16.8.2 Cyclic Redundancy Check Macro Definition Documentation

16.8.2.1 SYS_CRC_CONFIG

```
#define SYS_CRC_CONFIG Sys_Set_CRC_Config(CRC, (config))
```

Location: crc.h:164

Macro wrapper for Sys_Set_CRC_Config() Configure the CRC generator type, endianness of the input data, and standard vs non-standard CRC behavior.

Assumptions

Note that D_CRC supports only CRC_CCITT mode, hence no configuration is applied for this instance.

Example Code for SYS_CRC_CONFIG

```
// Configure the default CRC block to CRC-32 (IEEE 802.3) algorithm,
// using little endian and non-standard (opposite) bit order
SYS_CRC_CONFIG(CRC_32 | CRC_LITTLE_ENDIAN |
               CRC_BIT_ORDER_NON_STANDARD);
```

Parameters

Direction	*Name*	Description
in	*config*	CRC generator configuration; use CRC_[CCITT | 32], CRC_[BIG | LITTLE]_ENDIAN, CRC_BIT_ORDER_[STANDARD | NON_STANDARD], CRC_FINAL_REVERSE_[STANDARD | NON_STANDARD], and CRC_FINAL_XOR_[STANDARD | NON_STANDARD]

16.8.2.2 SYS_CRC_32INITVALUE

```
#define SYS_CRC_32INITVALUE Sys_CRC_32InitValue(CRC)
```

Location: crc.h:173

Macro wrapper for Sys_CRC_32InitValue() Initialize CRC for CRC-32.

Assumptions

Note that D_CRC supports only CRC_CCITT mode, hence no configuration is applied for this instance.

Example Code for SYS_CRC_32INITVALUE

```
//Initialize the default CRC block for CRC-32
SYS_CRC_32INITVALUE();
```

16.8.2.3 SYS_CRC_CCITTINITVALUE

```
#define SYS_CRC_CCITTINITVALUE Sys_CRC_CCITTInitValue(CRC)
```

Location: crc.h:181

Macro wrapperr for Sys_CRC_CCITTInitValue() Initialize CRC for CRC-CCITT.

Assumptions

CRC is configured to work in CRC-CCITT mode.

Example Code for SYS_CRC_CCITTINITVALUE

```
//Initialize the default CRC block for CRC-CCITT
SYS_CRC_CCITTINITVALUE();
```

16.8.2.4 SYS_CRC_GETCURRENTVALUE

```
#define SYS_CRC_GETCURRENTVALUE Sys_CRC_GetCurrentValue(CRC)
```

Location: crc.h:188

Macro wrapper for Sys_CRC_GetCurrentValue() Initialize CRC for CRC-CCITT.

Example Code for SYS_CRC_GETCURRENTVALUE

```
// Retrieve current value from the default CRC block
SYS_CRC_GETCURRENTVALUE();
```

16.8.2.5 SYS_CRC_GETFINALVALUE

```
#define SYS_CRC_GETFINALVALUE Sys_CRC_GetFinalValue(CRC)
```

Location: crc.h:200

Macro wrapper for Sys_CRC_GetFinalValue() Initialize final CRC value.

Returns:

CRC final value.

Assumptions

D_I2C only supports CRC-CCITT mode. Use Sys_CRC_GetCurrentValue instead. Returns initial value of CRC if D_CRC is passed or any other unknown instance.

Example Code for SYS_CRC_GETFINALVALUE

```
// Retrieve final value from the default CRC block
SYS_CRC_GETFINALVALUE();
```

16.8.2.6 SYS_CRC_ADD

```
#define SYS_CRC_ADD Sys_CRC_Add(CRC, (data), (size))
```

Location: crc.h:209

Macro wrapper for Sys_CRC_Add() Add data to the current CRC calculation, based on size.

Example Code for SYS_CRC_ADD

```
// Add 8 bits of data to the default CRC block
SYS_CRC_ADD(0xF1, 8);
```

Parameters

Direction	*Name*	Description
in	*data*	Data to add
in	*size*	Size of data to add, 1, 8, 16, 24, 32 are valid.

16.8.3 Cyclic Redundancy Check Function Documentation

16.8.3.1 Sys_Set_CRC_Config

```
void Sys_Set_CRC_Config(CRC_Type * crc, uint32_t config)
```

Location: crc.h:52

Configure the CRC generator type, endianness of the input data, and standard vs non-standard CRC behavior.

Assumptions

Note that D_CRC supports only CRC_CCITT mode, hence no configuration is applied for this instance.

Example Code for Sys_Set_CRC_Config

```
// Configure CRC block to CRC-32 (IEEE 802.3) algorithm,
// using little endian and non-standard (opposite) bit order
Sys_Set_CRC_Config(CRC, CRC_32 | CRC_LITTLE_ENDIAN |
                   CRC_BIT_ORDER_NON_STANDARD);
```

Parameters

Direction	Name	Description
in	*crc*	Pointer to the CRC instance
in	*config*	CRC generator configuration; use CRC_[CCITT | 32], CRC_[BIG | LITTLE]_ENDIAN, CRC_BIT_ORDER_[STANDARD | NON_STANDARD], CRC_FINAL_REVERSE_[STANDARD | NON_STANDARD], and CRC_FINAL_XOR_[STANDARD | NON_STANDARD]

16.8.3.2 Sys_CRC_32InitValue

```
void Sys_CRC_32InitValue(CRC_Type * crc)
```

Location: crc.h:68

Initialize CRC for CRC-32.

Assumptions

Note that D_CRC supports only CRC_CCITT mode, hence no configuration is applied for this instance.

Example Code for Sys_CRC_32InitValue

```
// Initialize the CRC block for CRC-32
Sys_CRC_32InitValue(CRC);
```

Parameters

Direction	Name	Description
in	*crc*	Pointer to the CRC instance

16.8.3.3 Sys_CRC_CCITTInitValue

```
void Sys_CRC_CCITTInitValue(CRC_Type * crc)
```

Location: crc.h:83

Initialize CRC for CRC-CCITT.

Assumptions

CRC is configured to work in CRC-CCITT mode.

Example Code for Sys_CRC_CCITTInitValue

```
// Initialize the CRC block for CRC-CCITT
Sys_CRC_CCITTInitValue(CRC);
```

Parameters

Direction	Name	Description
in	*crc*	Pointer to the CRC instance

16.8.3.4 Sys_CRC_GetCurrentValue

```
uint32_t Sys_CRC_GetCurrentValue(const CRC_Type * crc)
```

Location: crc.h:95

Retrieve current value from CRC.

Returns:

Current CRC value.

Example Code for Sys_CRC_GetCurrentValue

```
// Retrieve current value from the CRC block
value = Sys_CRC_GetCurrentValue(CRC);
```

Parameters

Direction	Name	Description
in	*crc*	Pointer to the CRC instance

16.8.3.5 Sys_CRC_GetFinalValue

```
uint32_t Sys_CRC_GetFinalValue(const CRC_Type * crc)
```

Location: crc.h:111

Initialize final CRC value.

Returns:

CRC final value.

Assumptions

D_CRC only supports CRC-CCITT mode. Use Sys_CRC_GetCurrentValue instead. Returns initial value of CRC if D_CRC is passed or any other unknown instance.

Example Code for Sys_CRC_GetFinalValue

```
// Retrieve final value from the CRC block
Sys_CRC_GetFinalValue(CRC);
```

Parameters

Direction	Name	Description
in	*crc*	Pointer to the CRC instance

16.8.3.6 Sys_CRC_Add

```
void Sys_CRC_Add(CRC_Type * crc, uint32_t data, uint32_t size)
```

Location: crc.h:138

Add data to the current CRC calculation, based on size.

Example Code for Sys_CRC_Add

```
// Add 8 bits of data to the current CRC block
Sys_CRC_Add(CRC, 0xF1, 8);
```

Parameters

Direction	Name	Description
in	*crc*	Pointer to the CRC instance
in	*data*	Data to add
in	*size*	Size of data to add, 1, 8, 16, 24, 32 are valid.

16.9 Direct Memory Access

Direct Memory Access (DMA) hardware abstraction layer.

16.9.1 Summary

Macros

- SYS_DMA_CHANNELCONFIG : Macro wrapper for Sys_DMA_ChannelConfig() Configure the DMA channels for a data transfer.
- SYS_DMA_MODE_ENABLE : Macro wrapper for Sys_DMA_Mode_Enable() Configure the DMA channels for a data transfer.

Functions

- Sys_DMA_ChannelConfig : Configure the DMA channels for a data transfer.
- Sys_DMA_Mode_Enable : Configure the DMA channels for a data transfer.
- Sys_DMA_Get_Status : Get the status register of the DMA instance.
- Sys_DMA_Clear_Status : Writes to the CNT_INT_CLEAR, COMPLETE_INT_CLEAR, or SRC_BUFFER_FILL_LVL_WR.
- Sys_DMA_Set_Ctrl : Sets the DMA_CTRL of the DMA instance.

16.9.2 Direct Memory Access Macro Definition Documentation

16.9.2.1 SYS_DMA_CHANNELCONFIG

```
#define SYS_DMA_CHANNELCONFIG Sys_DMA_ChannelConfig(DMA, cfg, \
                                                    transferlength, counterInt, srcAddr, \
                                                    destAddr)
```

Location: dma.h:176

Macro wrapper for Sys_DMA_ChannelConfig() Configure the DMA channels for a data transfer.

Example Code for SYS_DMA_CHANNELCONFIG

```
    // Configure the default DMA channels for data transfer using a transfer length of 4
    // and interrupting at the beginning of the transfer
    SYS_DMA_CHANNELCONFIG(DMA_BIG_ENDIAN | DEST_TRANS_LENGTH_SEL |
                          DMA_PRIORITY_1 | DMA_CNT_INT_ENABLE, 4, 0,
                          (uint32_t)0x2001ffc8, (uint32_t)0x2001ffb8);
```

Parameters

Direction	*Name*	Description
in	*cfg*	Configuration of the DMA transfer behavior; use DMA_[LITTLE | BIG]_ENDIAN, [DEST | SRC]_TRANS_LENGTH_SEL, DMA_PRIORITY_[0 | 1 | 2 | 3], DMA_SRC_* DMA_DEST_* WORD_SIZE_*, DMA_SRC_ADDR_*, DMA_DEST_ADDR_*, DMA_SRC_ADDR_LSB_TOGGLE_[DISABLE | ENABLE], DMA_CNT_INT_[DISABLE | ENABLE], DMA_COMPLETE_INT_[DISABLE | ENABLE]
in	*transferlength*	Configuration of the DMA transfer length
in	*counterInt*	Configuration of when the counter interrupt will occur during the transfer
in	*srcAddr*	Base source address for the DMA transfer
in	*destAddr*	Base destination address for the DMA transfer

16.9.2.2 SYS_DMA_MODE_ENABLE

```
#define SYS_DMA_MODE_ENABLE Sys_DMA_Mode_Enable(DMA, (mode))
```

Location: dma.h:191

Macro wrapper for Sys_DMA_Mode_Enable() Configure the DMA channels for a data transfer.

Example Code for SYS_DMA_MODE_ENABLE

```
// Enable the default DMA block
SYS_DMA_MODE_ENABLE(DMA_ENABLE);
```

Parameters

Direction	*Name*	Description
in	*mode*	Enable mode of operation of the DMA Channel; use DMA_[DISABLE | ENABLE | DMA_ENABLE_WRAP | DMA_ENABLE_WRAP_RESTART | DMA_TRIGGER DMA_TRIGGER_WRAP | DMA_TRIGGER_WRAP_RESTART]

16.9.3 Direct Memory Access Function Documentation

16.9.3.1 Sys_DMA_ChannelConfig

```
void Sys_DMA_ChannelConfig(DMA_Type * dma, uint32_t cfg, uint32_t transferLength, uint32_
t counterInt, uint32_t srcAddr, uint32_t destAddr)
```

Location: dma.h:62

Configure the DMA channels for a data transfer.

Example Code for Sys_DMA_ChannelConfig

```
// Configure the DMA1 channels for data transfer using a transfer length of 4
// and interrupting at the beginning of the transfer
uint32_t srcAddr = (uint32_t)0x2001ffc8;
uint32_t destAddr = (uint32_t)0x2001ffb8;
Sys_DMA_ChannelConfig(DMA1, DMA_BIG_ENDIAN | DEST_TRANS_LENGTH_SEL |
                      DMA_PRIORITY_1 | DMA_CNT_INT_ENABLE, 4, 0,
                      srcAddr, destAddr);
```

Parameters

Direction	Name	Description
in	*dma*	Pointer to the DMA instance
in	*cfg*	Configuration of the DMA transfer behavior; use DMA_[LITTLE | BIG]_ENDIAN, [DEST | SRC]_TRANS_LENGTH_SEL, DMA_PRIORITY_[0 | 1 | 2 | 3], DMA_SRC_* DMA_DEST_* WORD_SIZE_*, DMA_SRC_ADDR_*, DMA_DEST_ADDR_*, DMA_SRC_ADDR_LSB_TOGGLE_[DISABLE | ENABLE], DMA_CNT_INT_[DISABLE | ENABLE], DMA_COMPLETE_INT_[DISABLE | ENABLE]
in	*transferLength*	Configuration of the DMA transfer length
in	*counterInt*	Configuration of when the counter interrupt will occur during the transfer
in	*srcAddr*	Base source address for the DMA transfer
in	*destAddr*	Base destination address for the DMA transfer

16.9.3.2 Sys_DMA_Mode_Enable

```
void Sys_DMA_Mode_Enable(DMA_Type * dma, uint32_t mode)
```

Location: dma.h:107

Configure the DMA channels for a data transfer.

Example Code for Sys_DMA_Mode_Enable

```
// Enable DMA1
Sys_DMA_Mode_Enable(DMA1, DMA_ENABLE);
```

Parameters

Direction	Name	Description
in	*dma*	Pointer to the DMA instance
in	*mode*	Enable mode of operation of the DMA Channel; use DMA_[DISABLE | ENABLE | DMA_ENABLE_WRAP | DMA_ENABLE_WRAP_RESTART | DMA_TRIGGER DMA_TRIGGER_WRAP | DMA_TRIGGER_WRAP_RESTART]

16.9.3.3 Sys_DMA_Get_Status

```
uint32_t Sys_DMA_Get_Status(DMA_Type * dma)
```

Location: dma.h:120

Get the status register of the DMA instance.

Returns:

The DMA_STATUS of the DMA instance.

Example Code for Sys_DMA_Get_Status

```
// Get the current status of DMA channel 1
result = Sys_DMA_Get_Status(DMA1);
```

Parameters

Direction	Name	Description
in	*dma*	Pointer to the DMA instance

16.9.3.4 Sys_DMA_Clear_Status

```
void Sys_DMA_Clear_Status(DMA_Type * dma, uint32_t ctrl)
```

Location: dma.h:133

Writes to the CNT_INT_CLEAR, COMPLETE_INT_CLEAR, or SRC_BUFFER_FILL_LVL_WR.

Example Code for Sys_DMA_Clear_Status

```
// Clear the status register of DMA channel 1
result = Sys_DMA_Clear_Status(DMA1, DMA_COMPLETE_INT_CLEAR | DMA_CNT_INT_CLEAR);
```

Parameters

Direction	Name	Description
in	*dma*	Pointer to the DMA instance
in	*ctrl*	Data to be written to the DMA_SATUS register

16.9.3.5 Sys_DMA_Set_Ctrl

```
void Sys_DMA_Set_Ctrl(DMA_Type * dma, uint32_t ctrl)
```

Location: dma.h:145

Sets the DMA_CTRL of the DMA instance.

Example Code for Sys_DMA_Set_Ctrl

```
// Enables DMA channel 1 which will disable the DMA when the transfer is completed.
result = Sys_DMA_Set_Ctrl(DMA1, DMA_ENABLE);
```

Parameters

Direction	Name	Description
in	*dma*	Pointer to the DMA instance
in	*ctrl*	Data to be written to the DMA_CTRL register

16.10 Flash Copier

Flash copier hardware abstraction layer.

16.10.1 Summary

Functions

- Sys_Flash_Copy : Copy data from the flash memory to a RAM memory instance.
- Sys_Flash_Compare : Compare data in the flash to a pre-specified value.
- Sys_Flash_CalculateCRC : Calculate CRC of words in flash using flash copier.

16.10.2 Flash Copier Function Documentation

16.10.2.1 Sys_Flash_Copy

```
void Sys_Flash_Copy(FLASH_Type * flash, uint32_t src_addr, uint32_t dest_addr, uint32_t
length, uint32_t cpy_dest)
```

Location: flash_copier.h:55

Copy data from the flash memory to a RAM memory instance.

Assumptions

src_addr points to an address in flash memory dest_addr points to an address in RAM memory The flash copy does not need to be complete before returning If CRC is selected as the destination, dest_addr is ignored and 32-bit copy mode is selected automatically.

Example Code for Sys_Flash_Copy

```
// Copy a block of 256 bytes from flash to RAM.
uint32_t srcAddr = (uint32_t)0x001D8000;
uint32_t destAddr = (uint32_t)0x2000E000;
Sys_Flash_Copy(FLASH, srcAddr, destAddr, 0x100, COPY_TO_MEM | COPY_TO_32BIT);
```

Parameters

Direction	Name	Description
in	*flash*	Pointer to the flash instance
in	*src_addr*	Source address in flash to copy data from
in	*dest_addr*	Destination address in RAM to copy data to
in	*length*	Number of words to copy
in	*cpy_dest*	Destination copier is CRC or memories; use COPY_TO_[CRC | MEM], and COPY_TO_[32 | 40]_BIT

16.10.2.2 Sys_Flash_Compare

```
uint32_t Sys_Flash_Compare(FLASH_Type * flash, uint32_t cfg, uint32_t addr, uint32_t length, uint32_t value, uint32_t value_ecc)
```

Location: flash_copier.h:76

Compare data in the flash to a pre-specified value.

Returns:

0 if comparison succeeded, 1 if the comparison failed.

Assumptions

addr points to an address in flash memory

Example Code for Sys_Flash_Compare

```
// Verify that 256 bytes in flash are all 0xFFFFFFFF.
uint32_t srcAddr = (uint32_t)0x001D8000;
Sys_Flash_Compare(FLASH, COMP_MODE_CONSTANT_BYTE |
                         COMP_ADDR_UP_BYTE       |
                         COMP_ADDR_STEP_1_BYTE,
                         srcAddr, 0x100, 0xFFFFFFFF, 0);
```

Parameters

Direction	Name	Description
in	*flash*	Pointer to the flash instance
in	*cfg*	Flash comparator configuration; use COMP_MODE_[CONSTANT | CHBK]_BYTE, COMP_ADDR_[DOWN | UP]_BYTE, and COMP_ADDR_STEP_*_BYTE
in	*addr*	Base address of the area to verify
in	*length*	Number of words to verify
in	*value*	Value that the words read from flash will be compared against
in	*value_ecc*	Value that the error-correction coding bits from the extended words read from flash will be compared against

16.10.2.3 Sys_Flash_CalculateCRC

```
uint32_t Sys_Flash_CalculateCRC(FLASH_Type * flash, uint32_t addr, uint32_t length,
uint32_t * crc)
```

Location: flash_copier.h:91

Calculate CRC of words in flash using flash copier.

Returns:

0 if calculation has succeeded, 1 if the calculation has failed.

NOTE: Flash copier and CRC register are modified

Example Code for Sys_Flash_CalculateCRC

```
// Perform CRC of the 256 bytes in flash at srcAddr.
uint32_t srcAddr = (uint32_t)0x001D8000;
Sys_Flash_CalculateCRC(FLASH, COMP_MODE_CONSTANT_BYTE |
                       COMP_ADDR_UP_BYTE       |
                       COMP_ADDR_STEP_1_BYTE,
                       srcAddr, 0x100, 0xFFFFFFFF, 0);
```

Parameters

Direction	Name	Description
in	*flash*	Flash instance being used
in	*addr*	Address belonging to the flash instance
in	*length*	Total number of words used in CRC calculation
out	*crc*	CRC value calculated using CRC peripheral

16.11 General-Purpose I/O Interface

General-Purpose I/O (GPIO) Interface hardware abstraction layer.

16.11.1 Summary

Macros

- GPIO_LEVEL1_DRIVE : 1st level GPIO drive strength
- GPIO_LEVEL2_DRIVE : 2nd level GPIO drive strength
- GPIO_LEVEL3_DRIVE : 3rd level GPIO drive strength
- GPIO_LEVEL4_DRIVE : 4th level GPIO drive strength
- SYS_GPIO_CONFIG : Configure the specified digital I/O.

Functions

- Sys_GPIO_NMIConfig : Configure a source for NMI input selection.
- Sys_GPIO_IntConfig : Configure a GPIO interrupt source.
- Sys_GPIO_CM33JTAGConfig : Configure Arm Cortex-M3 SWJ-DP.
- Sys_GPIO_Set_High : Set the specified GPIO output value to high.
- Sys_GPIO_Set_Low : Set the specified GPIO output value to low.
- Sys_GPIO_Toggle : Toggle the current value of the specified GPIO output.
- Sys_GPIO_Read : Read the specified GPIO value.
- Sys_GPIO_Write : Write the specified GPIO value.
- Sys_GPIO_Set_SingleDirection : Set the input/output direction for a particular GPIO.
- Sys_GPIO_Set_Direction : Set the input/output direction for any GPIOs configured as GPIOs.

16.11.2 General-Purpose I/O Interface Macro Definition Documentation

16.11.2.1 GPIO_LEVEL1_DRIVE

```
#define GPIO_LEVEL1_DRIVE GPIO_2X_DRIVE
```

Location: gpio.h:45

1st level GPIO drive strength

16.11.2.2 GPIO_LEVEL2_DRIVE

```
#define GPIO_LEVEL2_DRIVE GPIO_3X_DRIVE
```

Location: gpio.h:48

2nd level GPIO drive strength

16.11.2.3 GPIO_LEVEL3_DRIVE

```
#define GPIO_LEVEL3_DRIVE GPIO_5X_DRIVE
```

Location: gpio.h:51

3rd level GPIO drive strength

16.11.2.4 GPIO_LEVEL4_DRIVE

```
#define GPIO_LEVEL4_DRIVE GPIO_6X_DRIVE
```

Location: gpio.h:54

4th level GPIO drive strength

16.11.2.5 SYS_GPIO_CONFIG

```
#define SYS_GPIO_CONFIG SYS_ASSERT(pad < GPIO_PAD_COUNT); \
   GPIO->CFG[(pad)] = (config)
```

Location: gpio.h:69

Configure the specified digital I/O.

Example Code for SYS_GPIO_CONFIG

```
// Enable GPIO 5 as GPIO input using no pull-up resistor
SYS_GPIO_CONFIG(GPIO5, GPIO_MODE_GPIO_IN | GPIO_NO_PULL);
```

Parameters

Direction	*Name*	Description
in	*pad*	Digital I/O pad to configure; use a constant
in	*config*	I/O configuration; use GPIO_*X_DRIVE, GPIO_LPF_[ENABLE | DISABLE], GPIO_*_PULL, and GPIO_MODE_*

16.11.3 General-Purpose I/O Interface Function Documentation

16.11.3.1 Sys_GPIO_NMIConfig

```
void Sys_GPIO_NMIConfig(uint32_t config, uint32_t source, uint32_t polarity)
```

Location: gpio.h:80

Configure a source for NMI input selection.

Example Code for Sys_GPIO_NMIConfig

```
// Configure NMI input using GPIO 13 as active-low source without low-pass filter
Sys_GPIO_NMIConfig(GPIO_LPF_ENABLE, NMI_SRC_GPIO_13, NMI_ACTIVE_LOW);
```

Parameters

Direction	Name	Description
in	*config*	GPIO pin configuration if NMI is pad
in	*source*	NMI source; use NMI_SRC_*
in	*polarity*	NMI polarity; use NMI_ACTIVE_[LOW | HIGH]

16.11.3.2 Sys_GPIO_IntConfig

```
void Sys_GPIO_IntConfig(uint32_t index, uint32_t config, uint32_t dbnc_clk, uint32_t
dbnc_cnt)
```

Location: gpio.h:105

Configure a GPIO interrupt source.

Example Code for Sys_GPIO_IntConfig

```
// configure GPIO interrupt channel 7 to use GPIO 11 as interrupt source, with
// 1 debounce filter count.
Sys_GPIO_IntConfig(7, (GPIO_INT_DEBOUNCE_ENABLE | GPIO_INT_SRC_GPIO_11),
                   GPIO_DEBOUNCE_SLOWCLK_DIV1024, 1);
```

Parameters

Direction	Name	Description
in	*index*	GPIO interrupt source to configure; use a constant
in	*config*	GPIO interrupt configuration; use GPIO__INT_DEBOUNCE_[DISABLE | ENABLE], GPIO_INT_SRC_*, and GPIO_INT_EVENT_[NONE | HIGH_LEVEL | LOW_LEVEL | RISING_EDGE | FALLING_EDGE | TRANSITION]
in	*dbnc_clk*	Interrupt button debounce filter clock; use GPIO_DEBOUNCE_SLOWCLK_DIV[32 | 1024]
in	*dbnc_cnt*	Interrupt button debounce filter count

16.11.3.3 Sys_GPIO_CM33JTAGConfig

```
void Sys_GPIO_CM33JTAGConfig(uint32_t config, uint8_t mode)
```

Location: gpio.h:130

Configure Arm Cortex-M3 SWJ-DP.

Example Code for Sys_GPIO_CM33JTAGConfig

```
// Enable Arm Cortex-M3 SWJ-DP port and JTAG
Sys_GPIO_CM33JTAGConfig(true, true);
```

Parameters

Direction	Name	Description
in	*config*	JTAG pad configuration; use JTMS_[NO_PULL | WEAK_PULL_UP | WEAK_PULL_DOWN | STRONG_PULL_UP], JTMS_[2X | 3X | 5X | 6X]_DRIVE, JTCK_[NO_PULL | WEAK_PULL_UP | WEAK_PULL_DOWN | STRONG_PULL_UP], JTMS_LPF_[DISABLED | ENABLED], and JTCK_LPF_[DISABLED | ENABLED]
in	*mode*	Enable one of the two JTAG modes or SW mode; use 0 for SW mode, 1 for JTAG with reset enabled, and 2 for JTAG with reset disabled

16.11.3.4 Sys_GPIO_Set_High

```
void Sys_GPIO_Set_High(uint32_t pad)
```

Location: gpio.h:160

Set the specified GPIO output value to high.

Example Code for Sys_GPIO_Set_High

```
// Set GPIO2 high
Sys_GPIO_Set_High(GPIO2);
```

Parameters

Direction	Name	Description
in	*pad*	GPIO pad to set high

16.11.3.5 Sys_GPIO_Set_Low

```
void Sys_GPIO_Set_Low(uint32_t pad)
```

Location: gpio.h:171

Set the specified GPIO output value to low.

Example Code for Sys_GPIO_Set_Low

```
// Set GPIO2 low
Sys_GPIO_Set_Low(GPIO2);
```

Parameters

Direction	Name	Description
in	*pad*	GPIO pad to set low

16.11.3.6 Sys_GPIO_Toggle

```
void Sys_GPIO_Toggle(uint32_t pad)
```

Location: gpio.h:182

Toggle the current value of the specified GPIO output.

Example Code for Sys_GPIO_Toggle

```
// Toggle the state of the GPIO2 pin
Sys_GPIO_Toggle(GPIO2);
```

Parameters

Direction	Name	Description
in	*pad*	GPIO pad to toggle

16.11.3.7 Sys_GPIO_Read

```
uint32_t Sys_GPIO_Read(uint32_t pad)
```

Location: gpio.h:194

Read the specified GPIO value.

Returns:

uint32_t pin value; 0 or 1

Example Code for Sys_GPIO_Read

```
// Read the current value of the GPIO2 pin
Sys_GPIO_Read(GPIO2);
```

Parameters

Direction	Name	Description
in	*pad*	GPIO pad to set low

16.11.3.8 Sys_GPIO_Write

```
void Sys_GPIO_Write(uint32_t pad, bool value)
```

Location: gpio.h:206

Write the specified GPIO value.

Example Code for Sys_GPIO_Write

```
// Write a 1 to GPIO2
Sys_GPIO_Write(GPIO2, 1);
```

Parameters

Direction	Name	Description
in	*pad*	GPIO pad to write value
in	*value*	1 or 0 written to GPIO

16.11.3.9 Sys_GPIO_Set_SingleDirection

```
void Sys_GPIO_Set_SingleDirection(uint32_t pad, uint32_t dir)
```

Location: gpio.h:221

Set the input/output direction for a particular GPIO.

Example Code for Sys_GPIO_Set_SingleDirection

```
// Set the direction of the GPIO2 pin
Sys_GPIO_Set_SingleDirection(GPIO2, GPIO2_DIR_OUT);
```

Parameters

Direction	Name	Description
in	*pad*	GPIO pad to write value
in	*dir*	Input/output configuration settings for any GPIOs from 0 to 15 that are configured as GPIO pads; use GPIO*_DIR_IN, and GPIO*_DIR_OUT

16.11.3.10 Sys_GPIO_Set_Direction

```
void Sys_GPIO_Set_Direction(uint32_t dir)
```

Location: gpio.h:244

Set the input/output direction for any GPIOs configured as GPIOs.

Example Code for Sys_GPIO_Set_Direction

```
// Set GPIO2 as an output
Sys_GPIO_Set_Direction(GPIO2_DIR_OUT);
```

Parameters

Direction	Name	Description
in	*dir*	Input/output configuration settings for any GPIOs from 0 to 15 that are configured as GPIO pads; use GPIO*_DIR_IN, and GPIO*_DIR_OUT

16.12 I2C

Inter-Integrated Circuit (I2C) hardware abstraction layer.

16.12.1 Summary

Macros

- I2C_CONFIG_MASK : Mask for the I2C_CFG register.
- I2C_PADS_NUM : Number of pads used for the I2C interface, for a single instance.
- SYS_I2C_GPIOCONFIG : Macro wrapper for Sys_I2C_GPIOConfig() Configure two GPIOs for the specified I2C interface.
- SYS_I2C_CONFIG : Macro wrapper for Sys_I2C_Config() Apply I2C Master mode related configuration.
- SYS_I2C_STARTREAD : Macro wrapper for Sys_I2C_StartRead() Send slave address on the bus with a read request.
- SYS_I2C_STARTWRITE : Macro wrapper for Sys_I2C_StartWrite() Send slave address on the bus with a write request.
- SYS_I2C_ACK : Macro wrapper for Sys_I2C_ACK() Issue an ACK on the I2C interface.
- SYS_I2C_NACK : Macro wrapper for Sys_I2C_NACK() Issue a NACK on the I2C interface.
- SYS_I2C_LASTDATA : Macro wrapper for Sys_I2C_LastData() Indicate that the current data is the last byte.
- SYS_I2C_RESET : Macro wrapper for Sys_I2C_Reset() Reset the I2C interface.
- SYS_I2C_NACKANDSTOP : Macro wrapper for Sys_I2C_NackAndStop() Issue a NACK followed by a Stop condition on I2C bus.

Functions

- Sys_I2C_GPIOConfig : Configure two GPIOs for the specified I2C interface.
- Sys_I2C_Config : Apply I2C Master mode related configuration.
- Sys_I2C_StartRead : Send slave address on the bus with a read request.
- Sys_I2C_StartWrite : Send slave address on the bus with a write request.
- Sys_I2C_ACK : Issue a ACK on the I2C interface.
- Sys_I2C_NACK : Issue a NACK on the I2C interface.
- Sys_I2C_LastData : Indicate that the current data is the last byte.
- Sys_I2C_Reset : Reset the I2C interface.
- Sys_I2C_NackAndStop : Issue a NACK followed by a Stop condition on I2C bus.

16.12.2 I2C Macro Definition Documentation

16.12.2.1 I2C_CONFIG_MASK

```
#define I2C_CONFIG_MASK (((uint32_t)1U << I2C_CFG_CONNECT_IN_STANDBY_Pos) | \
    ((uint32_t)1U << I2C_CFG_TX_DMA_ENABLE_Pos) | \
    ((uint32_t)1U << I2C_CFG_RX_DMA_ENABLE_Pos) | \
    ((uint32_t)1U << I2C_CFG_TX_INT_ENABLE_Pos) | \
    ((uint32_t)1U << I2C_CFG_RX_INT_ENABLE_Pos) | \
    ((uint32_t)1U << I2C_CFG_BUS_ERROR_INT_ENABLE_Pos) | \
    (1U << I2C_CFG_OVERRUN_INT_ENABLE_Pos) | \
    (1U << I2C_CFG_STOP_INT_ENABLE_Pos) | \
    (1U << I2C_CFG_AUTO_ACK_ENABLE_Pos) | \
    I2C_CFG_SLAVE_PRESCALE_Mask | \
    I2C_CFG_MASTER_PRESCALE_Mask | \
    I2C_CFG_SLAVE_ADDRESS_Mask | \
    (1U << I2C_CFG_SLAVE_Pos))
```

Location: i2c.h:41

Mask for the I2C_CFG register.

16.12.2.2 I2C_PADS_NUM

```
#define I2C_PADS_NUM 2U
```

Location: i2c.h:57

Number of pads used for the I2C interface, for a single instance.

16.12.2.3 SYS_I2C_GPIOCONFIG

```
#define SYS_I2C_GPIOCONFIG Sys_I2C_GPIOConfig(I2C, (config), (scl), (sda))
```

Location: i2c.h:206

Macro wrapper for Sys_I2C_GPIOConfig() Configure two GPIOs for the specified I2C interface.

Example Code for SYS_I2C_GPIOCONFIG

```
// Configure GPIO3 and GPIO4 as SCL and SDA, enable 1 kOhm pull-up resistors,
// and disable low-pass filter for default I2C interface
SYS_I2C_GPIOCONFIG((GPIO_LPF_DISABLE | GPIO_1K_PULL_UP),
                   GPIO3, GPIO4);
```

Parameters

Direction	*Name*	Description
in	*config*	GPIO pin configuration for the I2C pads
in	*scl*	GPIO to use as the I2C transmit pad; use an integer
in	*sda*	GPIO to use as the I2C receive pad; use an integer

16.12.2.4 SYS_I2C_CONFIG

```
#define SYS_I2C_CONFIG Sys_I2C_Config(I2C, (config))
```

Location: i2c.h:229

Macro wrapper for Sys_I2C_Config() Apply I2C Master mode related configuration.

Example Code for SYS_I2C_CONFIG

```
// Apply I2C Master mode related configuration for default I2C interface
// Set up prescaler, auto acknowledge, interrupt, and slave enable
SYS_I2C_CONFIG((I2C_SLAVE_PRESCALE_4     |
                I2C_AUTO_ACK_ENABLE      |
                I2C_RX_INT_ENABLE        |
                I2C_TX_INT_ENABLE        |
                I2C_STOP_INT_ENABLE      |
                I2C_OVERRUN_INT_ENABLE   |
                I2C_BUS_ERROR_INT_ENABLE |
                I2C_SLAVE_ENABLE         |
                (64 << I2C_CFG_SLAVE_ADDRESS_Pos)));
```

Parameters

Direction	*Name*	Description
in	*config*	I2C configurations for master mode; use I2C_[CONNECT | DISCONNECT]_IN_STANDBY, I2C_TX_DMA_[ENABLE | DISABLE] I2C_RX_DMA_[ENABLE | DISABLE] I2C_TX_INT_[ENABLE | DISABLE] I2C_RX_INT_[ENABLE | DISABLE] I2C_BUS_ERROR_INT_[ENABLE | DISABLE] I2C_OVERRUN_INT_[ENABLE | DISABLE] I2C_STOP_INT_[ENABLE | DISABLE] I2C_AUTO_ACK_[ENABLE | DISABLE] I2C_MASTER_PRESCALE_*, I2C_SLAVE_PRESCALE_*, a slave address constant shifted to I2C_CTRL0_SLAVE_ADDRESS_Pos, I2C_SLAVE_[ENABLE | DISABLE]

16.12.2.5 SYS_I2C_STARTREAD

```
#define SYS_I2C_STARTREAD Sys_I2C_StartRead(I2C, (addr))
```

Location: i2c.h:237

Macro wrapper for Sys_I2C_StartRead() Send slave address on the bus with a read request.

Example Code for SYS_I2C_STARTREAD

```
// Send slave address 64 on the default I2C bus with a read request
SYS_I2C_STARTREAD(64);
```

Parameters

Direction	*Name*	Description
in	*addr*	I2C address to use for write transaction

16.12.2.6 SYS_I2C_STARTWRITE

```
#define SYS_I2C_STARTWRITE Sys_I2C_StartWrite(I2C, (addr))
```

Location: i2c.h:245

Macro wrapper for Sys_I2C_StartWrite() Send slave address on the bus with a write request.

Example Code for SYS_I2C_STARTWRITE

```
// Send slave address 64 on the default I2C bus with a write request
SYS_I2C_STARTWRITE(64);
```

Parameters

Direction	*Name*	Description
in	*addr*	I2C address to use for write transaction

16.12.2.7 SYS_I2C_ACK

```
#define SYS_I2C_ACK Sys_I2C_ACK(I2C)
```

Location: i2c.h:252

Macro wrapper for Sys_I2C_ACK() Issue an ACK on the I2C interface.

Example Code for SYS_I2C_ACK

```
// Issue a ACK on the default I2C interface
SYS_I2C_ACK();
```

16.12.2.8 SYS_I2C_NACK

```
#define SYS_I2C_NACK Sys_I2C_NACK(I2C)
```

Location: i2c.h:259

Macro wrapper for Sys_I2C_NACK() Issue a NACK on the I2C interface.

Example Code for SYS_I2C_NACK

```
// Issue a NACK on the default I2C interface
SYS_I2C_NACK();
```

16.12.2.9 SYS_I2C_LASTDATA

```
#define SYS_I2C_LASTDATA Sys_I2C_LastData(I2C)
```

Location: i2c.h:266

Macro wrapper for Sys_I2C_LastData() Indicate that the current data is the last byte.

Example Code for SYS_I2C_LASTDATA

```
//Indicate that the current data is the last byte
SYS_I2C_LASTDATA();
```

16.12.2.10 SYS_I2C_RESET

```
#define SYS_I2C_RESET Sys_I2C_Reset(I2C)
```

Location: i2c.h:273

Macro wrapper for Sys_I2C_Reset() Reset the I2C interface.

Example Code for SYS_I2C_RESET

```
// Reset the default I2C interface
SYS_I2C_RESET();
```

16.12.2.11 SYS_I2C_NACKANDSTOP

```
#define SYS_I2C_NACKANDSTOP Sys_I2C_NackAndStop(I2C)
```

Location: i2c.h:280

Macro wrapper for Sys_I2C_NackAndStop() Issue a NACK followed by a Stop condition on I2C bus.

Example Code for SYS_I2C_NACKANDSTOP

```
// Issue a NACK followed by a Stop condition on default I2C bus
SYS_I2C_NACKANDSTOP();
```

16.12.3 I2C Function Documentation

16.12.3.1 Sys_I2C_GPIOConfig

```
void Sys_I2C_GPIOConfig(const I2C_Type * i2c, uint32_t config, uint32_t scl, uint32_t
sda)
```

Location: i2c.h:68

Configure two GPIOs for the specified I2C interface.

Example Code for Sys_I2C_GPIOConfig

```
// Configure GPIO3 and GPIO4 as SCL and SDA, enable 1 kOhm pull-up resistors,
// and disable low-pass filter for I2C interface
Sys_I2C_GPIOConfig(I2C, (GPIO_LPF_DISABLE | GPIO_1K_PULL_UP),
                   GPIO3, GPIO4);
```

Parameters

Direction	Name	Description
in	*i2c*	Pointer to the I2C instance
in	*config*	GPIO pin configuration for the I2C pads
in	*scl*	GPIO to use as the I2C transmit pad; use an integer
in	*sda*	GPIO to use as the I2C receive pad; use an integer

16.12.3.2 Sys_I2C_Config

```
void Sys_I2C_Config(I2C_Type * i2c, uint32_t config)
```

Location: i2c.h:110

Apply I2C Master mode related configuration.

Example Code for Sys_I2C_Config

```
// Apply I2C Master mode related configuration for I2C interface
// Set up prescaler, auto acknowledge, interrupt, and slave enable
Sys_I2C_Config(I2C, (I2C_SLAVE_PRESCALE_4 |
                I2C_AUTO_ACK_ENABLE       |
                I2C_RX_INT_ENABLE         |
                I2C_TX_INT_ENABLE         |
                I2C_STOP_INT_ENABLE       |
                I2C_OVERRUN_INT_ENABLE    |
                I2C_BUS_ERROR_INT_ENABLE  |
                I2C_SLAVE_ENABLE          |
                (64 << I2C_CFG_SLAVE_ADDRESS_Pos)));
```

Parameters

Direction	Name	Description
in	*i2c*	Pointer to the I2C instance
in	*config*	I2C configurations for master mode; use I2C_[CONNECT | DISCONNECT]_IN_STANDBY, I2C_TX_DMA_[ENABLE | DISABLE] I2C_RX_DMA_[ENABLE | DISABLE] I2C_TX_INT_[ENABLE | DISABLE] I2C_RX_INT_[ENABLE | DISABLE] I2C_BUS_ERROR_INT_[ENABLE | DISABLE] I2C_OVERRUN_INT_[ENABLE | DISABLE] I2C_STOP_INT_[ENABLE | DISABLE] I2C_AUTO_ACK_[ENABLE | DISABLE] I2C_MASTER_PRESCALE_*, I2C_SLAVE_PRESCALE_*, a slave address constant shifted to I2C_CTRL0_SLAVE_ADDRESS_Pos, I2C_SLAVE_[ENABLE | DISABLE]

16.12.3.3 Sys_I2C_StartRead

```
void Sys_I2C_StartRead(I2C_Type * i2c, uint32_t addr)
```

Location: i2c.h:122

Send slave address on the bus with a read request.

Example Code for Sys_I2C_StartRead

```
// Send slave address 64 on the I2C bus with a read request
Sys_I2C_StartRead(I2C, 64);
```

Parameters

Direction	Name	Description
in	*i2c*	Pointer to the I2C instance
in	*addr*	I2C address to use for write transaction

16.12.3.4 Sys_I2C_StartWrite

```
void Sys_I2C_StartWrite(I2C_Type * i2c, uint32_t addr)
```

Location: i2c.h:136

Send slave address on the bus with a write request.

Example Code for Sys_I2C_StartWrite

```
// Send slave address 64 on the I2C bus with a write request
Sys_I2C_StartWrite(I2C, 64);
```

Parameters

Direction	Name	Description
in	*i2c*	I2C instance number
in	*addr*	I2C address to use for write transaction

16.12.3.5 Sys_I2C_ACK

```
void Sys_I2C_ACK(I2C_Type * i2c)
```

Location: i2c.h:148

Issue a ACK on the I2C interface.

Example Code for Sys_I2C_ACK

```
// Issue a ACK on the I2C interface
Sys_I2C_ACK(I2C);
```

Parameters

Direction	Name	Description
in	*i2c*	Pointer to the I2C instance

16.12.3.6 Sys_I2C_NACK

```
void Sys_I2C_NACK(I2C_Type * i2c)
```

Location: i2c.h:159

Issue a NACK on the I2C interface.

Example Code for Sys_I2C_NACK

```
// Issue a NACK on the I2C interface
Sys_I2C_NACK(I2C);
```

Parameters

Direction	Name	Description
in	*i2c*	Pointer to the I2C instance

16.12.3.7 Sys_I2C_LastData

```
void Sys_I2C_LastData(I2C_Type * i2c)
```

Location: i2c.h:170

Indicate that the current data is the last byte.

Example Code for Sys_I2C_LastData

```
//Indicate that the current data is the last byte
Sys_I2C_LastData(I2C);
```

Parameters

Direction	Name	Description
in	*i2c*	Pointer to the I2C instance

16.12.3.8 Sys_I2C_Reset

```
void Sys_I2C_Reset(I2C_Type * i2c)
```

Location: i2c.h:181

Reset the I2C interface.

Example Code for Sys_I2C_Reset

```
// Reset the I2C interface
Sys_I2C_Reset(I2C);
```

Parameters

Direction	Name	Description
in	*i2c*	Pointer to the I2C instance

16.12.3.9 Sys_I2C_NackAndStop

```
void Sys_I2C_NackAndStop(I2C_Type * i2c)
```

Location: i2c.h:192

Issue a NACK followed by a Stop condition on I2C bus.

Example Code for Sys_I2C_NackAndStop

```
// Issue a NACK followed by a Stop condition on I2C bus
Sys_I2C_NackAndStop(I2C);
```

Parameters

Direction	Name	Description
in	*i2c*	Pointer to the I2C instance

16.13 LIN

LIN hardware abstraction layer.

16.13.1 Summary

Functions

- Sys_LIN_GPIOConfig : Configure two GPIOs for the specified LIN interface.
- Sys_LIN_Enable : Enable/Wake the connected transceiver, enable LIN.
- Sys_LIN_Disable : Disable the connected transceiver, disable LIN.
- Sys_LIN_ClearErrors : Clear all error flags.

16.13.2 LIN Function Documentation

16.13.2.1 Sys_LIN_GPIOConfig

```
void Sys_LIN_GPIOConfig(const LIN_Type * lin, uint32_t config, uint32_t tx, uint32_t rx)
```

Location: lin.h:50

Configure two GPIOs for the specified LIN interface.

Example Code for Sys_LIN_GPIOConfig

```
// Configure GPIO2 and GPIO3 for the LIN interface
Sys_LIN_GPIOConfig(LIN, GPIO_6X_DRIVE | GPIO_LPF_DISABLE | GPIO_STRONG_PULL_UP,
                   2, 3)
```

Parameters

Direction	Name	Description
in	*lin*	Pointer to the LIN instance
in	*config*	GPIO pin configuration for the LIN pads
in	*tx*	GPIO to use as the LIN transmit pad; use an integer
in	*rx*	GPIO to use as the LIN receive pad; use an integer

16.13.2.2 Sys_LIN_Enable

```
void Sys_LIN_Enable(LIN_Type * lin)
```

Location: lin.h:77

Enable/Wake the connected transceiver, enable LIN.

Assumptions

None

Example Code for Sys_LIN_Enable

```
// Disable the LIN interface
Sys_LIN_Enable(LIN)
```

Parameters

Direction	Name	Description
in	*lin*	Pointer to the LIN instance

16.13.2.3 Sys_LIN_Disable

```
void Sys_LIN_Disable(LIN_Type * lin)
```

Location: lin.h:91

Disable the connected transceiver, disable LIN.

Example Code for Sys_LIN_Disable

```
// Disable the LIN interface
Sys_LIN_Disable(LIN)
```

Parameters

Direction	Name	Description
in	*lin*	Pointer to the LIN instance

16.13.2.4 Sys_LIN_ClearErrors

```
void Sys_LIN_ClearErrors(LIN_Type * lin)
```

Location: lin.h:104

Clear all error flags.

Example Code for Sys_LIN_ClearErrors

```
// Clear any errors generated by the LIN interface
Sys_LIN_ClearErrors(LIN)
```

Parameters

Direction	Name	Description
in	*lin*	Pointer to the LIN instance

16.14 LSAD

LSAD hardware abstraction layer.

16.14.1 Summary

Typedefs

- lsad_io_t
- lsad_in_t
- lsad_channel_t
- lsad_prescale_t
- lsad_mode_t

Data Structures

- F_LSAD_TRIM : LSAD structure for offset/gain conversion function.

Enumerations

- lsad_io
- lsad_pol
- lsad_channel
- lsad_prescale
- lsad_mode

Macros

- LSAD_BIT_RES : Converts a raw ADC value to a voltage, calculated as follows voltage = adc_val * (2 V * 1000 [mV]/1 V / 2^14 steps.)
- RES_15BIT
- RES_16BIT
- V_REF_DIFF_MV
- RESOLUTION_DIV
- ADC_VAL_MV
- LSAD_OFFSET_ERROR_CONV_QUOTIENT
- LSAD_GAIN_ERROR_CONV_QUOTIENT
- ERROR_LSAD_INPUT_CFG
- PRE_SEL_SIZE
- LSAD_CFG_ARR_LENGTH
- LSAD_CFG_ARR_WIDTH
- GPIO_IDX
- IO_CFG_IDX
- MASK_IDX
- POS_SEL_IDX
- NEG_SEL_IDX
- NO_CFG
- LSAD_SPECIAL_CFG_ARR_LENGTH
- LSAD_SPECIAL_CFG_ARR_WIDTH
- LSAD_OCCUPIED_INPUT_ARR_WIDTH
- USER_CFG_IDX
- POS_SEL_SPEC_IDX
- NEG_SEL_SPEC_IDX
- SEL_IDX_SHIFT_DEF
- LSAD_CH_NUM
- LSAD_INPUTS_NUM
- OCCUPIED
- NOT_OCCUPIED
- EVEN_GPIO_NUM
- ODD_GPIO_NUM
- OCC_STATE_IDX
- EVENODD_IDX
- INPUT_MASK_IDX
- NEG_INPUTSEL_START
- POS_INPUTSEL_START

Functions

- Sys_LSAD_Gain_Offset : Convert a gain and offset value from NVR in integer format to float format.
- Sys_LSAD_TempSensor_Gain_Offset : Convert a gain and offset value from NVR in integer format to float format for the temperature sensor.
- Sys_LSAD_TrimsInit : Loads ATE trim values specific to the LSAD.
- Sys_LSAD_ModeConfig : Configure the LSAD to use either normal or continuous mode.
- Sys_LSAD_AlarmConfig : Configure the LSAD monitor alarm.
- Sys_LSAD_InterruptEnable : Sets LSAD interrupt to occur on conversion of the selected channel.
- Sys_LSAD_Start : Start LSAD measurements at the frequency determined by the specified prescale.
- Sys_LSAD_Stop : Stop LSAD measurements.
- Sys_LSAD_InputConfig : Configure LSAD channel inputs.
- Sys_LSAD_Special_InputConfig : Read special LSAD configuration from the LSAD_SpecialCfg array.
- Sys_LSAD_PreSelectWrite : Set LSAD->PRE_SEL_INPUT and return configuration which is written to LSAD->INPUT_SEL[channel].
- Sys_LSAD_GPIO_InputConfig : This function reads GPIO configuration from LSAD_GPIOCfg array.
- Sys_LSAD_GetRawData : Obtain the latest data from the DATA_TRIM_CH register.
- Sys_LSAD_ConvertToMVUntrimmed : Convert data from DATA_TRIM_CH to millivolts using calibrated trim data.
- Sys_LSAD_ConvertToMVTrimmed : Convert data from DATA_TRIM_CH to millivolts without using calibrated trim data.
- Sys_LSAD_NewSampleClear : Clears the ready bit from the MONITOR_STATUS register to acknowledge the most recent LSAD sample and indicate that the next sample will not cause an overrun.

16.14.2 LSAD Typedef Documentation

16.14.2.1 lsad_io_t

```
typedef enum lsad_io lsad_io_t
```

Location: lsad.h:163

16.14.2.2 lsad_in_t

```
typedef enum lsad_pol lsad_in_t
```

Location: lsad.h:169

16.14.2.3 lsad_channel_t

```
typedef enum lsad_channel lsad_channel_t
```

Location: lsad.h:181

16.14.2.4 lsad_prescale_t

```
typedef enum lsad_prescale lsad_prescale_t
```

Location: lsad.h:200

16.14.2.5 lsad_mode_t

```
typedef enum lsad_mode lsad_mode_t
```

Location: lsad.h:206

16.14.3 LSAD Data Structures Type Documentation

16.14.3.1 F_LSAD_TRIM

Location: lsad.h:117

LSAD structure for offset/gain conversion function.

Data Fields

Type	*Name*	Description
float	*lf_offset*	Low frequency LSAD offset.
float	*hf_offset*	High frequency LSAD offset.
float	*lf_gain*	Low frequency LSAD gain.
float	*hf_gain*	High frequency LSAD gain.

16.14.4 LSAD Enumeration Type Documentation

16.14.4.1 lsad_io

Location: lsad.h:133

Members

- `LSAD_GPIO0_CFG = 0U`
- `LSAD_GPIO1_CFG = 1U`
- `LSAD_GPIO2_CFG = 2U`
- `LSAD_GPIO3_CFG = 3U`
- `LSAD_GPIO4_CFG = 4U`
- `LSAD_GPIO5_CFG = 5U`
- `LSAD_GPIO6_CFG = 6U`
- `LSAD_GPIO7_CFG = 7U`
- `LSAD_GPIO8_CFG = 8U`
- `LSAD_GPIO9_CFG = 9U`
- `LSAD_GPIO10_CFG = 10U`
- `LSAD_GPIO11_CFG = 11U`
- `LSAD_GPIO12_CFG = 12U`
- `LSAD_GPIO13_CFG = 13U`
- `LSAD_GPIO14_CFG = 14U`
- `LSAD_GPIO15_CFG = 15U`
- `LSAD_GROUND_CFG = 17U`
- `LSAD_AOUT_CFG = 18U`
- `LSAD_TEMP_CFG = 19U`
- `LSAD_VBAT_CFG = 20U`
- `LSAD_NOT_USED_CFG = LSAD_VBAT_CFG`

16.14.4.2 lsad_pol

Location: lsad.h:165

Members

- NEG_IN
- POS_IN

16.14.4.3 lsad_channel

Location: lsad.h:171

Members

- LSAD_INPUT_CH0 = 0
- LSAD_INPUT_CH1 = 1
- LSAD_INPUT_CH2 = 2
- LSAD_INPUT_CH3 = 3
- LSAD_INPUT_CH4 = 4
- LSAD_INPUT_CH5 = 5
- LSAD_INPUT_CH6 = 6
- LSAD_INPUT_CH7 = 7

16.14.4.4 lsad_prescale

Location: lsad.h:183

Members

- LSAD_PS_200 = LSAD_PRESCALE_200
- LSAD_PS_400 = LSAD_PRESCALE_400
- LSAD_PS_640 = LSAD_PRESCALE_640
- LSAD_PS_800 = LSAD_PRESCALE_800
- LSAD_PS_1600 = LSAD_PRESCALE_1600

- `LSAD_PS_3200 = LSAD_PRESCALE_3200`
- `LSAD_PS_6400 = LSAD_PRESCALE_6400`
- `LSAD_PS_20H = LSAD_PRESCALE_20H`
- `LSAD_PS_40H = LSAD_PRESCALE_40H`
- `LSAD_PS_80H = LSAD_PRESCALE_80H`
- `LSAD_PS_128H = LSAD_PRESCALE_128H`
- `LSAD_PS_160H = LSAD_PRESCALE_160H`
- `LSAD_PS_320H = LSAD_PRESCALE_320H`
- `LSAD_PS_640H = LSAD_PRESCALE_640H`
- `LSAD_PS_1280H = LSAD_PRESCALE_1280H`

16.14.4.5 lsad_mode

Location: lsad.h:202

Members

- `LSAD_MODE_NORMAL = LSAD_NORMAL`
- `LSAD_MODE_CONTINUOUS = LSAD_CONTINUOUS`

16.14.5 LSAD Macro Definition Documentation

16.14.5.1 LSAD_BIT_RES

```
#define LSAD_BIT_RES (14U)
```

Location: lsad.h:54

Converts a raw ADC value to a voltage, calculated as follows voltage = adc_val * (2 V * 1000 [mV]/1 V / 2^14 steps.)

Returns:

The voltage output in mV

Assumptions

Low frequency mode for the ADC is used, meaning that the resolution of the ADC is 14-bits. CONVERT provides voltage level as a milliVolt value based on the input ADC value.

16.14.5.2 RES_15BIT

```
#define RES_15BIT (32768.0f)
```

Location: lsad.h:56

16.14.5.3 RES_16BIT

```
#define RES_16BIT (65536.0f)
```

Location: lsad.h:58

16.14.5.4 V_REF_DIFF_MV

```
#define V_REF_DIFF_MV (1000)
```

Location: lsad.h:60

16.14.5.5 RESOLUTION_DIV

```
#define RESOLUTION_DIV ((x) >> (LSAD_BIT_RES - 1))
```

Location: lsad.h:64

16.14.5.6 ADC_VAL_MV

```
#define ADC_VAL_MV ((int32_t)RESOLUTION_DIV((x) * V_REF_DIFF_MV))
```

Location: lsad.h:68

16.14.5.7 LSAD_OFFSET_ERROR_CONV_QUOTIENT

```
#define LSAD_OFFSET_ERROR_CONV_QUOTIENT (RES_15BIT)
```

Location: lsad.h:72

16.14.5.8 LSAD_GAIN_ERROR_CONV_QUOTIENT

```
#define LSAD_GAIN_ERROR_CONV_QUOTIENT (RES_16BIT)
```

Location: lsad.h:74

16.14.5.9 ERROR_LSAD_INPUT_CFG

```
#define ERROR_LSAD_INPUT_CFG (0x80)
```

Location: lsad.h:77

16.14.5.10 PRE_SEL_SIZE

```
#define PRE_SEL_SIZE (4)
```

Location: lsad.h:79

16.14.5.11 LSAD_CFG_ARR_LENGTH

```
#define LSAD_CFG_ARR_LENGTH (32U)
```

Location: lsad.h:82

16.14.5.12 LSAD_CFG_ARR_WIDTH

```
#define LSAD_CFG_ARR_WIDTH (5U)
```

Location: lsad.h:83

16.14.5.13 GPIO_IDX

```
#define GPIO_IDX (0U)
```

Location: lsad.h:84

16.14.5.14 IO_CFG_IDX

```
#define IO_CFG_IDX (1U)
```

Location: lsad.h:85

16.14.5.15 MASK_IDX

```
#define MASK_IDX (2U)
```

Location: lsad.h:86

16.14.5.16 POS_SEL_IDX

```
#define POS_SEL_IDX (3U)
```

Location: lsad.h:87

16.14.5.17 NEG_SEL_IDX

```
#define NEG_SEL_IDX (4U)
```

Location: lsad.h:88

16.14.5.18 NO_CFG

```
#define NO_CFG (0xFF)
```

Location: lsad.h:89

16.14.5.19 LSAD_SPECIAL_CFG_ARR_LENGTH

```
#define LSAD_SPECIAL_CFG_ARR_LENGTH (4U)
```

Location: lsad.h:91

16.14.5.20 LSAD_SPECIAL_CFG_ARR_WIDTH

```
#define LSAD_SPECIAL_CFG_ARR_WIDTH (3U)
```

Location: lsad.h:92

16.14.5.21 LSAD_OCCUPIED_INPUT_ARR_WIDTH

```
#define LSAD_OCCUPIED_INPUT_ARR_WIDTH (3U)
```

Location: lsad.h:93

16.14.5.22 USER_CFG_IDX

```
#define USER_CFG_IDX (0U)
```

Location: lsad.h:94

16.14.5.23 POS_SEL_SPEC_IDX

```
#define POS_SEL_SPEC_IDX (1U)
```

Location: lsad.h:95

16.14.5.24 NEG_SEL_SPEC_IDX

```
#define NEG_SEL_SPEC_IDX (2U)
```

Location: lsad.h:96

16.14.5.25 SEL_IDX_SHIFT_DEF

```
#define SEL_IDX_SHIFT_DEF (15U)
```

Location: lsad.h:98

16.14.5.26 LSAD_CH_NUM

```
#define LSAD_CH_NUM (8U)
```

Location: lsad.h:100

16.14.5.27 LSAD_INPUTS_NUM

```
#define LSAD_INPUTS_NUM (4U)
```

Location: lsad.h:102

16.14.5.28 OCCUPIED

```
#define OCCUPIED (1U)
```

Location: lsad.h:104

16.14.5.29 NOT_OCCUPIED

```
#define NOT_OCCUPIED (0U)
```

Location: lsad.h:105

16.14.5.30 EVEN_GPIO_NUM

```
#define EVEN_GPIO_NUM (0U)
```

Location: lsad.h:107

16.14.5.31 ODD_GPIO_NUM

```
#define ODD_GPIO_NUM (1U)
```

Location: lsad.h:108

16.14.5.32 OCC_STATE_IDX

```
#define OCC_STATE_IDX (2U)
```

Location: lsad.h:109

16.14.5.33 EVENODD_IDX

```
#define EVENODD_IDX (1U)
```

Location: lsad.h:110

16.14.5.34 INPUT_MASK_IDX

```
#define INPUT_MASK_IDX 0U
```

Location: lsad.h:111

16.14.5.35 NEG_INPUTSEL_START

```
#define NEG_INPUTSEL_START (1U)
```

Location: lsad.h:113

16.14.5.36 POS_INPUTSEL_START

```
#define POS_INPUTSEL_START (0U)
```

Location: lsad.h:114

16.14.6 LSAD Function Documentation

16.14.6.1 Sys_LSAD_Gain_Offset

```
void Sys_LSAD_Gain_Offset(const volatile struct LSAD_TRIM * i_gain_offset, struct F_LSAD_TRIM * f_gain_offset)
```

Location: lsad.h:220

Convert a gain and offset value from NVR in integer format to float format.

Example Code for Sys_LSAD_Gain_Offset

```
// Convert gain and offset values in integer format to floating point values
// used in compensation for LSAD measurements. Low frequency and high frequency
// values are converted.
F_LSAD_TRIM f_gain_offset[1];
Sys_LSAD_Gain_Offset(&(trim_region->temp_sensor), &(f_gain_offset[0]));
```

Parameters

Direction	Name	Description
in	*i_gain_offset*	Gain and offset error from NVR, raw integer form
out	*f_gain_offset*	Gain and offset error converted to floating point.

16.14.6.2 Sys_LSAD_TempSensor_Gain_Offset

```
void Sys_LSAD_TempSensor_Gain_Offset(const volatile struct TEMP_SENSOR_TRIM * i_gain_
offset, struct F_LSAD_TRIM * f_gain_offset)
```

Location: lsad.h:242

Convert a gain and offset value from NVR in integer format to float format for the temperature sensor.

Example Code for Sys_LSAD_TempSensor_Gain_Offset

```
// Convert gain and offset values in integer format to floating point values
// used in compensation of measurements for the temperature sensor. Only low
// frequency values are converted.
F_LSAD_TRIM f_gain_offset[1];
Sys_LSAD_TempSensor_Gain_Offset(&(trim_region->temp_sensor), &(f_gain_offset[0]);
```

Parameters

Direction	Name	Description
in	*i_gain_offset*	Gain and offset error from NVR, raw integer form
out	*f_gain_offset*	Gain and offset error converted to floating point.

16.14.6.3 Sys_LSAD_TrimsInit

```
void Sys_LSAD_TrimsInit()
```

Location: lsad.h:257

Loads ATE trim values specific to the LSAD.

Example Code for Sys_LSAD_TrimsInit

```
// Initialize the LSAD with calibrated trim values.
Sys_LSAD_TrimsInit();
```

16.14.6.4 Sys_LSAD_ModeConfig

```
void Sys_LSAD_ModeConfig(lsad_mode_t mode)
```

Location: lsad.h:264

Configure the LSAD to use either normal or continuous mode.

Example Code for Sys_LSAD_ModeConfig

```
// Configure the LSAD to normal mode, where the LSAD converts channels.
Sys_LSAD_ModeConfig(LSAD_MODE_NORMAL);
```

Parameters

Direction	Name	Description
in	*mode*	Either normal (sample all channels) or continuous (sample one channel)

16.14.6.5 Sys_LSAD_AlarmConfig

```
void Sys_LSAD_AlarmConfig(lsad_channel_t channel, uint8_t threshold, uint8_t count)
```

Location: lsad.h:274

Configure the LSAD monitor alarm.

Example Code for Sys_LSAD_AlarmConfig

```
     // Configure the LSAD to trigger the monitor alarm when the voltage on channel 0
     // drops below the value encoded in ALARM_THRESHOLD for ALARM_
COUNT number of samples.
     Sys_LSAD_AlarmConfig(LSAD_INPUT_CH0, ALARM_THRESHOLD, ALARM_COUNT);
```

Parameters

Direction	Name	Description
in	*channel*	The LSAD channel on which to enable the alarm
in	*threshold*	The value below which the alarm should be triggered. The values 0-255 are linearly mapped to 0-2 V with a step size of 7.8 mV.
in	*count*	The number of readings below the threshold needed to trigger the alarm

16.14.6.6 Sys_LSAD_InterruptEnable

```
void Sys_LSAD_InterruptEnable(lsad_channel_t IRQ_channel)
```

Location: lsad.h:282

Sets LSAD interrupt to occur on conversion of the selected channel.

Example Code for Sys_LSAD_InterruptEnable

```
     // Enable LSAD interrupt, set to trigger after sampling channel LSAD_IRQ_
CHANNEL by default.
     // It can be configured to a different channel in app.h of the LSAD sample.
     Sys_LSAD_InterruptEnable(LSAD_IRQ_CHANNEL);
```

Parameters

Direction	Name	Description
in	*IRQ_channel*	The channel after which to interrupt. In continuous mode, this is the only channel sampled.

16.14.6.7 Sys_LSAD_Start

```
void Sys_LSAD_Start(lsad_prescale_t prescale)
```

Location: lsad.h:289

Start LSAD measurements at the frequency determined by the specified prescale.

Example Code for Sys_LSAD_Start

```
// Start LSAD measurements with the frequency determined by prescale value.
Sys_LSAD_Start(LSAD_PRESCALE);
```

Parameters

Direction	Name	Description
in	*prescale*	The prescale setting used to derive the LSAD sampling frequency

16.14.6.8 Sys_LSAD_Stop

```
void Sys_LSAD_Stop()
```

Location: lsad.h:295

Stop LSAD measurements.

Example Code for Sys_LSAD_Stop

```
// Stop LSAD measurements
Sys_LSAD_Stop();
```

16.14.6.9 Sys_LSAD_InputConfig

```
uint32_t Sys_LSAD_InputConfig(lsad_channel_t channel, lsad_io_t pos_io, lsad_io_t neg_io)
```

Location: lsad.h:306

Configure LSAD channel inputs.

Returns:

Resulting input selection register

Example Code for Sys_LSAD_InputConfig

```
// Use LSAD channel 0.
// Set LSAD positive source to GPIO13.
// Set LSAD negative source to Ground.
Sys_LSAD_InputConfig(LSAD_INPUT_CH0, LSAD_GPIO13_CFG, LSAD_GROUND_CFG);
```

Parameters

Direction	Name	Description
in	*channel*	The LSAD channel to configure
in	*pos_io*	The signal to connect to the positive input of the channel
in	*neg_io*	The signal to connect to the negative input of the channel

16.14.6.10 Sys_LSAD_Special_InputConfig

```
uint8_t Sys_LSAD_Special_InputConfig(lsad_io_t gpio)
```

Location: lsad.h:314

Read special LSAD configuration from the LSAD_SpecialCfg array.

Returns:

The corresponding LSAD configuration

Example Code for Sys_LSAD_Special_InputConfig

```
// Get the positive input configurations of the LSAD.
lsad_io_t pos_io = LSAD_GPIO13_CFG;
uint8_t pos_lsad_cfg;
pos_lsad_cfg = Sys_LSAD_Special_InputConfig(pos_io);
```

Parameters

Direction	Name	Description
in	*gpio*	The signal to compare with special LSAD input values

16.14.6.11 Sys_LSAD_PreSelectWrite

```
uint32_t Sys_LSAD_PreSelectWrite(uint8_t cfg, lsad_io_t gpio, lsad_in_t input_type)
```

Location: lsad.h:326

Set LSAD->PRE_SEL_INPUT and return configuration which is written to LSAD->INPUT_SEL[channel].

Returns:

The configuration data for the input selection register

Example Code for Sys_LSAD_PreSelectWrite

```
// Set the LSAD pre-selection register to positive input and return the
// configuration that should be written to LSAD->INPUT_SEL[channel].
uint32_t pos_sel;
pos_sel = Sys_LSAD_PreSelectWrite(pos_lsad_cfg, pos_io, POS_IN);
```

Parameters

Direction	Name	Description
in	*cfg*	The LSAD configuration returned from Sys_LSAD_InputConfig()
in	*gpio*	The signal to be configured as an LSAD input
in	*input_type*	Specifies if the signal should be connected to the positive or negative input

16.14.6.12 Sys_LSAD_GPIO_InputConfig

```
uint8_t Sys_LSAD_GPIO_InputConfig(lsad_io_t gpio)
```

Location: lsad.h:335

This function reads GPIO configuration from LSAD_GPIOCfg array.

Returns:

lsad_conf

Example Code for Sys_LSAD_GPIO_InputConfig

```
// Get the GPIO configuration of the specified GPIO (13).
lsad_io_t gpio = LSAD_GPIO13_CFG;
cfg = Sys_LSAD_GPIO_InputConfig(gpio);
```

Parameters

Direction	Name	Description
in	*gpio*	The signal to compare with gpio numbers

16.14.6.13 Sys_LSAD_GetRawData

```
uint32_t Sys_LSAD_GetRawData(lsad_channel_t channel)
```

Location: lsad.h:344

Obtain the latest data from the DATA_TRIM_CH register.

Returns:

The obtained data

Example Code for Sys_LSAD_GetRawData

```
// Sum the incoming LSAD data to be averaged later.
static uint32_t raw_LSAD_sum[NUM_LSAD_CHANNELS_USED] = {0};
for(uint8_t chn = 0; chn < NUM_LSAD_CHANNELS_USED; chn++)
{
    raw_LSAD_sum[chn] += Sys_LSAD_GetRawData((lsad_channel_t)chn);
}
```

Parameters

Direction	Name	Description
in	*channel*	The LSAD channel from which to obtain data

16.14.6.14 Sys_LSAD_ConvertToMVUntrimmed

```
uint32_t Sys_LSAD_ConvertToMVUntrimmed(uint32_t raw_data)
```

Location: lsad.h:353

Convert data from DATA_TRIM_CH to millivolts using calibrated trim data.

Returns:

The converted data

Example Code for Sys_LSAD_ConvertToMVUntrimmed

```
// Process the averaged data without calibrated trim values.
uint32_t lsad_mv_raw = Sys_LSAD_ConvertToMVUntrimmed(raw_LSAD_avg);
```

Parameters

Direction	Name	Description
in	*raw_data*	The value obtained from DATA_TRIM_CH

16.14.6.15 Sys_LSAD_ConvertToMVTrimmed

```
int32_t Sys_LSAD_ConvertToMVTrimmed(uint32_t raw_data)
```

Location: lsad.h:362

Convert data from DATA_TRIM_CH to millivolts without using calibrated trim data.

Returns:

The converted data

Example Code for Sys_LSAD_ConvertToMVTrimmed

```
// Process the averaged data with calbirated trim values.
int32_t lsad_mv_calib = Sys_LSAD_ConvertToMVTrimmed(raw_LSAD_avg);
```

Parameters

Direction	Name	Description
in	*raw_data*	The value obtained from DATA_TRIM_CH

16.14.6.16 Sys_LSAD_NewSampleClear

```
void Sys_LSAD_NewSampleClear()
```

Location: lsad.h:370

Clears the ready bit from the MONITOR_STATUS register to acknowledge the most recent LSAD sample and indicate that the next sample will not cause an overrun.

Example Code for Sys_LSAD_NewSampleClear

```
// Clear LSAD ready bit, indicating LSAD data has been processed so
// the next conversion completion will not cause an overrun.
Sys_LSAD_NewSampleClear();
```

16.15 Nested Vectored Interrupt Controller

Nested Vectored Interrupt Controller (NVIC) hardware abstraction layer.

16.15.1 Summary

Functions

- Sys_NVIC_DisableAllInt : Disable all external interrupts.
- Sys_NVIC_ClearAllPendingInt : Clear the pending status for all external interrupts.

16.15.2 Nested Vectored Interrupt Controller Function Documentation

16.15.2.1 Sys_NVIC_DisableAllInt

```
__STATIC_INLINE void Sys_NVIC_DisableAllInt()
```

Location: nvic.h:42

Disable all external interrupts.

Example Code for Sys_NVIC_DisableAllInt

```
// Disable all external interrupts
Sys_NVIC_DisableAllInt();
```

16.15.2.2 Sys_NVIC_ClearAllPendingInt

```
__STATIC_INLINE void Sys_NVIC_ClearAllPendingInt()
```

Location: nvic.h:53

Clear the pending status for all external interrupts.

Example Code for Sys_NVIC_ClearAllPendingInt

```
// Clear all pending external interrupts
Sys_NVIC_ClearAllPendingInt();
```

16.16 Power Supply

Power supply hardware abstraction layer.

16.16.1 Summary

Functions

- Sys_Power_RFEnable : RF Power Switch and Isolation.
- Sys_Power_CC312_Enable : CryptoCell312 Power Enable.
- Sys_Power_CC312_Disable : CryptoCell312 Power Disable.
- Sys_Power_CC312AO_Enable : CryptoCell312AO Power Switch and Isolation.

- Sys_Power_CC312AO_Disable : CryptoCell312AO Power Switch and Isolation.
- Sys_Power_FPU_Enable : Enable the FPU.
- Sys_Power_FPU_Disable : Disable the FPU.

16.16.2 Power Supply Function Documentation

16.16.2.1 Sys_Power_RFEnable

```
void Sys_Power_RFEnable()
```

Location: power.h:45

RF Power Switch and Isolation.

Example Code for Sys_Power_RFEnable

```
// Enable RF power switches, remove RF Isolation
Sys_Power_RFEnable();
```

16.16.2.2 Sys_Power_CC312_Enable

```
void Sys_Power_CC312_Enable()
```

Location: power.h:67

CryptoCell312 Power Enable.

Example Code for Sys_Power_CC312_Enable

```
// Enable the CryptoCell block and remove its power isolation.
Sys_Power_CC312_Enable();
```

16.16.2.3 Sys_Power_CC312_Disable

```
void Sys_Power_CC312_Disable()
```

Location: power.h:92

CryptoCell312 Power Disable.

Example Code for Sys_Power_CC312_Disable

```
// Disable and isolate power for the CryptoCell block.
Sys_Power_CC312_Disable();
```

16.16.2.4 Sys_Power_CC312AO_Enable

```
void Sys_Power_CC312AO_Enable()
```

Location: power.h:112

CryptoCell312AO Power Switch and Isolation.

Example Code for Sys_Power_CC312AO_Enable

```
// Enable the CryptoCell block and remove its power isolation, and power
// on the Always On (AO) component for the CryptoCell block.
Sys_Power_CC312AO_Enable();
```

16.16.2.5 Sys_Power_CC312AO_Disable

```
void Sys_Power_CC312AO_Disable()
```

Location: power.h:140

CryptoCell312AO Power Switch and Isolation.

Example Code for Sys_Power_CC312AO_Disable

```
// Disable and isolate power for the CryptoCell block, and power off the
// Always On (AO) component for the CryptoCell block.
Sys_Power_CC312AO_Disable();
```

16.16.2.6 Sys_Power_FPU_Enable

```
void Sys_Power_FPU_Enable()
```

Location: power.h:166

Enable the FPU.

Example Code for Sys_Power_FPU_Enable

```
// Enables the FPU and removes its isolation.
Sys_Power_FPU_Enable();
```

16.16.2.7 Sys_Power_FPU_Disable

```
uint32_t Sys_Power_FPU_Disable()
```

Location: power.h:201

Disable the FPU.

Returns:

FPU_Q_ACCEPTED if the FPU power down was successful. FPU_Q_DENIED if the FPU power down failed or if the FPU is already off.

Example Code for Sys_Power_FPU_Disable

```
// Disables and isolates the FPU.
result = Sys_Power_FPU_Disable();
```

16.17 HAL Power Modes

16.17.1 Summary

Typedefs

- p_application_return : Function pointer for application return address in BOOT_CUSTOM boot config.
- p_appplication_gpio_config : Function pointer to configure GPIO after wakeup from sleep.
- sleep_mode_ret_regulator_cfg : sleep mode retention regulator configuration
- standby_ret_trim_cfg : standby regulator trim value
- clock_cfg : clock configuration for Power RUN mode
- sleep_mode_cfg : sleep mode configuration
- standby_mode_cfg : standby mode configuration
- deepsleep_mode_cfg : deep sleep mode configuration

Data Structures

- sleep_mode_ret_regulator_cfg_t : sleep mode retention regulator configuration
- standby_ret_trim_cfg_t : standby regulator trim value
- clock_cfg_t : clock configuration for Power RUN mode
- sleep_mode_cfg_t : sleep mode configuration
- standby_mode_cfg_t : standby mode configuration
- deepsleep_mode_cfg_t : deep sleep mode configuration

Enumerations

- Sleep_Retention_t : enum for sleep mode with retention type

Macros

- BLE_PRESENT : Macro for enabling BLE stack RF registers copy during sleep/wakeup, used in sleep_mode_cfg, or standby_mode_cfg, or deepsleep_mode_cfg.
- BLE_NOT_PRESENT : Macro for disabling BLE stack RF registers copy during sleep/wakeup, used in sleep_mode_cfg, or standby_mode_cfg, or deepsleep_mode_cfg.
- RF_REGISTERS_IMAGE_SIZE_BYTES : Size of RF registers in bytes.
- VDDMRETENTION_TRIM_MINIMUM : VDDM retention minimum trim value.
- VDDMRETENTION_TRIM : VDDM retention trim value.
- VDDMRETENTION_TRIM_MAXIMUM : VDDM retention maximum trim value.

- VDDTRETENTION_DISABLE : VDDT baseband retention regulator disable.
- VDDTRETENTION_ENABLE : VDDT baseband retention regulator enable.
- VDDCRETENTION_TRIM_MINIMUM : VDDC retention minimum trim value.
- VDDCRETENTION_TRIM : VDDC retention trim value.
- VDDCRETENTION_TRIM_MAXIMUM : VDDC retention maximum trim value.
- VDDACSRETENTION_TRIM_MINIMUM : VDDACS retention maximum trim value.
- VDDACSRETENTION_TRIM : VDDACS retention trim value.
- VDDACSRETENTION_TRIM_MAXIMUM : VDDACS retention maximum trim value.
- RSL15_CHIP_ID_SV2 : RSL15 SV2 Chip ID value.
- CHIP_ID_IS_SV2 : Is the current chip ID SV2?
- CP_ENABLED_IN_SLEEP : Is the charge pump enabled in sleep mode?
- STANDBYCLK_IS_ENABLED : Is the STANDBYCLK enabled?

Functions

- Sys_PowerModes_Sleep_Init : Initialize the Sleep Configuration.
- Sys_PowerModes_Sleep_Enter : Enter Sleep Mode with the retention_type as specified by the input parameter.
- Sys_PowerModes_Standby_Init : Initialize the Standby Configuration.
- Sys_PowerModes_Standby_Enter : Enter Standby mode.
- Sys_PowerModes_DeepSleep_Init : Initialize the DeepSleep Configuration.
- Sys_PowerModes_DeepSleep_Enter : Enter DeepSleep mode.
- Sys_PowerModes_Wakeup_WithReset : Wakeup from Flash with reset for sleep mode with SLEEP_NO_RETENTION and deep sleep power mode.
- Sys_PowerModes_Wakeup_ConfigEnable : ACS Wakeup configuration enable.
- Sys_PowerModes_Wakeup_ConfigDisable : ACS Wakeup configuration disable.

16.17.2 HAL Power Modes Typedef Documentation

16.17.2.1 p_application_return

```
typedef void(* p_application_return
```

Location: power_modes.h:112

Function pointer for application return address in BOOT_CUSTOM boot config.

16.17.2.2 p_appplication_gpio_config

```
typedef void(* p_appplication_gpio_config
```

Location: power_modes.h:115

Function pointer to configure GPIO after wakeup from sleep.

16.17.2.3 sleep_mode_ret_regulator_cfg

```
typedef struct sleep_mode_ret_regulator_cfg_t sleep_mode_ret_regulator_cfg
```

Location: power_modes.h:132

sleep mode retention regulator configuration

16.17.2.4 standby_ret_trim_cfg

```
typedef struct standby_ret_trim_cfg_t standby_ret_trim_cfg
```

Location: power_modes.h:143

standby regulator trim value

16.17.2.5 clock_cfg

```
typedef struct clock_cfg_t clock_cfg
```

Location: power_modes.h:154

clock configuration for Power RUN mode

16.17.2.6 sleep_mode_cfg

```
typedef struct sleep_mode_cfg_t sleep_mode_cfg
```

Location: power_modes.h:197

sleep mode configuration

16.17.2.7 standby_mode_cfg

```
typedef struct standby_mode_cfg_t standby_mode_cfg
```

Location: power_modes.h:233

standby mode configuration

16.17.2.8 deepsleep_mode_cfg

```
typedef struct deepsleep_mode_cfg_t deepsleep_mode_cfg
```

Location: power_modes.h:255

deep sleep mode configuration

16.17.3 HAL Power Modes Data Structures Type Documentation

16.17.3.1 sleep_mode_ret_regulator_cfg_t

Location: power_modes.h:121

sleep mode retention regulator configuration

Data Fields

Type	*Name*	Description
uint8_t	*vddm_ret_trim*	vddm retention trimming value; use 0x00 to 0x03
uint8_t	*vddt_ret*	vddt baseband timer regulator retention; use VDDTRETENTION_ENABLE, VDDTRETENTION_DISABLE
uint8_t	*vddacs_ret_trim*	vddacs retention trimming value; use 0x00 to 0x03
uint8_t	*vddc_ret_trim*	vddc retention trimming value; use 0x00 to 0x03

16.17.3.2 standby_ret_trim_cfg_t

Location: power_modes.h:137

standby regulator trim value

Data Fields

Type	*Name*	Description
uint8_t	*vddc_standby_trim*	vddc standby retention trimming value; use 0x00 to 0x3F
uint8_t	*vddm_standby_trim*	vddm standby retention trimming value; use 0x00 to 0x3F

16.17.3.3 clock_cfg_t

Location: power_modes.h:148

clock configuration for Power RUN mode

Data Fields

Type	*Name*	Description
uint32_t	*systemclk_freq*	System clock frequency value.
uint32_t	*uartclk_freq*	UART clock frequency derived from system clock.
uint32_t	*sensorclk_freq*	Sensor clock frequency value.
uint32_t	*userclk_freq*	user clock frequency

16.17.3.4 sleep_mode_cfg_t

Location: power_modes.h:160

sleep mode configuration

Data Fields

Type	*Name*	Description
uint32_t	*wakeup_cfg*	ACS Wake up configuration; use WAKEUP_DELAY_[1 | 2 | 4 | ... | 128], WAKEUP_DCDC_OVERLOAD_[ENABLE | DISABLE], WAKEUP_GPIO*_[RISING | FALLING], WAKEUP_GPIO*_[ENABLE | DISABLE], WAKEUP_FIFO_[ENABLE | DISABLE], WAKEUP_RTC_OVERFLOW_[ENABLE | DISABLE]
uint32_t	*boot_cfg*	Boot Configuration; use BOOT_FLASH_XTAL_DISABLE, BOOT_CUSTOM, BOOT_FLASH_XTAL_DEFAULT_TRIM, BOOT_FLASH_XTAL_CUSTOM_TRIM.
clock_cfg	*clock_cfg*	clock configuration
p_appplication_gpio_config	*app_gpio_config*	pointer to function for application GPIO Configuration after wakeup
uint8_t	*DMA_channel_RF*	DMA channel used to save/restore RF registers in each sleep/wake-up cycle.
uint8_t	*ble_present*	ble stack present flag
sleep_mode_ret_regulator_cfg	*vddret_ctrl*	VDD Retention regulator configuration.
p_application_return	*app_addr*	pointer to function for application resume address for BOOT_CUSTOM wakeup configuration

16.17.3.5 standby_mode_cfg_t

Location: power_modes.h:202

standby mode configuration

Data Fields

Type	*Name*	Description
uint32_t	*wakeup_cfg*	ACS Wake up configuration; use WAKEUP_DELAY_[1 | 2 | 4 | ... | 128], WAKEUP_DCDC_OVERLOAD_[ENABLE | DISABLE], WAKEUP_GPIO*_[RISING | FALLING], WAKEUP_GPIO*_[ENABLE | DISABLE], WAKEUP_FIFO_[ENABLE | DISABLE], WAKEUP_RTC_OVERFLOW_[ENABLE | DISABLE]
uint32_t	*boot_cfg*	Boot Configuration; use BOOT_FLASH_XTAL_DISABLE, BOOT_CUSTOM, BOOT_FLASH_XTAL_DEFAULT_TRIM, BOOT_FLASH_XTAL_CUSTOM_TRIM.
clock_cfg	*clock_cfg*	clock configuration
p_appplication_gpio_config	*app_gpio_config*	pointer to function for application GPIO Configuration after wakeup
uint8_t	*DMA_channel_RF*	DMA channel used to save/restore RF registers in each sleep/wake-up cycle.
uint8_t	*ble_present*	ble stack present flag
standby_ret_trim_cfg	*vdd_standby_trim*	standby Retention regulator trim

16.17.3.6 deepsleep_mode_cfg_t

Location: power_modes.h:238

deep sleep mode configuration

Data Fields

Type	*Name*	Description
uint32_t	*wakeup_cfg*	ACS Wake up configuration; use WAKEUP_DELAY_[1 | 2 | 4 | ... | 128], WAKEUP_GPIO*_[RISING | FALLING], WAKEUP_GPIO*_[ENABLE | DISABLE]
uint32_t	*boot_cfg*	Boot Configuration; use BOOT_FLASH_XTAL_DISABLE, BOOT_FLASH_XTAL_DEFAULT_TRIM, BOOT_FLASH_XTAL_CUSTOM_TRIM.

clock_cfg	*clock_cfg*	clock configuration
p_appplication_gpio_config	*app_gpio_config*	pointer to function for application GPIO Configuration after wakeup

16.17.4 HAL Power Modes Enumeration Type Documentation

16.17.4.1 Sleep_Retention_t

Location: power_modes.h:104

enum for sleep mode with retention type

Members

- `SLEEP_NO_RETENTION`

  Sleep with no retention.

- `SLEEP_MEM_RETENTION`

  Sleep with memory retention only.

- `SLEEP_CORE_RETENTION`

  Sleep with Core retention.

16.17.5 HAL Power Modes Macro Definition Documentation

16.17.5.1 BLE_PRESENT

```
#define BLE_PRESENT 1
```

Location: power_modes.h:42

Macro for enabling BLE stack RF registers copy during sleep/wakeup, used in sleep_mode_cfg, or standby_mode_cfg, or deepsleep_mode_cfg.

16.17.5.2 BLE_NOT_PRESENT

```
#define BLE_NOT_PRESENT 0
```

Location: power_modes.h:47

Macro for disabling BLE stack RF registers copy during sleep/wakeup, used in sleep_mode_cfg, or standby_mode_cfg, or deepsleep_mode_cfg.

16.17.5.3 RF_REGISTERS_IMAGE_SIZE_BYTES

```
#define RF_REGISTERS_IMAGE_SIZE_BYTES 0xf8
```

Location: power_modes.h:50

Size of RF registers in bytes.

16.17.5.4 VDDMRETENTION_TRIM_MINIMUM

```
#define VDDMRETENTION_TRIM_MINIMUM 0x0U
```

Location: power_modes.h:53

VDDM retention minimum trim value.

16.17.5.5 VDDMRETENTION_TRIM

```
#define VDDMRETENTION_TRIM 0x1U
```

Location: power_modes.h:56

VDDM retention trim value.

16.17.5.6 VDDMRETENTION_TRIM_MAXIMUM

```
#define VDDMRETENTION_TRIM_MAXIMUM 0x3U
```

Location: power_modes.h:59

VDDM retention maximum trim value.

16.17.5.7 VDDTRETENTION_DISABLE

```
#define VDDTRETENTION_DISABLE 0x0U
```

Location: power_modes.h:62

VDDT baseband retention regulator disable.

16.17.5.8 VDDTRETENTION_ENABLE

```
#define VDDTRETENTION_ENABLE 0x1U
```

Location: power_modes.h:65

VDDT baseband retention regulator enable.

16.17.5.9 VDDCRETENTION_TRIM_MINIMUM

```
#define VDDCRETENTION_TRIM_MINIMUM 0x0U
```

Location: power_modes.h:68

VDDC retention minimum trim value.

16.17.5.10 VDDCRETENTION_TRIM

```
#define VDDCRETENTION_TRIM 0x1U
```

Location: power_modes.h:71

VDDC retention trim value.

16.17.5.11 VDDCRETENTION_TRIM_MAXIMUM

```
#define VDDCRETENTION_TRIM_MAXIMUM 0x3U
```

Location: power_modes.h:74

VDDC retention maximum trim value.

16.17.5.12 VDDACSRETENTION_TRIM_MINIMUM

```
#define VDDACSRETENTION_TRIM_MINIMUM 0x0U
```

Location: power_modes.h:77

VDDACS retention maximum trim value.

16.17.5.13 VDDACSRETENTION_TRIM

```
#define VDDACSRETENTION_TRIM 0x1U
```

Location: power_modes.h:80

VDDACS retention trim value.

16.17.5.14 VDDACSRETENTION_TRIM_MAXIMUM

```
#define VDDACSRETENTION_TRIM_MAXIMUM 0x3U
```

Location: power_modes.h:83

VDDACS retention maximum trim value.

16.17.5.15 RSL15_CHIP_ID_SV2

```
#define RSL15_CHIP_ID_SV2 0x0B020200U
```

Location: power_modes.h:86

RSL15 SV2 Chip ID value.

16.17.5.16 CHIP_ID_IS_SV2

```
#define CHIP_ID_IS_SV2 (AHBREGS->CHIP_ID_NUM == RSL15_CHIP_ID_SV2)
```

Location: power_modes.h:91

Is the current chip ID SV2?

16.17.5.17 CP_ENABLED_IN_SLEEP

```
#define CP_ENABLED_IN_SLEEP (ACS->SLEEP_MODE_CFG & (1 << ACS_SLEEP_MODE_CFG_VDDCP_ENABLE_
Pos))
```

Location: power_modes.h:94

Is the charge pump enabled in sleep mode?

16.17.5.18 STANDBYCLK_IS_ENABLED

```
#define STANDBYCLK_IS_ENABLED ((ACS->XTAL32K_CTRL & (1 << ACS_XTAL32K_CTRL_READY_Pos)) ||
\
                                                (ACS->RCOSC_CTRL & (1 << ACS_RCOSC_CTRL_RC32_
OSC_EN_Pos)))
```

Location: power_modes.h:97

Is the STANDBYCLK enabled?

16.17.6 HAL Power Modes Function Documentation

16.17.6.1 Sys_PowerModes_Sleep_Init

```
void Sys_PowerModes_Sleep_Init(sleep_mode_cfg * p_sleep_mode_cfg)
```

Location: power_modes.h:264

Initialize the Sleep Configuration.

NOTE: This function will modify ACS->WAKEUP_CFG and ACS->BOOT_CFG

Example Code for Sys_PowerModes_Sleep_Init

```
// Initialize sleep before entering sleep
Sys_PowerModes_Sleep_Init(&app_sleep_mode_cfg);
```

Parameters

Direction	Name	Description
in	*p_sleep_mode_cfg*	pointer to the sleep_mode_cfg instance

16.17.6.2 Sys_PowerModes_Sleep_Enter

```
void Sys_PowerModes_Sleep_Enter(sleep_mode_cfg * p_sleep_mode_cfg, Sleep_Retention_t
retention_type)
```

Location: power_modes.h:278

Enter Sleep Mode with the retention_type as specified by the input parameter.

NOTE: This function will enter sleep mode if there are no wakeup events or NVIC interrupts pending, otherwise it will continue to RUN mode and service pending events and interrupts

Example Code for Sys_PowerModes_Sleep_Enter

```
// Power Mode enter sleep with core retention
Sys_PowerModes_Sleep_Enter(&app_sleep_mode_cfg, SLEEP_CORE_RETENTION);
```

Parameters

Direction	Name	Description
in	*p_sleep_mode_cfg*	pointer to the sleep_mode_cfg instance
in	*retention_type*	retention type for sleep mode; use SLEEP_NO_RETENTION, SLEEP_MEM_RETENTION, SLEEP_CORE_RETENTION

16.17.6.3 Sys_PowerModes_Standby_Init

```
void Sys_PowerModes_Standby_Init(standby_mode_cfg * p_standby_mode_cfg)
```

Location: power_modes.h:287

Initialize the Standby Configuration.

NOTE: This function will modify ACS->WAKEUP_CFG and ACS->BOOT_CFG

Example Code for Sys_PowerModes_Standby_Init

```
// Initialize standby mode
Sys_PowerModes_Standby_Init(&app_standby_mode_cfg);
```

Parameters

Direction	Name	Description
in	*p_standby_mode_cfg*	pointer to the standby_mode_cfg instance

16.17.6.4 Sys_PowerModes_Standby_Enter

```
void Sys_PowerModes_Standby_Enter(standby_mode_cfg * p_standby_mode_cfg)
```

Location: power_modes.h:298

Enter Standby mode.

NOTE: This function will enter standby mode if there are no wakeup events or NVIC interrupts pending, otherwise it will continue to RUN mode and service pending events and interrupts

Example Code for Sys_PowerModes_Standby_Enter

```
// Power Mode enter standby mode
Sys_PowerModes_Standby_Enter(&app_standby_mode_cfg);
```

Parameters

Direction	Name	Description
in	*p_standby_mode_cfg*	pointer to the sleep_mode_cfg instance

16.17.6.5 Sys_PowerModes_DeepSleep_Init

```
void Sys_PowerModes_DeepSleep_Init(deepsleep_mode_cfg * p_deepsleep_mode_cfg)
```

Location: power_modes.h:307

Initialize the DeepSleep Configuration.

NOTE: This function will modify ACS->WAKEUP_CFG and ACS->BOOT_CFG

Example Code for Sys_PowerModes_DeepSleep_Init

```
// Initialize deepsleep
Sys_PowerModes_DeepSleep_Init(&app_deepsleep_mode_cfg);
```

Parameters

Direction	Name	Description
in	*p_deepsleep_mode_cfg*	pointer to the deepsleep_mode_cfg instance

16.17.6.6 Sys_PowerModes_DeepSleep_Enter

```
void Sys_PowerModes_DeepSleep_Enter(deepsleep_mode_cfg * p_deepsleep_mode_cfg)
```

Location: power_modes.h:318

Enter DeepSleep mode.

NOTE: This function will enter standby mode if there are no wakeup events or NVIC interrupts pending, otherwise it will continue to RUN mode and service pending events and interrupts

Example Code for Sys_PowerModes_DeepSleep_Enter

```
// Power Mode enter deepsleep
Sys_PowerModes_DeepSleep_Enter(&app_deepsleep_mode_cfg);
```

Parameters

Direction	Name	Description
in	*p_deepsleep_mode_cfg*	pointer to the deepsleep_mode_cfg instance

16.17.6.7 Sys_PowerModes_Wakeup_WithReset

```
void Sys_PowerModes_Wakeup_WithReset(sleep_mode_cfg * p_sleep_mode_cfg)
```

Location: power_modes.h:328

Wakeup from Flash with reset for sleep mode with SLEEP_NO_RETENTION and deep sleep power mode.

NOTE: This function initialize the system after wakeup with reset

Example Code for Sys_PowerModes_Wakeup_WithReset

```
        // Initialize the system after a wakeup where the system was reset
(no retention).
        // Use the deep sleep configuration
        Sys_PowerModes_Wakeup_WithReset((sleep_mode_cfg*)p_deepsleep_mode_cfg);
```

Parameters

Direction	Name	Description
in	*p_sleep_mode_cfg*	pointer to the sleep_mode_cfg instance

16.17.6.8 Sys_PowerModes_Wakeup_ConfigEnable

```
void Sys_PowerModes_Wakeup_ConfigEnable(uint32_t acs_wakeup_cfg, uint32_t * p_wakeup_cfg)
```

Location: power_modes.h:345

ACS Wakeup configuration enable.

NOTE: This function will modify the corresponding enables bits only

Example Code for Sys_PowerModes_Wakeup_ConfigEnable

```
        // Power Mode wakeup configuration enable
        Sys_PowerModes_Wakeup_ConfigEnable((WAKEUP_FIFO_ENABLE | WAKEUP_GPIO1_
ENABLE)&app_sleep_mode_cfg->wakeup_cfg);
```

Parameters

Direction	Name	Description
in	*acs_wakeup_cfg*	Wakeup Config configuration; use WAKEUP_DCDC_OVERLOAD_ENABLE, WAKEUP_GPIO*_FALLING, WAKEUP_GPIO*_ENABLE, WAKEUP_RTC_OVERFLOW_ENABLE, WAKEUP_FIFO_ENABLE,
in	*p_wakeup_cfg*	pointer to the structure member instance of sleep_mode_cfg, or deepsleep_mode_cfg, or standby_mode_cfg

16.17.6.9 Sys_PowerModes_Wakeup_ConfigDisable

```
void Sys_PowerModes_Wakeup_ConfigDisable(uint32_t acs_wakeup_cfg, uint32_t * p_wakeup_
cfg)
```

Location: power_modes.h:362

ACS Wakeup configuration disable.

NOTE: This function will modify the corresponding enables bits only

Example Code for Sys_PowerModes_Wakeup_ConfigDisable

```
        // Power Mode wakeup configuration disable
        Sys_PowerModes_Wakeup_ConfigDisable(WAKEUP_FIFO_ENABLE | WAKEUP_GPIO1_
ENABLE)&app_sleep_mode_cfg->wakeup_cfg);
```

Parameters

Direction	Name	Description
in	*acs_wakeup_cfg*	Wakeup Config configuration; use WAKEUP_DCDC_OVERLOAD_ENABLE, WAKEUP_GPIO*_FALLING, WAKEUP_GPIO*_ENABLE, WAKEUP_RTC_OVERFLOW_ENABLE, WAKEUP_FIFO_ENABLE,
in	*p_wakeup_cfg*	pointer to the structure member instance of sleep_mode_cfg, or deepsleep_mode_cfg, or standby_mode_cfg

16.18 Pulse-Width Modulation

Pulse-Width Modulation (PWM) hardware abstraction layer.

16.18.1 Summary

Macros

- PWM_CHANNELS : The total number of PWM channels.
- HIGH_FRACTIONAL : High fractional.

Functions

- Sys_PWM_GPIOConfig : Configure a GPIO pad for the PWM interface.
- Sys_PWM_Config : Set length of PWM period and number of high cycles within a PWM period.
- Sys_PWM_Enable : Enable a set of PWM channels.
- Sys_PWM_Disable : Disable the PWM[4:0] channel Disabling the PWM channel will set the counter and prescaler to zero.
- Sys_PWM_Reset_Channel : Reset the PWM[4:0] channel Disable the PWM[4:0] channel, reset the PWM_CFG [4:0] and PWM_OFFSET[4:1] registers and the related counter.

16.18.2 Pulse-Width Modulation Macro Definition Documentation

16.18.2.1 PWM_CHANNELS

```
#define PWM_CHANNELS 5
```

Location: pwm.h:42

The total number of PWM channels.

16.18.2.2 HIGH_FRACTIONAL

```
#define HIGH_FRACTIONAL 0x100
```

Location: pwm.h:45

High fractional.

16.18.3 Pulse-Width Modulation Function Documentation

16.18.3.1 Sys_PWM_GPIOConfig

```
void Sys_PWM_GPIOConfig(uint8_t gpio, uint8_t channel, uint32_t cfg, uint8_t inverted)
```

Location: pwm.h:60

Configure a GPIO pad for the PWM interface.

Example Code for Sys_PWM_GPIOConfig

```
// Enable PWM channel 0 on GPIO5. Use a non-inverted waveform
Sys_PWM_GPIOConfig(PWM0_GPIO5, PWM_CH0,
                   GPIO_6X_DRIVE | GPIO_LPF_DISABLE | GPIO_STRONG_PULL_UP,
                   PWM_NON_INVERTED);
```

Parameters

Direction	Name	Description
in	*gpio*	Pad to configure as the output for the specified PWM channel
in	*channel*	PWM channel
in	*cfg*	Configuration of the GPIO pad
in	*inverted*	Output PWM interface signal inverted

16.18.3.2 Sys_PWM_Config

```
void Sys_PWM_Config(uint8_t channel, uint32_t period, float duty_cycle, uint32_t offset)
```

Location: pwm.h:85

Set length of PWM period and number of high cycles within a PWM period.

Example Code for Sys_PWM_Config

```
// Configure PWM channel 0 to a period of 2, with 50% duty cycle an no offset
Sys_PWM_Config(PWM_CH0, 2, 50, 0);
```

Parameters

Direction	Name	Description
in	*channel*	PWM channel
in	*period*	Length of a PWM period
in	*duty_cycle*	Duty cycle value (expressed in percentage)
in	*offset*	Cycles between the rising edge of the specified PWM channel and the rising edge of PWM channel 0

16.18.3.3 Sys_PWM_Enable

```
void Sys_PWM_Enable(uint32_t enable)
```

Location: pwm.h:113

Enable a set of PWM channels.

Example Code for Sys_PWM_Enable

```
// Enable output of PWM channel 0
Sys_PWM_Enable(PWM0_ENABLE);
```

Parameters

Direction	Name	Description
in	*enable*	PWM channel

16.18.3.4 Sys_PWM_Disable

```
void Sys_PWM_Disable(uint32_t disable)
```

Location: pwm.h:125

Disable the PWM[4:0] channel Disabling the PWM channel will set the counter and prescaler to zero.

Example Code for Sys_PWM_Disable

```
// Disable output of PWM channel 0, clear the counter and prescaler
Sys_PWM_Disable(PWM0_ENABLE);
```

Parameters

Direction	Name	Description
in	*disable*	PWM channel

16.18.3.5 Sys_PWM_Reset_Channel

```
void Sys_PWM_Reset_Channel(uint32_t reset)
```

Location: pwm.h:137

Reset the PWM[4:0] channel Disable the PWM[4:0] channel, reset the PWM_CFG[4:0] and PWM_OFFSET[4:1] registers and the related counter.

Example Code for Sys_PWM_Reset_Channel

```
// Reset PWM channel 0
Sys_PWM_Reset_Channel(PWM0_RESET);
```

Parameters

Direction	Name	Description
in	*reset*	PWM channel

16.19 RFFE Radio Frequency Front End

Radio Frequency Front End (RFFE) hardware abstraction layer.

16.19.1 Summary

Macros

- STABILIZATION_DELAY : External include files.
- MEASUREMENT_DELAY : Corresponds to sample rate of the LSAD as configured (625 Hz)
- V_TO_MV : Factor for converting back and forth from mV to V.
- V_TO_MV_F : Float iteration of factor for converting back and forth from mV to V.
- DEF_CHANNEL : Default LSAD channel used to measure voltage rails.
- MAX_LSAD_CHANNEL : Maximum number of LSAD channels in the design.
- VDDPA_EN : VDDPA enable selection.
- VDDPA_DIS : VDDPA disable selection.
- VCC_VDDRF_MARGIN : We strongly recommended having VCC at least 50mV higher than VDDRF.
- TRIM_MARGIN : Trim margin.
- MV_PER_DBM_VDDPA : Estimated voltage increase per 1dBm increased TX power.
- MV_PER_DBM_VDDRF : Estimated voltage increase per 1dBm increased TX power.
- STEPS_PER_DBM_VDDRF : Estimated trim steps per 1dBm increased TX power.
- STEPS_PER_DBM_VDDPA : Estimated trim steps per 1dBm increased TX power.
- RF_MAX_POWER : Maximum RF output power possible.
- RF_MAX_POWER_NO_VDDPA : Maximum RF output power possible without using VDDPA.
- RF_NO_VDDPA_TYPICAL_POWER : Typical RF output power when VDDPA is not used, with default trims.
- RF_DEFAULT_POWER : RF output power used by default.
- RF_MIN_POWER : Minimum possible RF output power.
- PA_PWR_BYTE_0DBM : RF Output Power code for 0dBm.
- PA_ENABLE_BIAS_SETTING : Power amplifier bias enable.
- PA_DISABLE_BIAS_SETTING : Power amplifier bias disable.
- SW_CTRL_DELAY_3_BYTE : Switch control delay.
- RAMPUP_DELAY_3_BYTE : Ramp-up delay.
- DISABLE_DELAY_3_BYTE : Disable delay.
- ERRNO_TX_POWER_MARKER : Error marker for RFFE errors.
- ERRNO_NO_TRIMS : No trims found when attempting to adjust voltage rails.
- ERRNO_RFFE_MISSINGSETTING_ERROR : Setting does not exist.
- ERRNO_RFFE_INVALIDSETTING_ERROR : Setting is not possible.
- ERRNO_RFFE_VCC_INSUFFICIENT : VCC is too low to increase VDDRF sufficiently to suppor the requested RF output power.
- WARNING_RFFE_VLOW_POWER_STATE : Warning that the device is in a very low RF output power state.
- WARNING_RFFE_PA_ENABLED_STATE : Warning that the device has the power amplifier enabled.
- CONVERT : Converts an ADC code to a voltage, calculated as follows voltage = adc_code * (2 V * 1000 [mV]/1 V / 2^14 steps).
- SWAP : Swap the values in variables a and b.
- SYS_RFFE_SETTXPOWER : Set the TX Power according to the desired target value with an accuracy of +/-1 dBm for +6 dBm to -17 dBm.

Functions

- Sys_RFFE_GetTXPower : Retrieve the current setting for RF output power by using the values retrieved from the appropriate registers.
- Sys_RFFE_SetTXPower : Set the TX Power according to the desired target value with an accuracy of +/-1 dBm for +6 dBm to -17 dBm.

16.19.2 RFFE Radio Frequency Front End Macro Definition Documentation

16.19.2.1 STABILIZATION_DELAY

```
#define STABILIZATION_DELAY (SystemCoreClock * 3 / 625)
```

Location: rffe.h:46

External include files.

Three times the length of time corresponding to the minimum sample rate, which is deemed sufficient to allow the LSAD to stabilize

16.19.2.2 MEASUREMENT_DELAY

```
#define MEASUREMENT_DELAY (SystemCoreClock / 625)
```

Location: rffe.h:49

Corresponds to sample rate of the LSAD as configured (625 Hz)

16.19.2.3 V_TO_MV

```
#define V_TO_MV 1000
```

Location: rffe.h:52

Factor for converting back and forth from mV to V.

16.19.2.4 V_TO_MV_F

```
#define V_TO_MV_F 1000.0f
```

Location: rffe.h:55

Float iteration of factor for converting back and forth from mV to V.

16.19.2.5 DEF_CHANNEL

```
#define DEF_CHANNEL 6
```

Location: rffe.h:58

Default LSAD channel used to measure voltage rails.

16.19.2.6 MAX_LSAD_CHANNEL

```
#define MAX_LSAD_CHANNEL 7
```

Location: rffe.h:61

Maximum number of LSAD channels in the design.

16.19.2.7 VDDPA_EN

```
#define VDDPA_EN 1
```

Location: rffe.h:64

VDDPA enable selection.

16.19.2.8 VDDPA_DIS

```
#define VDDPA_DIS 0
```

Location: rffe.h:67

VDDPA disable selection.

16.19.2.9 VCC_VDDRF_MARGIN

```
#define VCC_VDDRF_MARGIN 50
```

Location: rffe.h:70

We strongly recommended having VCC at least 50mV higher than VDDRF.

16.19.2.10 TRIM_MARGIN

```
#define TRIM_MARGIN 10    /*mv*/
```

Location: rffe.h:74

Trim margin.

The granularity of trims for VDDRF allow for a certain error above or below the set value

16.19.2.11 MV_PER_DBM_VDDPA

```
#define MV_PER_DBM_VDDPA 100
```

Location: rffe.h:77

Estimated voltage increase per 1dBm increased TX power.

16.19.2.12 MV_PER_DBM_VDDRF

```
#define MV_PER_DBM_VDDRF 75
```

Location: rffe.h:80

Estimated voltage increase per 1dBm increased TX power.

16.19.2.13 STEPS_PER_DBM_VDDRF

```
#define STEPS_PER_DBM_VDDRF 6
```

Location: rffe.h:83

Estimated trim steps per 1dBm increased TX power.

16.19.2.14 STEPS_PER_DBM_VDDPA

```
#define STEPS_PER_DBM_VDDPA 10
```

Location: rffe.h:86

Estimated trim steps per 1dBm increased TX power.

16.19.2.15 RF_MAX_POWER

```
#define RF_MAX_POWER 6
```

Location: rffe.h:89

Maximum RF output power possible.

16.19.2.16 RF_MAX_POWER_NO_VDDPA

```
#define RF_MAX_POWER_NO_VDDPA 2
```

Location: rffe.h:92

Maximum RF output power possible without using VDDPA.

16.19.2.17 RF_NO_VDDPA_TYPICAL_POWER

```
#define RF_NO_VDDPA_TYPICAL_POWER 0
```

Location: rffe.h:95

Typical RF output power when VDDPA is not used, with default trims.

16.19.2.18 RF_DEFAULT_POWER

```
#define RF_DEFAULT_POWER 0
```

Location: rffe.h:98

RF output power used by default.

16.19.2.19 RF_MIN_POWER

```
#define RF_MIN_POWER -17
```

Location: rffe.h:101

Minimum possible RF output power.

16.19.2.20 PA_PWR_BYTE_0DBM

```
#define PA_PWR_BYTE_0DBM 0x0C
```

Location: rffe.h:104

RF Output Power code for 0dBm.

16.19.2.21 PA_ENABLE_BIAS_SETTING

```
#define PA_ENABLE_BIAS_SETTING 0xF3
```

Location: rffe.h:107

Power amplifier bias enable.

16.19.2.22 PA_DISABLE_BIAS_SETTING

```
#define PA_DISABLE_BIAS_SETTING 0x73
```

Location: rffe.h:110

Power amplifier bias disable.

16.19.2.23 SW_CTRL_DELAY_3_BYTE

```
#define SW_CTRL_DELAY_3_BYTE ((uint8_t)0x2U)
```

Location: rffe.h:114

Switch control delay.

16.19.2.24 RAMPUP_DELAY_3_BYTE

```
#define RAMPUP_DELAY_3_BYTE ((uint8_t)0x2U)
```

Location: rffe.h:117

Ramp-up delay.

16.19.2.25 DISABLE_DELAY_3_BYTE

```
#define DISABLE_DELAY_3_BYTE ((uint8_t)0x2U)
```

Location: rffe.h:120

Disable delay.

16.19.2.26 ERRNO_TX_POWER_MARKER

```
#define ERRNO_TX_POWER_MARKER 0x30
```

Location: rffe.h:124

Error marker for RFFE errors.

16.19.2.27 ERRNO_NO_TRIMS

```
#define ERRNO_NO_TRIMS (0x01 | ERRNO_TX_POWER_MARKER)
```

Location: rffe.h:127

No trims found when attempting to adjust voltage rails.

16.19.2.28 ERRNO_RFFE_MISSINGSETTING_ERROR

```
#define ERRNO_RFFE_MISSINGSETTING_ERROR (0x02 | ERRNO_TX_POWER_MARKER)
```

Location: rffe.h:130

Setting does not exist.

16.19.2.29 ERRNO_RFFE_INVALIDSETTING_ERROR

```
#define ERRNO_RFFE_INVALIDSETTING_ERROR (0x03 | ERRNO_TX_POWER_MARKER)
```

Location: rffe.h:133

Setting is not possible.

16.19.2.30 ERRNO_RFFE_VCC_INSUFFICIENT

```
#define ERRNO_RFFE_VCC_INSUFFICIENT (0x04 | ERRNO_TX_POWER_MARKER)
```

Location: rffe.h:136

VCC is too low to increase VDDRF sufficiently to suppor the requested RF output power.

16.19.2.31 WARNING_RFFE_VLOW_POWER_STATE

```
#define WARNING_RFFE_VLOW_POWER_STATE (0x05 | ERRNO_TX_POWER_MARKER)
```

Location: rffe.h:140

Warning that the device is in a very low RF output power state.

16.19.2.32 WARNING_RFFE_PA_ENABLED_STATE

```
#define WARNING_RFFE_PA_ENABLED_STATE (0x06 | ERRNO_TX_POWER_MARKER)
```

Location: rffe.h:143

Warning that the device has the power amplifier enabled.

16.19.2.33 CONVERT

```
#define CONVERT ((uint32_t)((x * 1000) >> 13))
```

Location: rffe.h:154

Converts an ADC code to a voltage, calculated as follows voltage = adc_code * (2 V * 1000 [mV]/1 V / 2^14 steps).

Returns:

The voltage output in mV

Assumptions

Low frequency mode for the ADC is used, meaning that the resolution of the ADC is 14-bits. CONVERT provides voltage level as a milliVolt value based on the input ADC code.

Parameters

Direction	*Name*	Description
in	*x*	the ADC code input

16.19.2.34 SWAP

```
#define SWAP ((t) = (a), (a) = (b), (b) = (t))
```

Location: rffe.h:164

Swap the values in variables a and b.

Returns:

a holds the value previously in b

Returns:

b holds the value previously in a

Parameters

Direction	*Name*	Description
in	*a*	holds the value that must go to b
in	*b*	holds the value that must go to a
in	*t*	a temporary buffer for the swap

16.19.2.35 SYS_RFFE_SETTXPOWER

```
#define SYS_RFFE_SETTXPOWER Sys_RFFE_SetTXPower(target, DEF_CHANNEL, VDDPA_DIS)
```

Location: rffe.h:208

Set the TX Power according to the desired target value with an accuracy of +/-1 dBm for +6 dBm to -17 dBm.

This function sets VDDRF, VDDPA, and PA_PWR (RF_REG1A) when applicable. Note: This function provides RF TX power configurations that match the requested levels, without considering the potential for increased power consumption due to the use of VDDPA. target Target transmission power in the range from -17 to +6 dBm in 1 dBm increments.

Returns:

Return value error value, if any

Example Code for SYS_RFFE_SETTXPOWER

```
// Reads the current register settings and measures VDDRF, if requried, to
// determine the current TX power setting.
result = SYS_RFFE_SETTXPOWER(0);
```

Returns:

Return value error value, if any

Example Code for SYS_RFFE_SETTXPOWER

```
// Reads the current register settings and measures VDDRF, if requried, to
// determine the current TX power setting.
result = SYS_RFFE_SETTXPOWER(0);
```

Parameters

Direction	*Name*	Description
in	*target*	Target transmission power in the range from -17 to +6 dBm in 1 dBm increments.

16.19.3 RFFE Radio Frequency Front End Function Documentation

16.19.3.1 Sys_RFFE_GetTXPower

```
int8_t Sys_RFFE_GetTXPower(uint32_t lsad_channel)
```

Location: rffe.h:173

Retrieve the current setting for RF output power by using the values retrieved from the appropriate registers.

Returns:

The currently set TX output power. Returns -100 in error state.

Example Code for Sys_RFFE_GetTXPower

```
// Use LSAD channel 0 to measure VDDRF if neccessary to determine the currently
// set TX output power.
result = Sys_RFFE_GetTXPower(0);
```

Parameters

Direction	Name	Description
in	*lsad_channel*	The LSAD channel used for measuring VDDRF

16.19.3.2 Sys_RFFE_SetTXPower

```
uint32_t Sys_RFFE_SetTXPower(int8_t target, uint8_t lsad_channel, bool pa_en)
```

Location: rffe.h:193

Set the TX Power according to the desired target value with an accuracy of +/-1 dBm for +6 dBm to -17 dBm.

This function sets VDDRF, VDDPA, and PA_PWR (RF_REG1A) when applicable. Note: This function provides RF TX power configurations that match the requested levels, without considering the potential for increased power consumption due to the use of VDDPA. target Target transmission power in the range from -17 to +6 dBm in 1 dBm increments. lsad_channel LSAD channel to use to measure rails, if necessary. pa_en If 1, the power amplifier will be enabled, otherwise, it will be disabled. The power amplifier will be enabled regardless if 'target' is greater than 2.

Returns:

Return value error value, if any

Example Code for Sys_RFFE_SetTXPower

```
// Set the TX power for the device to 6 dBm, uses LSAD channel 0 to measure
// (unused in the case of 6 dBm), and enables VDDPA.
result = Sys_RFFE_SetTXPower(6, 0, VDDPA_EN);
```

Returns:

Return value error value, if any

Example Code for Sys_RFFE_SetTXPower

```
// Set the TX power for the device to 6 dBm, uses LSAD channel 0 to measure
// (unused in the case of 6 dBm), and enables VDDPA.
result = Sys_RFFE_SetTXPower(6, 0, VDDPA_EN);
```

Parameters

Direction	Name	Description
in	*target*	Target transmission power in the range from -17 to +6 dBm in 1 dBm increments.
in	*lsad_channel*	LSAD channel to use to measure rails, if necessary.
in	*pa_en*	If 1, the power amplifier will be enabled, otherwise, it will be disabled. The power amplifier will be enabled regardless if 'target' is greater than 2.

16.20 Real-Time Clock

Real-time clock hardware abstraction layer.

16.20.1 Summary

Functions

- Sys_RTC_Config : Configure RTC block.
- Sys_RTC_Count_Threshold : Configure RTC Counter Threshold.
- Sys_RTC_Value_Seconds : Read the current value of the RTC timer in seconds.
- Sys_RTC_Value : Read the current value of the RTC timer.

16.20.2 Real-Time Clock Function Documentation

16.20.2.1 Sys_RTC_Config

```
void Sys_RTC_Config(uint32_t rtc_cfg, uint32_t rtc_ctrl)
```

Location: rtc.h:49

Configure RTC block.

Example Code for Sys_RTC_Config

```
    // Enable clock and alarm events and enable the RTC
    // Configure the RTC clock to trigger once a second
    // Use external 32kHz crystal as the clock
    Sys_RTC_Config(RTC_CLK_SRC_XTAL32K | RTC_CLOCK_1S,
                   RTC_ENABLE_CLOCK_EVENT | RTC_ENABLE_ALARM_EVENT | RTC_ENABLE);
```

Parameters

Direction	Name	Description
in	*rtc_cfg*	Select the RTC clock event and clock source; use RTC_CLOCK_*, and RTC_CLK_SRC_*
in	*rtc_ctrl*	RTC control register; use RTC_FORCE_CLOCK, RTC_[DISABLE | ENABLE]_CLOCK_EVENT, RTC_[DISABLE | ENABLE]_ALARM_EVENT, RTC_RESET, and RTC_[DISABLE | ENABLE]

16.20.2.2 Sys_RTC_Count_Threshold

```
void Sys_RTC_Count_Threshold(uint32_t threshold)
```

Location: rtc.h:63

Configure RTC Counter Threshold.

Example Code for Sys_RTC_Count_Threshold

```
// Set the RTC counter threshold to 32767 (1 second, assuming an RTC clock
// frequency of 32768 Hz)
result = Sys_RTC_Count_Threshold(32767);
```

Parameters

Direction	Name	Description
in	*threshold*	Compare value for the RTC counter; use RTC_COUNT_THRES_*

16.20.2.3 Sys_RTC_Value_Seconds

```
uint32_t Sys_RTC_Value_Seconds()
```

Location: rtc.h:73

Read the current value of the RTC timer in seconds.

Returns:

RTC timer current value in seconds

Example Code for Sys_RTC_Value_Seconds

```
// Return the current value of the RTC counter, converted to seconds
result = Sys_RTC_Value_Seconds();
```

16.20.2.4 Sys_RTC_Value

```
uint32_t Sys_RTC_Value()
```

Location: rtc.h:83

Read the current value of the RTC timer.

Returns:

RTC timer counter current value

Example Code for Sys_RTC_Value

```
// Return the current value of the RTC counter
result = Sys_RTC_Value();
```

16.21 SAR_ADC

SAR_ADC hardware abstraction layer.

16.21.1 Summary

Macros

- VDDSAR_FREQ_THRESHOLD
- MAX_SENSOR_CLK_HIGH
- MAX_SENSOR_CLK_LOW

Functions

- Sys_Calibrate_SARADC : Calibrate the SAR ADC, generating gain and offset values.

16.21.2 SAR_ADC Macro Definition Documentation

16.21.2.1 VDDSAR_FREQ_THRESHOLD

```
#define VDDSAR_FREQ_THRESHOLD (2500)
```

Location: sar_adc.h:44

16.21.2.2 MAX_SENSOR_CLK_HIGH

```
#define MAX_SENSOR_CLK_HIGH (16000000UL)
```

Location: sar_adc.h:48

16.21.2.3 MAX_SENSOR_CLK_LOW

```
#define MAX_SENSOR_CLK_LOW (2000000UL)
```

Location: sar_adc.h:52

16.21.3 SAR_ADC Function Documentation

16.21.3.1 Sys_Calibrate_SARADC

```
void Sys_Calibrate_SARADC(uint32_t vddsar_mv, uint32_t supply_src, uint32_t * offset,
float * gain)
```

Location: sar_adc.h:72

Calibrate the SAR ADC, generating gain and offset values.

The gain and offset values are stored in the SAR ADC but are returned here as well. vddsar_mv Set the SAR supply voltage. 2.5 to 3.3 V sets SENSOR_CLK between 8 and 16 MHz. Less than 2.5 V sets SENSOR_CLK to 2 MHz. supply_src Select the SAR supply source: use SAR_SUPPLY_BY_VBAT, SAR_SUPPLY_BY_GPIO9 offset Gets the offset stored in SAR_ADC gain Gets the calibrated gain correction

Assumptions

The SAR ADC is configured to use VBAT for the calibration.

Example Code for Sys_Calibrate_SARADC

```
     // Calibrate the SAR ADC with SAR positive input having signal between VSSA
     // and 3000mV.
     // Set gain_correction to the calibrated gain correction
     // Set sar_adc_offset to the offset stored in SAR-ADC (i.e. 8192 levels)
     float gain_correction = 0;
     uint32_t sar_adc_offset = 0;
     Sys_Calibrate_SARADC(SAR_ADC_VBAT_SUPPLY_MV, SAR_SUPPLY_BY_VBAT, &sar_adc_
offset, &gain_correction);
```

Assumptions

The SAR ADC is configured to use VBAT for the calibration.

Example Code for Sys_Calibrate_SARADC

```
     // Calibrate the SAR ADC with SAR positive input having signal between VSSA
     // and 3000mV.
     // Set gain_correction to the calibrated gain correction
     // Set sar_adc_offset to the offset stored in SAR-ADC (i.e. 8192 levels)
     float gain_correction = 0;
     uint32_t sar_adc_offset = 0;
     Sys_Calibrate_SARADC(SAR_ADC_VBAT_SUPPLY_MV, SAR_SUPPLY_BY_VBAT, &sar_adc_
offset, &gain_correction);
```


Parameters

Direction	Name	Description
in	*vddsar_mv*	Set the SAR supply voltage. 2.5 to 3.3 V sets SENSOR_CLK between 8 and 16 MHz. Less than 2.5 V sets SENSOR_CLK to 2 MHz.
in	*supply_src*	Select the SAR supply source: use SAR_SUPPLY_BY_VBAT, SAR_SUPPLY_BY_GPIO9
out	*offset*	Gets the offset stored in SAR_ADC
out	*gain*	Gets the calibrated gain correction

16.22 Simple Assertions

Simple Assertion (SASSERT) hardware abstraction layer.

16.22.1 Summary

Macros

- SYS_ASSERT : Assertion handler; default behavior is no operation.

16.22.2 Simple Assertions Macro Definition Documentation

16.22.2.1 SYS_ASSERT

```
#define SYS_ASSERT False
```

Location: sassert.h:47

Assertion handler; default behavior is no operation.

16.23 Ultra-Low Power Data Acquisition Subsystem

Ultra-low power data acquisition (sensor) subsystem hardware abstraction layer.

16.23.1 Summary

Functions

- Sys_Sensor_SARConfig : Configure the SAR-ADC interface.
- Sys_Sensor_PulseCountConfig : Configure sensor pulse count state.
- Sys_Sensor_StorageConfig : Configure data storage settings.
- Sys_Sensor_DelayConfig : Configure sensor delay clocks and length.
- Sys_Sensor_Enable : Enable the sensor.
- Sys_Sensor_Disable : Disable the sensor.
- Sys_Sensor_TimerReset : The sensor timer counter, the sensor timer enable and the ADC counter are reset.
- Sys_Sensor_CurrentState : Read the current status and delay state of the sensor interface.
- Sys_Sensor_CurrentCountValue : Read the current value of the sensor's main counter.

16.23.2 Ultra-Low Power Data Acquisition Subsystem Function Documentation

16.23.2.1 Sys_Sensor_SARConfig

```
void Sys_Sensor_SARConfig(uint32_t sar_cfg, uint32_t sar_ctrl, uint32_t clk_cfg)
```

Location: sensor.h:52

Configure the SAR-ADC interface.

Example Code for Sys_Sensor_SARConfig

```
// Configure the SAR to measure on GPIO0 and GPIO1.
//   - Define positive input and negative input signal as supply voltage/2
//   - Do not use DMA
//   - Data buffer updated on finished conversion
//   - SAR supply is only on when used
//   - SAR supply is VBAT
//   - Gated 2's complement offset compensated output (gated means output is
//       zero until valid data becomes available)
//   - Data sampling lasts 1 cycle
//   - Create 12 bit samples
//   - Stop continuous conversions
//   - Use standby clock as the sensor clock source
Sys_Sensor_SARConfig(SAR_PRE_SEL_GPIO_1                |
                     SAR_PRE_SEL_GPIO_0                |
                     SAR_IN_P_SRC_SAR_SUPPLY_DIV2      |
                     SAR_IN_N_SRC_SAR_SUPPLY_DIV2      |
                     SAR_DATA_OUT_RX_DMA_DISABLED      |
                     SAR_DATA_OUT_UPDATE_AUTO          |
                     SAR_SUPPLY_AUTO                   |
                     SAR_SUPPLY_BY_VBAT,
                     SAR_SEL_GATED_SIGNED_COMPENSATED  |
                     SAR_NSAMPLING_CYCLE_1             |
                     SAR_CONV_12BIT                    |
                     SAR_STOP_CONTINUOUS,
                     SENSOR_CLK_STANDBY_BYTE);
```

Parameters

Direction	Name	Description
in	*sar_cfg*	SAR-ADC configuration; use SAR_PRE_SEL_GPIO_*, SAR_IN_*_SRC_*, SAR_DATA_OUT_RX_DMA_[ENABLED | DISABLED], SAR_DATA_OUT_UPDATE_[AUTO | ENABLE], SAR_SUPPLY_[AUTO | ENABLE], and SAR_SUPPLY_BY_[VBAT | VDDA]
in	*sar_ctrl*	SAR-ADC control configuration; use SAR_SEL_*, SAR_NSAMPLING_CYCLE_*, SAR_[CONV_12BIT | CONV_14BIT | CAL_WEIGHT | AUTO_ZERO], SAR_[START_SINGLE | START_CONTINUOUS | STOP_CONTINUOUS)
in	*clk_cfg*	select interface clock source; use SENSOR_CLK_[STANDBY | SYSCLK_DIV]_BYTE

16.23.2.2 Sys_Sensor_PulseCountConfig

```
void Sys_Sensor_PulseCountConfig(uint32_t pc_cfg, uint32_t clk_cfg)
```

Location: sensor.h:73

Configure sensor pulse count state.

Example Code for Sys_Sensor_PulseCountConfig

```
    //   - Set pulse count source to constant high
    //   - Set total count state to be 4 periods of sensor clock
    //   - Select standby clock as sensor clock source
    Sys_Sensor_PulseCountConfig(PC_SRC_CONST_HIGH | COUNT_INT_4, SENSOR_CLK_STANDBY_
BYTE)
```

Parameters

Direction	Name	Description
in	*pc_cfg*	Pulse-counter configuration; use COUNT_INT_*, PC_SRC_*, PC_COUNT_ON_[HIGH_LEVEL | RISING_EDGE]
in	*clk_cfg*	select interface clock source; use SENSOR_CLK_[STANDBY | SYSCLK_DIV]_BYTE

16.23.2.3 Sys_Sensor_StorageConfig

```
void Sys_Sensor_StorageConfig(uint32_t sum_en, uint32_t nbr_samples, uint32_t threshold_min, uint32_t threshold_max, uint32_t store_en, uint32_t fifo_size, uint32_t fifo_cfg)
```

Location: sensor.h:107

Configure data storage settings.

Example Code for Sys_Sensor_StorageConfig

```
//   - Disable summing of sequential samples
//   - Number of data samples to generate before wakeup generated
//   - Enable sensor threshold and set it to a value of 1
//   - Enable storing of sensor data samples in the sensor FIFO
//   - Set sensor FIFO size to 1 sample
//   - Disable interrupts and dma transfer of data from FIFO
Sys_Sensor_StorageConfig(SENSOR_SUMMATION_DISABLED, SENSOR_NBR_SAMPLES_1,
                         SENSOR_THRESHOLD_MIN_ENABLED | SENSOR_THRESHOLD_MIN_1,
                         SENSOR_THRESHOLD_MAX_ENABLED | SENSOR_THRESHOLD_MAX_1,
                         SENSOR_FIFO_STORE_ENABLED, SENSOR_FIFO_SIZE1,
                         FIFO_RX_DMA_DISABLED | FIFO_RX_INT_ENABLED);
```

Parameters

Direction	Name	Description
in	*sum_en*	Enable summation mode; use SENSOR_SUMMATION_[ENABLED | DISABLED]
in	*nbr_samples*	Number of samples to store before wakeup in sensor detect mode or for impedance measurement; use SENSOR_NBR_SAMPLES_*
in	*threshold_min*	Sensor data level threshold (min) for wakeup; use SENSOR_THRESHOLD_MIN_*, SENSOR_THRESHOLD_MIN_[DISABLED | ENABLED]
in	*threshold_max*	Sensor data level threshold (max) for wakeup; use SENSOR_THRESHOLD_MAX_*, SENSOR_THRESHOLD_MAX_[DISABLED | ENABLED]
in	*store_en*	Enable storing samples in FIFO
in	*fifo_size*	Number of samples to store in FIFO before wakeup of core.
in	*fifo_cfg*	Enable FIFO interrupts and/or DMA triggers; use FIFO_RX_DMA_[ENABLED | DISABLED], and FIFO_RX_INT_[ENABLED | DISABLED]

16.23.2.4 Sys_Sensor_DelayConfig

```
void Sys_Sensor_DelayConfig(uint32_t dly_cfg)
```

Location: sensor.h:129

Configure sensor delay clocks and length.

Use sub-register defines.

NOTE: Clock can be configured for 32 kHz (0) or 1 kHz (1-default)

dly_cfg select interface clock source; use DLY_*_SHORT, DLY_DIV_[ENABLED | DISABLED]_SHORT, and DLY_[USED | NOT_USED]_SHORT

Example Code for Sys_Sensor_DelayConfig

```
//   - Enable delay state
//   - Delay state runs at sensor clock divided by 32
//   - Delay state is 256 sensor clock/32 periods
Sys_Sensor_DelayConfig(DLY_USED | DLY_DIV_ENABLED | DLY_256)
```

NOTE: Clock can be configured for 32 kHz (0) or 1 kHz (1-default)

Example Code for Sys_Sensor_DelayConfig

```
//   - Enable delay state
//   - Delay state runs at sensor clock divided by 32
//   - Delay state is 256 sensor clock/32 periods
Sys_Sensor_DelayConfig(DLY_USED | DLY_DIV_ENABLED | DLY_256)
```

Parameters

Direction	Name	Description
in	*dly_cfg*	select interface clock source; use DLY_*_SHORT, DLY_DIV_[ENABLED | DISABLED]_SHORT, and DLY_[USED | NOT_USED]_SHORT

16.23.2.5 Sys_Sensor_Enable

```
void Sys_Sensor_Enable()
```

Location: sensor.h:139

Enable the sensor.

In secure mode, also enable power to the sensor.

Example Code for Sys_Sensor_Enable

```
// Enable the ULP power data acquisition (sensor) subsystem
Sys_Sensor_Enable()
```

Example Code for Sys_Sensor_Enable

```
// Enable the ULP power data acquisition (sensor) subsystem
Sys_Sensor_Enable()
```

16.23.2.6 Sys_Sensor_Disable

```
void Sys_Sensor_Disable()
```

Location: sensor.h:157

Disable the sensor.

In secure mode, also disable power to the sensor.

Example Code for Sys_Sensor_Disable

```
// Disable the ULP power data acquisition (sensor) subsystem
Sys_Sensor_Disable()
```

Example Code for Sys_Sensor_Disable

```
// Disable the ULP power data acquisition (sensor) subsystem
Sys_Sensor_Disable()
```

16.23.2.7 Sys_Sensor_TimerReset

```
void Sys_Sensor_TimerReset()
```

Location: sensor.h:176

The sensor timer counter, the sensor timer enable and the ADC counter are reset.

Example Code for Sys_Sensor_TimerReset

```
// The sensor timer counter, the sensor timer enable and the ADC counter are reset
Sys_Sensor_TimerReset()
```

16.23.2.8 Sys_Sensor_CurrentState

```
uint8_t Sys_Sensor_CurrentState()
```

Location: sensor.h:188

Read the current status and delay state of the sensor interface.

Returns:

current state of the sensor interface.

Example Code for Sys_Sensor_CurrentState

```
// Read the current state of the sensor interface
Sys_Sensor_CurrentState()
```

16.23.2.9 Sys_Sensor_CurrentCountValue

```
uint32_t Sys_Sensor_CurrentCountValue()
```

Location: sensor.h:200

Read the current value of the sensor's main counter.

Returns:

current value of the main counter.

Example Code for Sys_Sensor_CurrentCountValue

```
// Read the current value of the main sensor counter
Sys_Sensor_CurrentCountValue()
```

16.24 SPI

Serial Peripheral Interface (SPI) hardware abstraction layer.

16.24.1 Summary

Macros

- SPI_CONFIG_MASK : Mask for the SPI_CFG register.
- SPI_PADS_NUM : Number of pads used for the SPI interface, for a single instance.
- IS_GPIO_REAL : Verify if the GPIO number corresponds to an actual pin, as opposed to a constant high/low value.
- SYS_SPI_CONFIG : Configure the specified SPI interface's operation and controller information.
- SYS_SPI_TRANSFERCONFIG : Configure the SPI transfer information for the specified SPI instance.
- SYS_SPI_READ : Generate clock and CS to read data from SPI interface.
- SYS_SPI_WRITE : Generate clock and CS to write data to SPI interface.
- SYS_SPI_GPIOCONFIG : Configure four GPIOs for the SPI0 interface.
- SYS_DSPI_GPIOCONFIG : Configure four GPIOs for the SPI0 interface for DSPI.
- SYS_QSPI_GPIOCONFIG : Configure six GPIOs for the SPI0 interface for QSPI cfg GPIO pin configuration for the SPI pads clk GPIO to use as the QSPI clock pad cs GPIO to use as the QSPI chip select pad io0 GPIO to use as the QSPI io0 io1 GPIO to use as the QSPI io1 io2 GPIO to use as the QSPI io2 io3 GPIO to use as the QSPI io3.

Functions

- Sys_SPI_Config : Configure the specified SPI interface's operation and controller information.
- Sys_SPI_TransferConfig : Configure the SPI transfer information for the specified SPI instance.
- Sys_SPI_Read : Generate clock and CS to read data from SPI interface.
- Sys_SPI_Write : Generate clock and CS to write data to SPI interface.
- Sys_SPI_GPIOConfig : Configure four GPIOs for the specified SPI interface.
- Sys_DSPI_GPIOConfig : Configure four GPIOs for the specified SPI interface.
- Sys_QSPI_GPIOConfig : Configure six GPIOs for the specified SPI interface.

16.24.2 SPI Macro Definition Documentation

16.24.2.1 SPI_CONFIG_MASK

```
#define SPI_CONFIG_MASK ((1 << SPI_CFG_TX_DMA_ENABLE_Pos)           | \
    (1 << SPI_CFG_RX_DMA_ENABLE_Pos)           | \
    (1 << SPI_CFG_TX_END_INT_ENABLE_Pos)       | \
    (1 << SPI_CFG_TX_START_INT_ENABLE_Pos)     | \
    (1 << SPI_CFG_RX_INT_ENABLE_Pos)           | \
    (1 << SPI_CFG_CS_RISE_INT_ENABLE_Pos)      | \
    (1 << SPI_CFG_OVERRUN_INT_ENABLE_Pos)      | \
```

```
(1 << SPI_CFG_UNDERRUN_INT_ENABLE_Pos)      | \
(1 << SPI_CFG_MODE_Pos)                     | \
SPI_CFG_MODE_Mask                           | \
(1 << SPI_CFG_WORD_SIZE_Pos)                | \
SPI_CFG_WORD_SIZE_Mask                      | \
(1 << SPI_CFG_PRESCALE_Pos)                 | \
SPI_CFG_PRESCALE_Mask                       | \
(1 << SPI_CFG_CLK_POLARITY_Pos)             | \
(1 << SPI_CFG_SLAVE_Pos))
```

Location: spi.h:61

Mask for the SPI_CFG register.

16.24.2.2 SPI_PADS_NUM

```
#define SPI_PADS_NUM 6
```

Location: spi.h:81

Number of pads used for the SPI interface, for a single instance.

16.24.2.3 IS_GPIO_REAL

```
#define IS_GPIO_REAL (gpio_number < GPIO_PAD_COUNT)
```

Location: spi.h:88

Verify if the GPIO number corresponds to an actual pin, as opposed to a constant high/low value.

16.24.2.4 SYS_SPI_CONFIG

```
#define SYS_SPI_CONFIG Sys_SPI_Config(SPI, (config))
```

Location: spi.h:341

Configure the specified SPI interface's operation and controller information.

Example Code for SYS_SPI_CONFIG

```
// Configure the default SPI interface's operation and controller information:
//  - Master mode
//  - Use 8-bit words
//  - Select interrupts enabled
//  - Prescale the SPI interface clock by 32
SYS_SPI_CONFIG((SPI_MODE_QSPI | SPI_WORD_SIZE_8 | SPI_TX_START_INT_ENABLE |
                SPI_RX_INT_ENABLE | SPI_OVERRUN_INT_ENABLE |
                SPI_PRESCALE_32 | SPI_UNDERRUN_INT_ENABLE));
```

Parameters

Direction	*Name*	Description
in	*config*	Interface operation configuration; use SPI_SELECT_[MASTER | SLAVE], SPI_CLK_POLARITY_[NORMAL | INVERSE], SPI_PRESCALE_*, SPI_WORD_SIZE_*, SPI_MODE_[SPI | DSPI | QSPI] SPI_UNDERRUN_INT_[ENABLE | DISABLE], SPI_OVERRRUN_INT_[ENABLE | DISABLE], SPI_CS_RISE_INT_[ENABLE | DISABLE], SPI_RX_INT_[ENABLE | DISABLE] SPI_TX_INT_[ENABLE | DISABLE] SPI_RX_DMA_[ENABLE | DISABLE] SPI_TX_DMA_[ENABLE | DISABLE]

16.24.2.5 SYS_SPI_TRANSFERCONFIG

```
#define SYS_SPI_TRANSFERCONFIG Sys_SPI_TransferConfig(SPI, (config))
```

Location: spi.h:359

Configure the SPI transfer information for the specified SPI instance.

Example Code for SYS_SPI_TRANSFERCONFIG

```
// Enable and configure default SPI interface to read operation mode
SYS_SPI_TRANSFERCONFIG(SPI_ENABLE | SPI_MODE_READ);
```

Parameters

Direction	*Name*	Description
in	*config*	Interface transfer configuration; use SPI_ENABLE, SPI_DISABLE, SPI_RESET, SPI_START, SPI_MODE_READ_WRITE, SPI_MODE_READ, SPI_MODE_WRITE, SPI_MODE_NOP, SPI_CS_0, SPI_CS_1,

16.24.2.6 SYS_SPI_READ

```
#define SYS_SPI_READ Sys_SPI_Read(SPI)
```

Location: spi.h:369

Generate clock and CS to read data from SPI interface.

Returns:

Data read from the SPI interface

Assumptions

SPI is configured as master mode and transfer operation mode is SPI_MODE_READ_WRITE(full duplex) or (SPI_MODE_READ) half duplex

Example Code for SYS_SPI_READ

```
// Generate clock and CS to read data from the default SPI interface
SYS_SPI_READ();
```

16.24.2.7 SYS_SPI_WRITE

```
#define SYS_SPI_WRITE Sys_SPI_Write(SPI, (data))
```

Location: spi.h:379

Generate clock and CS to write data to SPI interface.

Assumptions

SPI is configured as master mode and transfer operation mode is SPI_MODE_READ_WRITE(full duplex) or (SPI_MODE_WRITE) half duplex

Example Code for SYS_SPI_WRITE

```
// Generate clock and CS to write data to the default SPI interface
SYS_SPI_WRITE(0xFF);
```

Parameters

Direction	*Name*	Description
in	*data*	Data to be sent over SPI

16.24.2.8 SYS_SPI_GPIOCONFIG

```
#define SYS_SPI_GPIOCONFIG Sys_SPI_GPIOConfig(SPI, (slave), (cfg), (clk), (cs), (seri), (sero))
```

Location: spi.h:392

Configure four GPIOs for the SPI0 interface.

Example Code for SYS_SPI_GPIOCONFIG

```
// Configure GPIOs 0, 1, 2, and 3 for the default SPI interface with
// low-pass filter disabled, 8X drive-strength, and 1 kOhm pull-up resistors
SYS_SPI_GPIOCONFIG(GPIO0, (GPIO_LPF_DISABLE |GPIO_8X_DRIVE |
                   GPIO_1K_PULL_UP), GPIO0, GPIO1, GPIO2, GPIO3);
```

Parameters

Direction	*Name*	Description
in	*slave*	SPI master/slave configuration; use SPI*_SELECT_ [MASTER | SLAVE]
in	*cfg*	GPIO pin configuration for the SPI pads
in	*clk*	GPIO to use as the SPI clock pad
in	*cs*	GPIO to use as the SPI chip select pad
in	*seri*	GPIO to use as the SPI serial input pad
in	*sero*	GPIO to use as the SPI serial output pad

16.24.2.9 SYS_DSPI_GPIOCONFIG

```
#define SYS_DSPI_GPIOCONFIG Sys_DSPI_GPIOConfig(SPI, (cfg), (clk), (cs), (io0), (io1))
```

Location: spi.h:404

Configure four GPIOs for the SPI0 interface for DSPI.

Example Code for SYS_DSPI_GPIOCONFIG

```
// Configure GPIOs 0, 1, 2, and 3 for the default DSPI interface with
// low-pass filter disabled, 8X drive-strength, and 1 kOhm pull-up resistors
SYS_DSPI_GPIOCONFIG((GPIO_LPF_DISABLE | GPIO_8X_DRIVE |
                    GPIO_1K_PULL_UP), GPIO0, GPIO1, GPIO2, GPIO3);
```

Parameters

Direction	*Name*	Description
in	*cfg*	GPIO pin configuration for the SPI pads
in	*clk*	GPIO to use as the DSPI clock pad
in	*cs*	GPIO to use as the DSPI chip select pad
in	*io0*	GPIO to use as the DSPI io0
in	*io1*	GPIO to use as the DSPI io1

16.24.2.10 SYS_QSPI_GPIOCONFIG

```
#define SYS_QSPI_GPIOCONFIG Sys_QSPI_GPIOConfig(SPI, (cfg), (clk), (cs), (io0), (io1), \
                            (io2), (io3))
```

Location: spi.h:418

Configure six GPIOs for the SPI0 interface for QSPI cfg GPIO pin configuration for the SPI pads clk GPIO to use as the QSPI clock pad cs GPIO to use as the QSPI chip select pad io0 GPIO to use as the QSPI io0 io1 GPIO to use as the QSPI io1 io2 GPIO to use as the QSPI io2 io3 GPIO to use as the QSPI io3.

Example Code for SYS_QSPI_GPIOCONFIG

```
// Configure GPIOs 0, 1, 2, 3, 4, and 5 for the default QSPI interface with
// low-pass filter disabled, 8X drive-strength, and 1 kOhm pull-up resistors
SYS_QSPI_GPIOCONFIG((GPIO_LPF_DISABLE | GPIO_8X_DRIVE | GPIO_1K_PULL_UP),
                    GPIO0, GPIO1, GPIO2, GPIO3, GPIO4, GPIO5);
```

16.24.3 SPI Function Documentation

16.24.3.1 Sys_SPI_Config

```
void Sys_SPI_Config(SPI_Type * spi, uint32_t config)
```

Location: spi.h:109

Configure the specified SPI interface's operation and controller information.

Example Code for Sys_SPI_Config

```
// Configure the SPI interface's operation and controller information:
//  - Master mode
//  - Use 8-bit words
//  - Select interrupts enabled
//  - Prescale the SPI interface clock by 32
Sys_SPI_Config(SPI, SPI_SELECT_MASTER | SPI_WORD_SIZE_8 | SPI_TX_START_INT_ENABLE |
               SPI_RX_INT_ENABLE | SPI_OVERRUN_INT_ENABLE |
               SPI_PRESCALE_32 | SPI_UNDERRUN_INT_ENABLE);
```

Parameters

Direction	Name	Description
in	*spi*	Pointer to the SPI instance
in	*config*	Interface operation configuration; use SPI_SELECT_[MASTER | SLAVE], SPI_CLK_POLARITY_[NORMAL | INVERSE], SPI_PRESCALE_*, SPI_WORD_SIZE_*, SPI_MODE_[SPI | DSPI | QSPI] SPI_UNDERRUN_INT_[ENABLE | DISABLE], SPI_OVERRRUN_INT_[ENABLE | DISABLE], SPI_CS_RISE_INT_[ENABLE | DISABLE], SPI_RX_INT_[ENABLE | DISABLE] SPI_TX_INT_[ENABLE | DISABLE] SPI_RX_DMA_[ENABLE | DISABLE] SPI_TX_DMA_[ENABLE | DISABLE]

16.24.3.2 Sys_SPI_TransferConfig

```
void Sys_SPI_TransferConfig(SPI_Type * spi, uint32_t config)
```

Location: spi.h:132

Configure the SPI transfer information for the specified SPI instance.

Example Code for Sys_SPI_TransferConfig

```
// Enable and configure the SPI interface to read operation mode
Sys_SPI_TransferConfig(SPI, SPI_ENABLE | SPI_MODE_READ);
```

Parameters

Direction	Name	Description
in	*spi*	Pointer to the SPI instance
in	*config*	Interface transfer configuration; use SPI_ENABLE, SPI_DISABLE, SPI_RESET, SPI_START, SPI_MODE_READ_WRITE, SPI_MODE_READ, SPI_MODE_WRITE, SPI_MODE_NOP, SPI_CS_0, SPI_CS_1,

16.24.3.3 Sys_SPI_Read

```
uint32_t Sys_SPI_Read(const SPI_Type * spi)
```

Location: spi.h:147

Generate clock and CS to read data from SPI interface.

Returns:

data Data read from the SPI interface

Assumptions

SPI is configured as master mode and transfer operation mode is SPI_MODE_READ_WRITE(full duplex) or (SPI_MODE_READ) half duplex

Example Code for Sys_SPI_Read

```
// Generate clock and CS to read data from SPI interface
Sys_SPI_Read(SPI);
```

Parameters

Direction	Name	Description
in	*spi*	Pointer to the SPI instance

16.24.3.4 Sys_SPI_Write

```
void Sys_SPI_Write(SPI_Type * spi, uint32_t data)
```

Location: spi.h:162

Generate clock and CS to write data to SPI interface.

Assumptions

SPI is configured as master mode and transfer operation mode is SPI_MODE_READ_WRITE(full duplex) or (SPI_MODE_WRITE) half duplex

Example Code for Sys_SPI_Write

```
// Generate clock and CS to write data to SPI interface
Sys_SPI_Write(SPI, 0xFF);
```

Parameters

Direction	Name	Description
in	*spi*	Pointer to the SPI instance
in	*data*	Data to be sent over SPI

16.24.3.5 Sys_SPI_GPIOConfig

```
void Sys_SPI_GPIOConfig(const SPI_Type * spi, uint32_t slave, uint32_t cfg, uint32_t clk,
uint32_t cs, uint32_t seri, uint32_t sero)
```

Location: spi.h:181

Configure four GPIOs for the specified SPI interface.

Example Code for Sys_SPI_GPIOConfig

```
// Configure GPIOs 0, 1, 2, and 3 for the SPI0 interface with
// low-pass filter disabled, 8X drive-strength, and 1 kOhm pull-up resistors
Sys_SPI_GPIOConfig(SPI, GPIO0, (GPIO_LPF_DISABLE | GPIO_8X_DRIVE |
                   GPIO_1K_PULL_UP), GPIO0, GPIO1, GPIO2, GPIO3);
```

Parameters

Direction	Name	Description
in	*spi*	Pointer to the SPI instance
in	*slave*	SPI master/slave configuration; use SPI*_SELECT_ [MASTER | SLAVE]
in	*cfg*	GPIO pin configuration for the SPI pads
in	*clk*	GPIO to use as the SPI clock pad
in	*cs*	GPIO to use as the SPI chip select pad
in	*seri*	GPIO to use as the SPI serial input pad
in	*sero*	GPIO to use as the SPI serial output pad

16.24.3.6 Sys_DSPI_GPIOConfig

```
void Sys_DSPI_GPIOConfig(const SPI_Type * spi, uint32_t cfg, uint32_t clk, uint32_t cs,
uint32_t io0, uint32_t io1)
```

Location: spi.h:265

Configure four GPIOs for the specified SPI interface.

Example Code for Sys_DSPI_GPIOConfig

```
// Configure GPIOs 0, 1, 2, and 3 for the DSPI0 interface with
// low-pass filter disabled, 8X drive-strength, and 1 kOhm pull-up resistors
Sys_DSPI_GPIOConfig(SPI, (GPIO_LPF_DISABLE | GPIO_8X_DRIVE |
                    GPIO_1K_PULL_UP), GPIO0, GPIO1, GPIO2, GPIO3);
```

Parameters

Direction	Name	Description
in	*spi*	Pointer to the SPI instance
in	*cfg*	GPIO pin configuration for the SPI pads
in	*clk*	GPIO to use as the DSPI clock pad

Direction	Name	Description
in	*cs*	GPIO to use as the DSPI chip select pad
in	*io0*	GPIO to use as the DSPI io0
in	*io1*	GPIO to use as the DSPI io1

16.24.3.7 Sys_QSPI_GPIOConfig

```
void Sys_QSPI_GPIOConfig(const SPI_Type * spi, uint32_t cfg, uint32_t clk, uint32_t cs,
uint32_t io0, uint32_t io1, uint32_t io2, uint32_t io3)
```

Location: spi.h:297

Configure six GPIOs for the specified SPI interface.

Example Code for Sys_QSPI_GPIOConfig

```
// Configure GPIOs 0, 1, 2, 3, 4, and 5 for the QSPI0 interface with
// low-pass filter disabled, 8X drive-strength, and 1 kOhm pull-up resistors
Sys_QSPI_GPIOConfig(SPI, (GPIO_LPF_DISABLE | GPIO_8X_DRIVE | GPIO_1K_PULL_UP),
                    GPIO0, GPIO1, GPIO2, GPIO3, GPIO4, GPIO5);
```

Parameters

Direction	Name	Description
in	*spi*	Pointer to the SPI instance
in	*cfg*	GPIO pin configuration for the SPI pads
in	*clk*	GPIO to use as the QSPI clock pad
in	*cs*	GPIO to use as the QSPI chip select pad
in	*io0*	GPIO to use as the QSPI io0
in	*io1*	GPIO to use as the QSPI io1
in	*io2*	GPIO to use as the QSPI io2
in	*io3*	GPIO to use as the QSPI io3

16.25 General-Purpose Timer

General-purpose timer hardware abstraction layer.

16.25.1 Summary

Macros

- SYS_TIMER_CONFIG : Configures the default timer instance.
- SYS_TIMER_START : Starts the default timer instance.
- SYS_TIMER_STOP : Stops the default timer instance.

Functions

- Sys_Timer_Config : Configure timer instance.
- Sys_Timer_Start : Start or restart timer instance.
- Sys_Timer_Stop : Stop the timer instance.

16.25.2 General-Purpose Timer Macro Definition Documentation

16.25.2.1 SYS_TIMER_CONFIG

```
#define SYS_TIMER_CONFIG Sys_Timer_Config(TIMER, (cfg0), \
                                                    (cfg1), (timeout))
```

Location: timer.h:83

Configures the default timer instance.

16.25.2.2 SYS_TIMER_START

```
#define SYS_TIMER_START Sys_Timer_Start(TIMER)
```

Location: timer.h:87

Starts the default timer instance.

16.25.2.3 SYS_TIMER_STOP

```
#define SYS_TIMER_STOP Sys_Timer_Stop(TIMER)
```

Location: timer.h:90

Stops the default timer instance.

16.25.3 General-Purpose Timer Function Documentation

16.25.3.1 Sys_Timer_Config

```
void Sys_Timer_Config(TIMER_Type * timer, uint32_t cfg0, uint32_t cfg1, uint32_t timeout)
```

Location: timer.h:51

Configure timer instance.

Example Code for Sys_Timer_Config

```
// Configure TIMER2 instance:
//  - Divide the input clock frequency by 2
//  - Stop on 2nd Time-out occurrence and issue an interrupt
//  - Select the GPIO interrupt defined in GPIO_INT_CFG3
//  - GPIO interrupt single capture mode
//  - Free-run mode
//  - Long timeout
Sys_Timer_Config(TIMER2, TIMER_PRESCALE_2,
                 TIMER_MULTI_COUNT_2 |
                 TIMER_SRC_GPIO_INT3  |
                 TIMER_GPIO_INT_SINGLE |
                 TIMER_FREE_RUN, 0xFFFF);
```

Parameters

Direction	Name	Description
in	*timer*	Pointer to the timer instance
in	*cfg0*	Timer configuration 0; use TIMER_PRESCALE_*,
in	*cfg1*	Timer configuration 1; use TIMER_MULTI_COUNT_* [TIMER_SHOT_MODE | TIMER_FREE_RUN]
in	*timeout*	number of timer clock cycles before a timeout would occur

16.25.3.2 Sys_Timer_Start

```
void Sys_Timer_Start(TIMER_Type * timer)
```

Location: timer.h:65

Start or restart timer instance.

Example Code for Sys_Timer_Start

```
// Start or restart TIMER instance
Sys_Timer_Start(TIMER);
```

Parameters

Direction	Name	Description
in	*timer*	Pointer to the timer instance

16.25.3.3 Sys_Timer_Stop

```
void Sys_Timer_Stop(TIMER_Type * timer)
```

Location: timer.h:76

Stop the timer instance.

Example Code for Sys_Timer_Stop

```
// Stop the TIMER3 instance
Sys_Timer_Stop(TIMER3);
```

Parameters

Direction	Name	Description
in	*timer*	Pointer to the timer instance

16.26 Time of Flight

Time of Flight (TOF) hardware abstraction layer.

16.26.1 Summary

Functions

- Sys_TOF_Config : Configure TOF module.
- Sys_TOF_Start : Start the time of flight module.
- Sys_TOF_Stop : Stop the time of flight module.

16.26.2 Time of Flight Function Documentation

16.26.2.1 Sys_TOF_Config

```
void Sys_TOF_Config(uint32_t cfg)
```

Location: tof.h:53

Configure TOF module.

Example Code for Sys_TOF_Config

```
// Configure time-of-flight module:
//  - Enable the error, overrun and average data complete interrupts
//  - Average data interrupt is triggered after 16 samples
Sys_TOF_Config(TOF_ERROR_INT_ENABLE     |
               TOF_OVERRUN_INT_ENABLE   |
               TOF_AVG_DATA_INT_ENABLE  |
               TOF_DATA_INT_ENABLE      |
               TOF_AVG_DATA_16);
```

Parameters

Direction	Name	Description
in	*cfg*	Time of flight configuration; use [TOF_ERROR_INT_ENABLE | TOF_ERROR_INT_DISABLE], [TOF_OVERRUN_INT_ENABLE | TOF_OVERRUN_INT_DISABLE], [TOF_AVG_DATA_INT_ENABLE | TOF_AVG_DATA_INT_DISABLE], [TOF_DATA_INT_ENABLE | TOF_DATA_INT_DISABLE], [TOF_AVG_DATA_DMA_ENABLE | TOF_AVG_DATA_DMA_DISABLE], [TOF_DATA_DMA_ENABLE | TOF_DATA_DMA_DISABLE], TOF_AVG_DATA_*, TOF_STOP_SRC_*, TOF_START_SRC_*, TOF_CLK_PRESCALE_*

16.26.2.2 Sys_TOF_Start

```
void Sys_TOF_Start()
```

Location: tof.h:63

Start the time of flight module.

Example Code for Sys_TOF_Start

```
// Start the time-of-flight counter.
Sys_TOF_Start();
```

16.26.2.3 Sys_TOF_Stop

```
void Sys_TOF_Stop()
```

Location: tof.h:73

Stop the time of flight module.

Example Code for Sys_TOF_Stop

```
// Stop the time-of-flight counter.
Sys_TOF_Stop();
```

16.27 TRIMMING SUPPORT

Power, clock, and sensor component trimming hardware abstraction layer.

16.27.1 Summary

Variables

- trim_args1 : Trim targets for items needing one trim.
- trim_args2 : Trim targets for items needing two trims.

Enumerations

- TrimTarget_t : Default trim targets present in NVR7.
- TrimName_t : Voltage rail and oscillator names.

Macros

- NULL_POINTER : NULL pointer.
- MIN_32_BIT : Minimum 32-bit value.
- MAX_32_BIT : Maximum 32-bit value.
- MIN_18_BIT : Minimum 18-bit value.
- MAX_18_BIT : Maximum 18-bit value.
- MIN_16_BIT : Minimum 16-bit value.
- MAX_16_BIT : Maximum 16-bit value.
- MIN_8_BIT : Minimum 8-bit value.
- MAX_8_BIT : Maximum 8-bit value.
- MAX_4_BIT : Maximum 4-bit value.
- ERROR_NO_ERROR
- ERROR_NULL : Null pointer error.
- ERROR_NO_TRIM_FOUND : Target trim value not found.
- ERROR_INVALID_TRIM : Trims in region specified are not valid.
- ERROR_INVALID_CRC : Trim region CRC has failed.
- ERROR_BG_INVALID : Bandgap target value is invalid.
- ERROR_BG_V_INVALID : Bandgap voltage trim is invalid.
- ERROR_BG_I_INVALID : Bandgap current trim is invalid.
- ERROR_DCDC_INVALID : DCDC trim is invalid.
- ERROR_VDDC_INVALID : VDDC trim is invalid.
- ERROR_VDDC_STBY_INVALID : VDCC standby trim is invalid.

- ERROR_VDDM_INVALID : VDDM trim is invalid.
- ERROR_VDDM_STBY_INVALID : VDCM standby trim is invalid.
- ERROR_VDDRF_INVALID : VDDRF trim is invalid.
- ERROR_VDDPA_INVALID : VDDPA trim is invalid.
- ERROR_VDDPA_MIN_INVALID : VDDPA minimum trim is invalid.
- ERROR_VDDIF_INVALID : VDDIF trim is invalid.
- ERROR_VDDFLASH_INVALID : VDDFLASH trim is invalid.
- ERROR_RCOSC_INVALID : RC start oscillator trim is invalid.
- ERROR_RCOSC32_INVALID : RC standby oscillator trim is invalid.
- ERROR_LSAD_INVALID : LSAD gain or offset is invalid.
- ERROR_TEMPERATURE_INVALID : Temperature sensor gain or offset is invalid.
- ERROR_THERMISTOR_INVALID : Thermistor gain or offset is invalid.
- ERROR_MEASURED_INVALID : Measured reference temperature is invalid.
- ERROR_TRIM_CUSTOM_SIGNATURE_INVALID : Custom signature check is invalid.
- ERROR_TRIM_CUSTOM_ICH_INVALID : Custom ICH trim value is invalid.
- ERROR_TRIM_CUSTOM_XTAL_INVALID : Custom Xtal trim value is invalid.
- TR_REG_TRIM_MASK : Temperature record 18-bit trim value mask.
- TRIM_8_BIT_TRIM_MASK : 8-bit trim value mask
- TRIM_16_BIT_TRIM_MASK : 16-bit trim value mask
- TRIM_NVR7_2_BIT_RC_FSEL_MASK
- TRIM_NVR7_VCC_DCDC_Pos
- LSAD_HF : LSAD high frequency compensation values.
- LSAD_LF : LSAD low frequency compensation values.
- LSAD_OFFSET : LSAD offset compensation address offset.
- LSAD_OFFSET_MASK : LSAD offset compensation mask.
- LSAD_GAIN : LSAD gain compensation address offset.
- LSAD_GAIN_MASK : LSAD gain compensation mask.
- TRIM : Default trim instance, pointing to NVR7.
- TRIM_SUPPLEMENTAL : Supplemental trim instance, pointing to NVR4.
- TRIM_CUSTOM_SIP1_SIGNATURE : SiP Signature for NVR6 custom trim calibration.
- TRIM_CUSTOM_CUST_SIGNATURE : Custom Signature for NVR6 custom trim calibration.
- ICH_TRIM_DEFAULT
- SYS_TRIM_LOAD_DEFAULT : Load default trim values from NVR7.
- SYS_TRIM_LOAD_SUPPLEMENTAL : Load supplemental trim values from NVR4.
- SYS_TRIM_LOAD_CUSTOM : Load custom trim values from NVR6.

Functions

- Sys_Trim_LoadTrims : Load trim values from the specified memory location.
- Sys_Trim_LoadSingleTrim : Load a trim value for a specific voltage regulator or oscillator.
- Sys_Trim_VerifyTrims : Verify if the trims memory is populated correctly.
- Sys_Trim_CheckCRC : Check if the CRC for the indicated region is valid.
- Sys_Trim_GetTrim : Get the trim value requested, check if it is valid.
- Sys_Trim_LoadBandgap : Load target trim value, if present.
- Sys_Trim_LoadDCDC : Load target trim value for current mode (LDO or BUCK).
- Sys_Trim_LoadVDDC : Load target trim value, if present.
- Sys_Trim_LoadVDDM : Load target trim value, if present.
- Sys_Trim_LoadVDDPA : Load target trim value, if present.

- Sys_Trim_LoadVDDRF : Load target trim value, if present.
- Sys_Trim_LoadCustom : Load custom trim values from NVR6.
- Sys_Trim_LoadVDDFLASH : Load target trim value, if present.
- Sys_Trim_LoadRCOSC : Load target trim value, if present.
- Sys_Trim_LoadRCOSC32 : Load target trim value, if present.
- Sys_Trim_LoadThermistor : Load target trim value, if present.
- Sys_Trim_GetLSADTrim : Load LSAD gain and offset value from specified address. Verifies valid values first.

16.27.2 Trimming Support Variable Documentation

16.27.2.1 trim_args1

```
uint32_t trim_args1[TRIM_NUM_FUNCTIONS_1_ARG]
```

Location: trim.h:280

Trim targets for items needing one trim.

16.27.2.2 trim_args2

```
uint32_t trim_args2[TRIM_NUM_FUNCTIONS_2_ARGS][2]
```

Location: trim.h:281

Trim targets for items needing two trims.

16.27.3 Trimming Support Enumeration Type Documentation

16.27.3.1 TrimTarget_t

Location: trim.h:169

Default trim targets present in NVR7.

Members

- `TARGET_BANDGAP_V = 75`

  750mV

- `TARGET_BANDGAP_I = 100`

  1000nA

- `TARGET_DCDC_1200 = 120`

  1.2V

- `TARGET_DCDC_1120 = 112`

  1.12V

- `TARGET_DCDC_1350 = 135`

  1.35V

- `TARGET_DCDC_1100 = 110`

  1.10V

- `TARGET_VDDC_1150 = 115`

  1.15V

- `TARGET_VDDC_1000 = 100`

  1.00V

- `TARGET_VDDC_1080 = 108`

  1.08V

- `TARGET_VDDC_920 = 92`

  0.92V

- `TARGET_VDDC_1050 = 105`

  1.05V

- `TARGET_VDDC_STANDBY = 80`

  0.80V

- `TARGET_VDDM_1150 = 115`

  1.15V

- `TARGET_VDDM_1080 = 108`

  1.08V

- `TARGET_VDDM_1100 = 110`

  1.05V

- `TARGET_VDDM_STANDBY = 80`

  0.80V

- `TARGET_VDDRF_1100 = 110`

  1.10V

- `TARGET_VDDRF_1070 = 107`

  1.07V

- `TARGET_VDDRF_1200 = 120`

  1.20V

- `TARGET_VDDPA_1300 = 130`

  1.30V

- `TARGET_VDDPA_1260 = 126`

  1.26V

- `TARGET_VDDPA_1600 = 160`

  1.60V

- `TARGET_VDDPA_MIN_1100 = 110`

  1.10V

- `TARGET_VDDIF_1800 = 180`

  1.80V

- `TARGET_FLASH_1600 = 160`

  1.60V

- `TARGET_RC3 = 3000`

  3MHz

- `TARGET_RC12 = 12000`

  12MHz

- `TARGET_RC24 = 24000`

  24MHz

- `TARGET_RC48 = 48000`

  48MHz

- `TARGET_RC32K = 32768`

  32kHz

- `TARGET_THERMISTOR_10 = 10`

  10uA

- `TARGET_THERMISTOR_5 = 5`

  5.0uA

16.27.3.2 TrimName_t

Location: trim.h:208

Voltage rail and oscillator names.

Members

- `TRIM_BANDGAP`
- `TRIM_DCDC`

  Select loading bandgap trim values.

- `TRIM_VDDC`

  Select loading DCDC trim values.

- `TRIM_VDDM`

  Select loading VDDC trim values.

- `TRIM_VDDRF`

  Select loading VDDM trim values.

- `TRIM_VDDPA`

  Select loading VDDRF trim values.

- `TRIM_VDDIF`

  Select loading VDDPA trim values.

- `TRIM_FLASH`

  Select loading VDDIF trim values.

- `TRIM_RCOSC`

  Select loading VDDFLASH trim values.

- `TRIM_RCOSC32`

  Select loading RC oscillator trim values.

16.27.4 Trimming Support Macro Definition Documentation

16.27.4.1 NULL_POINTER

```
#define NULL_POINTER 0
```

Location: trim.h:54

NULL pointer.

16.27.4.2 MIN_32_BIT

```
#define MIN_32_BIT 0x00000000UL
```

Location: trim.h:58

Minimum 32-bit value.

16.27.4.3 MAX_32_BIT

```
#define MAX_32_BIT 0xFFFFFFFFUL
```

Location: trim.h:61

Maximum 32-bit value.

16.27.4.4 MIN_18_BIT

```
#define MIN_18_BIT 0x00000U
```

Location: trim.h:64

Minimum 18-bit value.

16.27.4.5 MAX_18_BIT

```
#define MAX_18_BIT 0x3FFFFU
```

Location: trim.h:67

Maximum 18-bit value.

16.27.4.6 MIN_16_BIT

```
#define MIN_16_BIT 0x0000U
```

Location: trim.h:70

Minimum 16-bit value.

16.27.4.7 MAX_16_BIT

```
#define MAX_16_BIT 0xFFFFU
```

Location: trim.h:73

Maximum 16-bit value.

16.27.4.8 MIN_8_BIT

```
#define MIN_8_BIT 0x00U
```

Location: trim.h:76

Minimum 8-bit value.

16.27.4.9 MAX_8_BIT

```
#define MAX_8_BIT 0xFFU
```

Location: trim.h:79

Maximum 8-bit value.

16.27.4.10 MAX_4_BIT

```
#define MAX_4_BIT 0xFU
```

Location: trim.h:82

Maximum 4-bit value.

16.27.4.11 ERROR_NO_ERROR

```
#define ERROR_NO_ERROR 0
```

Location: trim.h:86

errors

16.27.4.12 ERROR_NULL

```
#define ERROR_NULL (1 << 1)
```

Location: trim.h:89

Null pointer error.

16.27.4.13 ERROR_NO_TRIM_FOUND

```
#define ERROR_NO_TRIM_FOUND (1 << 3)
```

Location: trim.h:92

Target trim value not found.

16.27.4.14 ERROR_INVALID_TRIM

```
#define ERROR_INVALID_TRIM (1 << 4)
```

Location: trim.h:95

Trims in region specified are not valid.

16.27.4.15 ERROR_INVALID_CRC

```
#define ERROR_INVALID_CRC (1 << 5)
```

Location: trim.h:98

Trim region CRC has failed.

16.27.4.16 ERROR_BG_INVALID

```
#define ERROR_BG_INVALID (1 << 6)
```

Location: trim.h:101

Bandgap target value is invalid.

16.27.4.17 ERROR_BG_V_INVALID

```
#define ERROR_BG_V_INVALID (1 << 7)
```

Location: trim.h:104

Bandgap voltage trim is invalid.

16.27.4.18 ERROR_BG_I_INVALID

```
#define ERROR_BG_I_INVALID (1 << 8)
```

Location: trim.h:107

Bandgap current trim is invalid.

16.27.4.19 ERROR_DCDC_INVALID

```
#define ERROR_DCDC_INVALID (1 << 9)
```

Location: trim.h:110

DCDC trim is invalid.

16.27.4.20 ERROR_VDDC_INVALID

```
#define ERROR_VDDC_INVALID (1 << 10)
```

Location: trim.h:113

VDDC trim is invalid.

16.27.4.21 ERROR_VDDC_STBY_INVALID

```
#define ERROR_VDDC_STBY_INVALID (1 << 11)
```

Location: trim.h:116

VDCC standby trim is invalid.

16.27.4.22 ERROR_VDDM_INVALID

```
#define ERROR_VDDM_INVALID (1 << 12)
```

Location: trim.h:119

VDDM trim is invalid.

16.27.4.23 ERROR_VDDM_STBY_INVALID

```
#define ERROR_VDDM_STBY_INVALID (1 << 13)
```

Location: trim.h:122

VDCM standby trim is invalid.

16.27.4.24 ERROR_VDDRF_INVALID

```
#define ERROR_VDDRF_INVALID (1 << 14)
```

Location: trim.h:125

VDDRF trim is invalid.

16.27.4.25 ERROR_VDDPA_INVALID

```
#define ERROR_VDDPA_INVALID (1 << 15)
```

Location: trim.h:128

VDDPA trim is invalid.

16.27.4.26 ERROR_VDDPA_MIN_INVALID

```
#define ERROR_VDDPA_MIN_INVALID (1 << 16)
```

Location: trim.h:131

VDDPA minimum trim is invalid.

16.27.4.27 ERROR_VDDIF_INVALID

```
#define ERROR_VDDIF_INVALID (1 << 17)
```

Location: trim.h:134

VDDIF trim is invalid.

16.27.4.28 ERROR_VDDFLASH_INVALID

```
#define ERROR_VDDFLASH_INVALID (1 << 18)
```

Location: trim.h:137

VDDFLASH trim is invalid.

16.27.4.29 ERROR_RCOSC_INVALID

```
#define ERROR_RCOSC_INVALID (1 << 19)
```

Location: trim.h:140

RC start oscillator trim is invalid.

16.27.4.30 ERROR_RCOSC32_INVALID

```
#define ERROR_RCOSC32_INVALID (1 << 20)
```

Location: trim.h:143

RC standby oscillator trim is invalid.

16.27.4.31 ERROR_LSAD_INVALID

```
#define ERROR_LSAD_INVALID (1 << 21)
```

Location: trim.h:146

LSAD gain or offset is invalid.

16.27.4.32 ERROR_TEMPERATURE_INVALID

```
#define ERROR_TEMPERATURE_INVALID (1 << 22)
```

Location: trim.h:149

Temperature sensor gain or offset is invalid.

16.27.4.33 ERROR_THERMISTOR_INVALID

```
#define ERROR_THERMISTOR_INVALID (1 << 23)
```

Location: trim.h:152

Thermistor gain or offset is invalid.

16.27.4.34 ERROR_MEASURED_INVALID

```
#define ERROR_MEASURED_INVALID (1 << 25)
```

Location: trim.h:155

Measured reference temperature is invalid.

16.27.4.35 ERROR_TRIM_CUSTOM_SIGNATURE_INVALID

```
#define ERROR_TRIM_CUSTOM_SIGNATURE_INVALID (1 << 26)
```

Location: trim.h:158

Custom signature check is invalid.

16.27.4.36 ERROR_TRIM_CUSTOM_ICH_INVALID

```
#define ERROR_TRIM_CUSTOM_ICH_INVALID (1 << 27)
```

Location: trim.h:161

Custom ICH trim value is invalid.

16.27.4.37 ERROR_TRIM_CUSTOM_XTAL_INVALID

```
#define ERROR_TRIM_CUSTOM_XTAL_INVALID (1 << 28)
```

Location: trim.h:164

Custom Xtal trim value is invalid.

16.27.4.38 TR_REG_TRIM_MASK

```
#define TR_REG_TRIM_MASK 0x3FU
```

Location: trim.h:224

Temperature record 18-bit trim value mask.

16.27.4.39 TRIM_8_BIT_TRIM_MASK

```
#define TRIM_8_BIT_TRIM_MASK 0xFFU
```

Location: trim.h:227

8-bit trim value mask

16.27.4.40 TRIM_16_BIT_TRIM_MASK

```
#define TRIM_16_BIT_TRIM_MASK 0xFFFFU
```

Location: trim.h:230

16-bit trim value mask

16.27.4.41 TRIM_NVR7_2_BIT_RC_FSEL_MASK

```
#define TRIM_NVR7_2_BIT_RC_FSEL_MASK (0x3U << 7)
```

Location: trim.h:233

16.27.4.42 TRIM_NVR7_VCC_DCDC_Pos

```
#define TRIM_NVR7_VCC_DCDC_Pos 8
```

Location: trim.h:236

16.27.4.43 LSAD_HF

```
#define LSAD_HF 0
```

Location: trim.h:240

LSAD high frequency compensation values.

16.27.4.44 LSAD_LF

```
#define LSAD_LF 1
```

Location: trim.h:243

LSAD low frequency compensation values.

16.27.4.45 LSAD_OFFSET

```
#define LSAD_OFFSET 0x00U
```

Location: trim.h:246

LSAD offset compensation address offset.

16.27.4.46 LSAD_OFFSET_MASK

```
#define LSAD_OFFSET_MASK 0xFFU
```

Location: trim.h:249

LSAD offset compensation mask.

16.27.4.47 LSAD_GAIN

```
#define LSAD_GAIN 0x04U
```

Location: trim.h:252

LSAD gain compensation address offset.

16.27.4.48 LSAD_GAIN_MASK

```
#define LSAD_GAIN_MASK 0x3FFU
```

Location: trim.h:255

LSAD gain compensation mask.

16.27.4.49 TRIM

```
#define TRIM (TRIM_Type *)TRIM_BASE_DEFAULT
```

Location: trim.h:265

Default trim instance, pointing to NVR7.

16.27.4.50 TRIM_SUPPLEMENTAL

```
#define TRIM_SUPPLEMENTAL (TRIM_Type *)FLASH0_NVR4_BASE
```

Location: trim.h:268

Supplemental trim instance, pointing to NVR4.

16.27.4.51 TRIM_CUSTOM_SIP1_SIGNATURE

```
#define TRIM_CUSTOM_SIP1_SIGNATURE 0x53495031
```

Location: trim.h:271

SiP Signature for NVR6 custom trim calibration.

16.27.4.52 TRIM_CUSTOM_CUST_SIGNATURE

```
#define TRIM_CUSTOM_CUST_SIGNATURE 0x43555354
```

Location: trim.h:274

Custom Signature for NVR6 custom trim calibration.

16.27.4.53 ICH_TRIM_DEFAULT

```
#define ICH_TRIM_DEFAULT (0x5U)
```

Location: trim.h:277

16.27.4.54 SYS_TRIM_LOAD_DEFAULT

```
#define SYS_TRIM_LOAD_DEFAULT Sys_Trim_LoadTrims(TRIM, trim_args1, trim_args2)
```

Location: trim.h:520

Load default trim values from NVR7.

Returns:

A code indicating whether an error has occurred. Error codes: errors

Example Code for SYS_TRIM_LOAD_DEFAULT

```
// Load all valid default trim values from MNVR/NVR7 for
// power rails and oscillators
result = SYS_TRIM_LOAD_DEFAULT();
```

16.27.4.55 SYS_TRIM_LOAD_SUPPLEMENTAL

```
#define SYS_TRIM_LOAD_SUPPLEMENTAL Sys_Trim_LoadTrims(TRIM_SUPPLEMENTAL,x,y)
```

Location: trim.h:536

Load supplemental trim values from NVR4.

Returns:

A code indicating whether an error has occurred. Error codes: errors

NOTE: For detail on input parameters, see trim_args1 & 2 in trim.c

Example Code for SYS_TRIM_LOAD_SUPPLEMENTAL

```
// Load the requested supplemental trim values
result = SYS_TRIM_LOAD_DEFAULT();
```

Parameters

Direction	*Name*	Description
in	*x*	uint32_t[6] containing targets in this order: DCDC, VDDRF, VDDIF, VDDFLASH, RC 3MHz, RC 32kHz
in	*y*	uint32_t[4][2] containing targets in this order: Bandgap voltage | Bandgap current VDDC voltage | VDDC standby voltage VDDM voltage | VDDM standby voltage VDDPA voltage | VDDPA minimum voltage

16.27.4.56 SYS_TRIM_LOAD_CUSTOM

```
#define SYS_TRIM_LOAD_CUSTOM Sys_Trim_LoadCustom()
```

Location: trim.h:544

Load custom trim values from NVR6.

Returns:

A code indicating whether an error has occurred. Error codes: errors

Example Code for SYS_TRIM_LOAD_CUSTOM

```
// Load all valid custom trim values from NVR6
result = SYS_TRIM_LOAD_CUSTOM()
```

16.27.5 Trimming Support Function Documentation

16.27.5.1 Sys_Trim_LoadTrims

```
uint32_t Sys_Trim_LoadTrims(TRIM_Type * trim_region, uint32_t targets_1, uint32_t
targets_2)
```

Location: trim.h:301

Load trim values from the specified memory location.

Returns:

A code indicating whether an error has occurred. Error codes: errors

NOTE: Does not work with LSAD gain/offset. Use lsadload Sys_Trim_GetLSADTrim() instead.

NOTE: For detail on input parameters, see trim_args1 & 2 in trim.c

Example Code for Sys_Trim_LoadTrims

```
// Load all valid default trim values from NVR7 for
// power rails and oscillators
result = Sys_Trim_LoadTrims(trim_region);
```

Parameters

Direction	Name	Description
in	*trim_region*	Pointer to section of memory containing trim values, typically base of NVR7.
in	*targets_1*	uint32_t[6] containing targets in this order: DCDC, VDDRF, VDDIF, VDDFLASH, RC 3MHz, RC 32kHz
in	*targets_2*	uint32_t[4][2] containing targets in this order: Bandgap voltage | Bandgap current VDDC voltage | VDDC standby voltage VDDM voltage | VDDM standby voltage VDDPA voltage | VDDPA minimum voltage

16.27.5.2 Sys_Trim_LoadSingleTrim

```
uint32_t Sys_Trim_LoadSingleTrim(uint32_t target_name, uint32_t target_value1, uint32_t
target_value2)
```

Location: trim.h:319

Load a trim value for a specific voltage regulator or oscillator.

This function attempts to load calibration values from customer trim settings in NVR4, then load manufacturing calibration values from NVR7 if customer calibration values are not found. target_name Voltage regulator or oscillator to load trim values for. target_value1 Main trim target value target_value2 Secondary trim target value used on some regulators.

Returns:

A code indicating whether an error has occurred. Error codes: errors

NOTE: Does not work with LSAD gain/offset. Use lsadload Sys_Trim_GetLSADTrim() instead.

Example Code for Sys_Trim_LoadSingleTrim

```
// Load all valid default trim values from NVR7 for
// power rails and oscillators
result = Sys_Trim_LoadTrims(trim_region);
```

Returns:

A code indicating whether an error has occurred. Error codes: errors

NOTE: Does not work with LSAD gain/offset. Use lsadload Sys_Trim_GetLSADTrim() instead.

Example Code for Sys_Trim_LoadSingleTrim

```
// Load all valid default trim values from NVR7 for
// power rails and oscillators
result = Sys_Trim_LoadTrims(trim_region);
```

Parameters

Direction	Name	Description
in	*target_name*	Voltage regulator or oscillator to load trim values for.
in	*target_value1*	Main trim target value
in	*target_value2*	Secondary trim target value used on some regulators.

16.27.5.3 Sys_Trim_VerifyTrims

```
uint32_t Sys_Trim_VerifyTrims(TRIM_Type * trim_region)
```

Location: trim.h:331

Verify if the trims memory is populated correctly.

Returns:

A code indicating whether an error has occurred. Error codes: errors

Example Code for Sys_Trim_VerifyTrims

```
// Verify that the input region contains validly prorammed values.
result = Sys_Trim_VerifyTrims(trim_region);
```

Parameters

Direction	Name	Description
in	*trim_region*	Pointer to section of memory containing trim values, typically base of NVR7.

16.27.5.4 Sys_Trim_CheckCRC

```
uint32_t Sys_Trim_CheckCRC(TRIM_Type * trim_region)
```

Location: trim.h:341

Check if the CRC for the indicated region is valid.

Returns:

A code indicating whether an error has occurred. Error codes: errors

Example Code for Sys_Trim_CheckCRC

```
// Performs a CRC on the data in the region storing the trim settings.
result = Sys_Trim_CheckCRC(trim_region);
```

Parameters

Direction	Name	Description
in	*trim_region*	Pointer to section of memory containing trim values, typically base of NVR7.

16.27.5.5 Sys_Trim_GetTrim

```
uint32_t Sys_Trim_GetTrim(uint32_t * addr, uint16_t trim_target, uint32_t record_length,
uint16_t * trim_val)
```

Location: trim.h:353

Get the trim value requested, check if it is valid.

Returns:

A code indicating whether an error has occurred. Error codes: errors

Example Code for Sys_Trim_GetTrim

```
// Retrieves a trim setting from the indicated region, if that target exists.
result = Sys_Trim_GetTrim(trim_region, TARGET_RC12, 4, &trim_voltage);
```

Parameters

Direction	Name	Description
in	*addr*	Pointer to address of base of trim record.
in	*trim_target*	Target voltage/current/clock
in	*record_length*	Number of records for that trim value.
out	*trim_val*	Pointer to return retrieved trim value.

16.27.5.6 Sys_Trim_LoadBandgap

```
uint32_t Sys_Trim_LoadBandgap(TRIM_Type * trim_values, uint32_t target_v, uint32_t
target_i)
```

Location: trim.h:368

Load target trim value, if present.

Returns:

A code indicating whether an error has occurred. Error codes: errors

Example Code for Sys_Trim_LoadBandgap

```
// Loads the bandgap trim settings from NVR7.
result = Sys_Trim_LoadBandgap(trim_region, TARGET_BANDGAP_V, TARGET_BANDGAP_I);
```

Parameters

Direction	Name	Description
in	*trim_values*	Pointer to section of memory containing trim values, typically base of NVR7.
in	*target_v*	The target voltage trim setting.
in	*target_i*	The target current trim setting.

16.27.5.7 Sys_Trim_LoadDCDC

```
uint32_t Sys_Trim_LoadDCDC(TRIM_Type * trim_values, uint32_t target)
```

Location: trim.h:381

Load target trim value for current mode (LDO or BUCK).

Returns:

A code indicating whether an error has occurred. Error codes: errors

Example Code for Sys_Trim_LoadDCDC

```
// Loads the DC-DC converter trim settings from NVR7 for 1.2 V.
result = Sys_Trim_LoadDCDC(trim_region, TARGET_DCDC_1200);
```

Parameters

Direction	Name	Description
in	*trim_values*	Pointer to section of memory containing trim values, typically base of NVR7.
in	*target*	The target voltage trim setting.

16.27.5.8 Sys_Trim_LoadVDDC

```
uint32_t Sys_Trim_LoadVDDC(TRIM_Type * trim_values, uint32_t target, uint32_t target_
standby)
```

Location: trim.h:393

Load target trim value, if present.

Returns:

A code indicating whether an error has occurred. Error codes: errors

Example Code for Sys_Trim_LoadVDDC

```
// Loads the VDDC regulator trim settings from NVR7 for 1.15 V.
result = Sys_Trim_LoadVDDC(trim_region, TARGET_VDDC_1150, TARGET_VDDC_STANDBY);
```

Parameters

Direction	Name	Description
in	*trim_values*	Pointer to section of memory containing trim values, typically base of NVR7.
in	*target*	The target voltage trim setting.
in	*target_standby*	The target standby voltage trim setting.

16.27.5.9 Sys_Trim_LoadVDDM

```
uint32_t Sys_Trim_LoadVDDM(TRIM_Type * trim_values, uint32_t target, uint32_t target_
standby)
```

Location: trim.h:406

Load target trim value, if present.

Returns:

A code indicating whether an error has occurred. Error codes: errors

Example Code for Sys_Trim_LoadVDDM

```
// Loads the VDDM regulator trim settings from NVR7 for 1.15 V.
result = Sys_Trim_LoadVDDM(trim_region, TARGET_VDDM_1150, TARGET_VDDM_STANDBY);
```

Parameters

Direction	Name	Description
in	*trim_values*	Pointer to section of memory containing trim values, typically base of NVR7.
in	*target*	The target voltage trim setting.
in	*target_standby*	The target standby voltage trim setting.

16.27.5.10 Sys_Trim_LoadVDDPA

```
uint32_t Sys_Trim_LoadVDDPA(TRIM_Type * trim_values, uint32_t target)
```

Location: trim.h:418

Load target trim value, if present.

Returns:

A code indicating whether an error has occurred. Error codes: errors

Example Code for Sys_Trim_LoadVDDPA

```
// Loads the VDDPA regulator trim settings from NVR7 for 1.6 V.
result = Sys_Trim_LoadVDDPA(trim_region, TARGET_VDDPA_1600, TARGET_VDDPA_MIN_1100);
```

Parameters

Direction	Name	Description
in	*trim_values*	Pointer to section of memory containing trim values, typically base of NVR7.
in	*target*	The target voltage trim setting.

16.27.5.11 Sys_Trim_LoadVDDRF

```
uint32_t Sys_Trim_LoadVDDRF(TRIM_Type * trim_values, uint32_t target)
```

Location: trim.h:429

Load target trim value, if present.

Returns:

A code indicating whether an error has occurred. Error codes: errors

Example Code for Sys_Trim_LoadVDDRF

```
// Loads the VDDRF regulator trim settings from NVR7 for 1.1 V.
result = Sys_Trim_LoadVDDRF(trim_region, TARGET_VDDRF_1100);
```

Parameters

Direction	Name	Description
in	*trim_values*	Pointer to section of memory containing trim values, typically base of NVR7.
in	*target*	The target voltage trim setting.

16.27.5.12 Sys_Trim_LoadCustom

```
uint32_t Sys_Trim_LoadCustom()
```

Location: trim.h:437

Load custom trim values from NVR6.

Returns:

A code indicating whether an error has occurred. Error codes: errors

Example Code for Sys_Trim_LoadCustom

```
// Load all valid custom trim values from NVR6
result = SYS_TRIM_LOAD_CUSTOM()
```

16.27.5.13 Sys_Trim_LoadVDDFLASH

```
uint32_t Sys_Trim_LoadVDDFLASH(TRIM_Type * trim_values, uint32_t target)
```

Location: trim.h:465

Load target trim value, if present.

Returns:

A code indicating whether an error has occurred. Error codes: errors

Example Code for Sys_Trim_LoadVDDFLASH

```
// Loads the VDDFLASH regulator trim settings from NVR7 for 1.6 V.
result = Sys_Trim_LoadVDDFLASH(trim_region, TARGET_VDDFLASH_1600);
```

Parameters

Direction	Name	Description
in	*trim_values*	Pointer to section of memory containing trim values, typically base of NVR7.
in	*target*	The voltage trim setting desired.

16.27.5.14 Sys_Trim_LoadRCOSC

```
uint32_t Sys_Trim_LoadRCOSC(TRIM_Type * trim_values, uint32_t target)
```

Location: trim.h:476

Load target trim value, if present.

Returns:

A code indicating whether an error has occurred. Error codes: errors

Example Code for Sys_Trim_LoadRCOSC

```
// Loads the RC Start oscillator trim settings from NVR7 for 12 MHz.
result = Sys_Trim_LoadRCOSC(trim_region, TARGET_RC12);
```

Parameters

Direction	Name	Description
in	*trim_values*	Pointer to section of memory containing trim values, typically base of NVR7.
in	*target*	The target clock trim setting.

16.27.5.15 Sys_Trim_LoadRCOSC32

```
uint32_t Sys_Trim_LoadRCOSC32(TRIM_Type * trim_values, uint32_t target)
```

Location: trim.h:487

Load target trim value, if present.

Returns:

A code indicating whether an error has occurred. Error codes: errors

Example Code for Sys_Trim_LoadRCOSC32

```
// Loads the RC 32768Hz oscillator trim settings from NVR7 for 32768 Hz.
result = Sys_Trim_LoadRCOSC32(trim_region, TARGET_RC32K);
```

Parameters

Direction	Name	Description
in	*trim_values*	Pointer to section of memory containing trim values, typically base of NVR7.
in	*target*	The target clock trim setting.

16.27.5.16 Sys_Trim_LoadThermistor

```
uint32_t Sys_Trim_LoadThermistor(TRIM_Type * trim_values, uint16_t target)
```

Location: trim.h:498

Load target trim value, if present.

Returns:

A code indicating whether an error has occurred. Error codes: errors

Example Code for Sys_Trim_LoadThermistor

```
// Loads the trim settings for the thermistor current source.
result = Sys_Trim_LoadThermistor(trim_region, TARGET_THERMISTOR_10);
```

Parameters

Direction	Name	Description
in	*trim_values*	Pointer to section of memory containing trim values, typically base of NVR7.
in	*target*	The target thermistor trim setting.

16.27.5.17 Sys_Trim_GetLSADTrim

```
uint32_t Sys_Trim_GetLSADTrim(uint32_t * addr, uint32_t * gain, uint32_t * offset)
```

Location: trim.h:512

Load LSAD gain and offset value from specified address. Verifies valid values first.

lsadload

Returns:

A code indicating whether an error has occurred. Error codes: errors

NOTE: Assumes format of LSAD gain and offset storage.

Example Code for Sys_Trim_GetLSADTrim

```
// Loads gain and offset values for the LSAD module from NVR7.
result = Sys_Trim_GetLSADTrim(trim_region, &gain, &offset);
```

Parameters

Direction	Name	Description
in	*addr*	Pointer to memory containing offset in the first word, and gain in the second word.
out	*gain*	Pointer to return gain value.
out	*offset*	Pointer to return offset value.

16.28 UART

Universal Asynchronous Receiver/Transmitter (UART) hardware abstraction layer.

16.28.1 Summary

Macros

- UART_PADS_NUM : The number of input GPIO pad configurations for a UART interface (1 per instance)
- SYS_UART_GPIOCONFIG : Macro wrapper for Sys_UART_GPIOConfig().
- SYS_UART_CONFIG : Macro wrapper for Sys_UART_Config().

Functions

- Sys_UART_GPIOConfig : Configure two GPIOs for the specified UART interface.
- Sys_UART_Config : Configure and enable a UART interface.

16.28.2 UART Macro Definition Documentation

16.28.2.1 UART_PADS_NUM

```
#define UART_PADS_NUM 1
```

Location: uart.h:41

The number of input GPIO pad configurations for a UART interface (1 per instance)

16.28.2.2 SYS_UART_GPIOCONFIG

```
#define SYS_UART_GPIOCONFIG Sys_UART_GPIOConfig(UART, (cfg), (pad_tx), (pad_rx))
```

Location: uart.h:91

Macro wrapper for Sys_UART_GPIOConfig().

Configure two GPIOs for the specified UART interface. cfg GPIO pin configuration for the UART pads pad_tx GPIO to use as the UART transmit pad pad_rx GPIO to use as the UART receive pad

Example Code for SYS_UART_GPIOCONFIG

```
// Configure GPIOs 5 and 6 for the default UART interface with
// low-pass filter disabled, 8X drive-strength, and 1 kOhm pull-up resistors
SYS_UART_GPIOCONFIG((GPIO_LPF_DISABLE | GPIO_1K_PULL_UP  |
                    GPIO_6X_DRIVE), GPIO5, GPIO6);
```

Example Code for SYS_UART_GPIOCONFIG

```
// Configure GPIOs 5 and 6 for the default UART interface with
// low-pass filter disabled, 8X drive-strength, and 1 kOhm pull-up resistors
SYS_UART_GPIOCONFIG((GPIO_LPF_DISABLE | GPIO_1K_PULL_UP  |
                    GPIO_6X_DRIVE), GPIO5, GPIO6);
```


Parameters

Direction	*Name*	Description
in	*cfg*	GPIO pin configuration for the UART pads
in	*pad_tx*	GPIO to use as the UART transmit pad
in	*pad_rx*	GPIO to use as the UART receive pad

16.28.2.3 SYS_UART_CONFIG

```
#define SYS_UART_CONFIG Sys_UART_Config(UART, (uart_clk_hz), (baud), (config))
```

Location: uart.h:107

Macro wrapper for Sys_UART_Config().

Configure and enable a UART interface. uart_clk_hz UART clock speed in hertz baud Baud rate to which UART* is configured config DMA and interrupt mode enable; use UART_TX_DMA_[ENABLE | DISABLE] UART_RX_DMA_[ENABLE | DISABLE] UART_TX_INT_[ENABLE | DISABLE] UART_RX_INT_[ENABLE | DISABLE] UART_OVERRUN_INT_[ENABLE | DISABLE]

Example Code for SYS_UART_CONFIG

```
// Enable and Configure the default UART:
//  - 8 MHz clock speed
//  - 9600 Hz baud rate
//  - A TX DMA request is generated when new data is
//    requested by the UART interface
//  - An RX DMA request is generated when new data is
//    received by the UART interface
//  - Interrupts enabled
SYS_UART_CONFIG(8000000, 9600, (UART_TX_DMA_ENABLE |
                UART_RX_DMA_ENABLE | UART_TX_START_INT_ENABLE |
                UART_RX_INT_ENABLE | UART_OVERRUN_INT_ENABLE));
```

Example Code for SYS_UART_CONFIG

```
// Enable and Configure the default UART:
//  - 8 MHz clock speed
//  - 9600 Hz baud rate
//  - A TX DMA request is generated when new data is
//    requested by the UART interface
//  - An RX DMA request is generated when new data is
//    received by the UART interface
//  - Interrupts enabled
SYS_UART_CONFIG(8000000, 9600, (UART_TX_DMA_ENABLE |
                UART_RX_DMA_ENABLE | UART_TX_START_INT_ENABLE |
                UART_RX_INT_ENABLE | UART_OVERRUN_INT_ENABLE));
```


Parameters

Direction	*Name*	Description
in	*uart_clk_hz*	UART clock speed in hertz
in	*baud*	Baud rate to which UART* is configured
in	*config*	DMA and interrupt mode enable; use UART_TX_DMA_[ENABLE | DISABLE] UART_RX_DMA_[ENABLE | DISABLE] UART_TX_INT_[ENABLE | DISABLE] UART_RX_INT_[ENABLE | DISABLE] UART_OVERRUN_INT_[ENABLE | DISABLE]

16.28.3 UART Function Documentation

16.28.3.1 Sys_UART_GPIOConfig

```
void Sys_UART_GPIOConfig(const UART_Type * uart, uint32_t cfg, uint32_t pad_tx, uint32_t pad_rx)
```

Location: uart.h:52

Configure two GPIOs for the specified UART interface.

Example Code for Sys_UART_GPIOConfig

```
// Configure GPIOs 5 and 6 for the UART interface with
// low-pass filter disabled, 8X drive-strength, and 1 kOhm pull-up resistors
Sys_UART_GPIOConfig(UART, (GPIO_LPF_DISABLE | GPIO_1K_PULL_UP |
                    GPIO_8X_DRIVE), GPIO5, GPIO6);
```

Parameters

Direction	Name	Description
in	*uart*	Pointer to the UART instance
in	*cfg*	GPIO pin configuration for the UART pads
in	*pad_tx*	GPIO to use as the UART transmit pad
in	*pad_rx*	GPIO to use as the UART receive pad

16.28.3.2 Sys_UART_Config

```
void Sys_UART_Config(UART_Type * uart, uint32_t uart_clk_hz, uint32_t baud, uint32_t config)
```

Location: uart.h:80

Configure and enable a UART interface.

Example Code for Sys_UART_Config

```
// Enable and Configure a UART:
//  - 8 MHz clock speed
//  - 9600 Hz baud rate
//  - A TX DMA request is generated when new data is
//    requested by the UART interface
//  - An RX DMA request is generated when new data is
//    received by the UART interface
//  - Interrupts enabled
Sys_UART_Config(UART, 8000000, 9600, (UART_TX_DMA_ENABLE |
                UART_RX_DMA_ENABLE | UART_TX_START_INT_ENABLE |
                UART_RX_INT_ENABLE | UART_OVERRUN_INT_ENABLE));
```

Parameters

Direction	Name	Description
in	*uart*	Pointer to the UART instance
in	*uart_clk_hz*	UART clock speed in hertz
in	*baud*	Baud rate to which UART* is configured
in	*config*	DMA and interrupt mode enable; use UART_TX_DMA_[ENABLE | DISABLE] UART_RX_DMA_[ENABLE | DISABLE] UART_TX_INT_[ENABLE | DISABLE] UART_RX_INT_[ENABLE | DISABLE] UART_OVERRUN_INT_[ENABLE | DISABLE]

16.29 WATCHDOG

Watchdog timer hardware abstraction layer.

16.29.1 Summary

Macros

- SYS_WATCHDOG_REFRESH : Refresh the chip and software watchdog timers.

16.29.2 Watchdog Macro Definition Documentation

16.29.2.1 SYS_WATCHDOG_REFRESH

```
#define SYS_WATCHDOG_REFRESH WATCHDOG->CTRL = WATCHDOG_REFRESH
```

Location: watchdog.h:40

Refresh the chip and software watchdog timers.

CHAPTER 17

Flash Library Reference

Flash Library Reference.

17.1 Summary

Variables

- FlashLib_Version : Firmware revision code variable.

Macros

- FLASH_FW_VER_MAJOR : Flash library major version number.
- FLASH_FW_VER_MINOR : Flash library minor version number.
- FLASH_FW_VER_REVISION : Flash library revision version number.
- FLASH_FW_VER : Flash library version number, concatenation of all version numbers.
- FLASH0 : Define FLASH0 as the first flash instance, if this is not defined in the headers.
- FLASH_INSTANCE_NUM : Total number of flash instances.
- FLASH_0_DESCR_NUM : Total number of descriptor types in flash 0 (both RSL15 variants).
- RSL15_284_FLASH_SET : Define the set of memories enabled for the RSL15-284 configuration.

Functions

- Flash_Initialize : Initialize clock and access to flash.
- Flash_WriteWord : Write a word to a flash address.
- Flash_WriteBuffer : Write contents of a static memory buffer to flash.
- Flash_WriteDouble : Write a 38-bit word to flash.
- Flash_ReadWord : Read a 32-bit word from flash.
- Flash_ReadBuffer : Read contents of flash into a static memory buffer.
- Flash_ReadDouble : Read a 38-bit word from flash.
- Flash_EraseFlashBank : Erase a single flash bank.
- Flash_EraseChip : Erase all data and code flash.
- Flash_EraseSector : Erase a sector flash.
- Flash_BlankCheck : Check if flash region is blank.

17.2 Detailed Description

This reference chapter presents a detailed description of all the functions in the flash programming and erase support library. This reference includes calling parameters, returned values, and assumptions.

Warning: All functions provided by the flash library should be executed from RAM or ROM, as executing them from flash can result in hidden, flash-access-related failures.

17.3 Flash Library Reference Variable Documentation

17.3.1 FlashLib_Version

```
const short FlashLib_Version
```

Location: flash.h:59

Firmware revision code variable.

Access to this variable is available through the ROM tables.

17.4 Flash Library Reference Macro Definition Documentation

17.4.1 FLASH_FW_VER_MAJOR

```
#define FLASH_FW_VER_MAJOR 0x03
```

Location: flash.h:44

Flash library major version number.

17.4.2 FLASH_FW_VER_MINOR

```
#define FLASH_FW_VER_MINOR 0x00
```

Location: flash.h:47

Flash library minor version number.

17.4.3 FLASH_FW_VER_REVISION

```
#define FLASH_FW_VER_REVISION 0x02
```

Location: flash.h:50

Flash library revision version number.

17.4.4 FLASH_FW_VER

```
#define FLASH_FW_VER ((FLASH_FW_VER_MAJOR << 12) | \
                                          (FLASH_FW_VER_MINOR << 8)  | \
                                          FLASH_FW_VER_REVISION)
```

Location: flash.h:53

Flash library version number, concatenation of all version numbers.

17.4.5 FLASH0

```
#define FLASH0 ((FLASH_Type *)FLASH_BASE)
```

Location: flash.h:63

Define FLASH0 as the first flash instance, if this is not defined in the headers.

17.4.6 FLASH_INSTANCE_NUM

```
#define FLASH_INSTANCE_NUM 0x1U
```

Location: flash_rsl15.h:30

Total number of flash instances.

17.4.7 FLASH_0_DESCR_NUM

```
#define FLASH_0_DESCR_NUM 0x3U
```

Location: flash_rsl15.h:33

Total number of descriptor types in flash 0 (both RSL15 variants).

17.4.8 RSL15_284_FLASH_SET

```
#define RSL15_284_FLASH_SET 0x17
```

Location: flash_rsl15.h:36

Define the set of memories enabled for the RSL15-284 configuration.

17.5 Flash Library Reference Function Documentation

17.5.1 Flash_Initialize

```
FlashStatus_t Flash_Initialize(unsigned int num, FlashClockFrequency_t freq)
```

Location: flash.h:85

Initialize clock and access to flash.

This function powers-up and enables access to a flash region. It also applies the correct delay settings based on the specified flash clock frequency (freq). num Flash instance to be initialized freq Flash clock frequency in Hertz, only defined frequencies supported

See: FlashClockFrequency_t

Returns:

Flash API status code

See: FlashStatus_t

NOTE: System clock frequency should not be changed while the flash is being erased or programmed. An accurate system clock frequency of 1 MHz or higher is required for proper flash operation. If using the RC oscillator, care must be taken as the trimmed frequency for this oscillator has a high temperature dependency.

See: FlashClockFrequency_t

Returns:

Flash API status code

See: FlashStatus_t

NOTE: System clock frequency should not be changed while the flash is being erased or programmed. An accurate system clock frequency of 1 MHz or higher is required for proper flash operation. If using the RC oscillator, care must be taken as the trimmed frequency for this oscillator has a high temperature dependency.

Parameters

Direction	Name	Description
`in`	*num*	Flash instance to be initialized
`in`	*freq*	Flash clock frequency in Hertz, only defined frequencies supported

17.5.2 Flash_WriteWord

```
FlashStatus_t Flash_WriteWord(uint32_t addr, uint32_t word, bool enb_endurance)
```

Location: flash.h:105

Write a word to a flash address.

This function writes a single word to flash. addr Address of the word to be written. word Data to be written to flash. enb_endurance Set to 0 for default flash endurance;
Set to 1 to enable two-stage programming for higher endurance of the data programmed.
Returns:
Flash API status code

See: FlashStatus_t

NOTE: addr must be word aligned.
Contents of flash must be erased prior to performing a write.
Interrupts are not disabled; operation is undefined if the calling application modifies the values of flash registers before returning from this function call.

Returns:

Flash API status code

See: FlashStatus_t

NOTE: addr must be word aligned.
Contents of flash must be erased prior to performing a write.
Interrupts are not disabled; operation is undefined if the calling application modifies the values of flash registers before returning from this function call.

Parameters

Direction	Name	Description
in	*addr*	Address of the word to be written.
in	*word*	Data to be written to flash.
in	*enb_endurance*	Set to 0 for default flash endurance; Set to 1 to enable two-stage programming for higher endurance of the data programmed.

17.5.3 Flash_WriteBuffer

```
FlashStatus_t Flash_WriteBuffer(uint32_t addr, uint32_t word_length, const uint32_t *
words, bool enb_endurance)
```

Location: flash.h:137

Write contents of a static memory buffer to flash.

This function writes the contents of a static memory buffer to flash. A read-back verification is performed after write to ensure the write has been successful. addr Address of first word location in flash. word_length Total number of words to be written to flash. words A 32-bit C pointer to the memory location of
the buffer to be written to flash. enb_endurance Set to 0 for default flash endurance;
Set to 1 to enable two-stage programming for higher endurance of data programmed.
Returns:
Flash API status code

See: FlashStatus_t

NOTE: addr must be word aligned.
Contents of flash must be erased prior to performing a write.
Interrupts are disabled during critical sections, to ensure proper flash operation.
Applications must ensure that the function completes and that the return value is FLASH_ERR_NONE to consider the two-stage programming to be complete.
Source address of data being read and destination address being written, can not be part the same

flash instance.
CRC peripheral registers are modified during execution, and restored before returning. The CRC must not be used by the application while writing the buffer to flash.

Returns:

Flash API status code

See: FlashStatus_t

NOTE: addr must be word aligned.
Contents of flash must be erased prior to performing a write.
Interrupts are disabled during critical sections, to ensure proper flash operation.
Applications must ensure that the function completes and that the return value is FLASH_ERR_NONE to consider the two-stage programming to be complete.
Source address of data being read and destination address being written, can not be part the same flash instance.
CRC peripheral registers are modified during execution, and restored before returning. The CRC must not be used by the application while writing the buffer to flash.

Parameters

Direction	Name	Description
in	*addr*	Address of first word location in flash.
in	*word_length*	Total number of words to be written to flash.
in	*words*	A 32-bit C pointer to the memory location of the buffer to be written to flash.
in	*enb_endurance*	Set to 0 for default flash endurance; Set to 1 to enable two-stage programming for higher endurance of data programmed.

17.5.4 Flash_WriteDouble

```
FlashStatus_t Flash_WriteDouble(uint32_t addr, const uint32_t * word, bool enb_endurance)
```

Location: flash.h:164

Write a 38-bit word to flash.

This function temporarily disables automatic flash ECC generation, allowing the user to write 38-bits to a single word address in flash.
A read-back verification is performed after write to ensure the write has been successful. addr Address of the word to be written in flash. word 32-bit C pointer to the word and ECC data to be written to flash:

- word[0] contains the data to be written to flash
- word[1] [5:0] contains the 6-bit data to be written as the ECC value.

enb_endurance Set to 0 for default flash endurance;
Set to 1 to enable two-stage programming for higher endurance of data programmed.
Returns:

Flash API status code

See: FlashStatus_t

NOTE: addr must be word aligned.
Interrupts are not disabled; operation is undefined if the calling application modifies the values of flash registers before returning from this function call.

A read-back verification is performed after write to ensure the write has been successful.
Returns:

Flash API status code

See: FlashStatus_t

NOTE: addr must be word aligned.
Interrupts are not disabled; operation is undefined if the calling application modifies the values of flash registers before returning from this function call.

Parameters

Direction	Name	Description
`in`	*addr*	Address of the word to be written in flash.
`in`	*word*	32-bit C pointer to the word and ECC data to be written to flash: • word[0] contains the data to be written to flash • word[1] [5:0] contains the 6-bit data to be written as the ECC value.
`in`	*enb_endurance*	Set to 0 for default flash endurance; Set to 1 to enable two-stage programming for higher endurance of data programmed.

17.5.5 Flash_ReadWord

```
FlashStatus_t Flash_ReadWord(uint32_t addr, uint32_t * word)
```

Location: flash.h:181

Read a 32-bit word from flash.

This function reads a 32-bit word from flash. If ECC is enabled (default), hardware will log/generate interrupt on ECC errors. addr Address in flash to be read. word 32-bit C pointer to the word read from flash.

Returns:

Flash API status code

See: FlashStatus_t

NOTE: addr must be word aligned.
Interrupts are not disabled; operation is undefined if the calling application modifies the values of flash registers before returning from this function call.

Returns:

Flash API status code

See: FlashStatus_t

NOTE: addr must be word aligned.
Interrupts are not disabled; operation is undefined if the calling application modifies the values of flash registers before returning from this function call.

Parameters

Direction	Name	Description
in	*addr*	Address in flash to be read.
out	*word*	32-bit C pointer to the word read from flash.

17.5.6 Flash_ReadBuffer

```
FlashStatus_t Flash_ReadBuffer(uint32_t flash_addr, uint32_t dram_addr, unsigned int
word_length)
```

Location: flash.h:202

Read contents of flash into a static memory buffer.

This function uses the flash copier to read contents of flash into a memory buffer. flash_addr Address of first word location in flash. dram_addr Address of first word location in static memory. word_length Total number of words to be read from flash.

Returns:

Flash API status code

See: FlashStatus_t

NOTE: flash_addr and dram_addr must be word aligned.
Interrupts are not disabled; operation is undefined if the calling application modifies the values of flash registers before returning from this function call.
This function fails if the DMA or CryptoCell continuously blocks memory accesses by the flash copier by accessing memory on every cycle.

Returns:

Flash API status code

See: FlashStatus_t

NOTE: flash_addr and dram_addr must be word aligned.
Interrupts are not disabled; operation is undefined if the calling application modifies the values of flash registers before returning from this function call.
This function fails if the DMA or CryptoCell continuously blocks memory accesses by the flash copier by accessing memory on every cycle.

Parameters

Direction	Name	Description
in	*flash_addr*	Address of first word location in flash.
in	*dram_addr*	Address of first word location in static memory.
in	*word_length*	Total number of words to be read from flash.

17.5.7 Flash_ReadDouble

```
FlashStatus_t Flash_ReadDouble(uint32_t addr, uint32_t * word)
```

Location: flash.h:227

Read a 38-bit word from flash.

This function temporarily disables automatic flash ECC generation, allowing the user to read all 38 bits from a single word address in flash.

NOTE: ECC checks are not performed on the 32-bit data word or 6-bit ECC value.

addr Address of the word to be read from flash. word 32-bit C pointer to the word and ECC data read from flash:

- word[0] contains the data to word read from flash.
- word[1] [5:0] contains the 6-bit data ECC value read.

Returns:

Flash API status code

See: FlashStatus_t

NOTE: addr must be word aligned.
Interrupts are not disabled; operation is undefined if the calling application modifies the values of flash registers before returning from this function call.

NOTE: ECC checks are not performed on the 32-bit data word or 6-bit ECC value.

Returns:

Flash API status code

See: FlashStatus_t

NOTE: addr must be word aligned.
Interrupts are not disabled; operation is undefined if the calling application modifies the values of flash registers before returning from this function call.

Parameters

Direction	Name	Description
`in`	*addr*	Address of the word to be read from flash.
`out`	*word*	32-bit C pointer to the word and ECC data read from flash: • word[0] contains the data to word read from flash. • word[1] [5:0] contains the 6-bit data ECC value read.

17.5.8 Flash_EraseFlashBank

```
FlashStatus_t Flash_EraseFlashBank(uint32_t num)
```

Location: flash.h:245

Erase a single flash bank.

This function erases all code and data regions of a flash instance. num Flash instance not to be erased.
Returns:
Flash API status code

See: FlashStatus_t

NOTE: A blank check is not performed to ensure that the flash has been successfully erased. Flash_BlankCheck can be used by an application to verify if the erase has been successful.
NVR regions are not erased. Interrupts are not disabled; operation is undefined if the calling application modifies the values of flash registers before returning from this function call.

Returns:

Flash API status code

See: FlashStatus_t

NOTE: A blank check is not performed to ensure that the flash has been successfully erased. Flash_BlankCheck can be used by an application to verify if the erase has been successful.
NVR regions are not erased. Interrupts are not disabled; operation is undefined if the calling application modifies the values of flash registers before returning from this function call.

Parameters

Direction	Name	Description
in	*num*	Flash instance not to be erased.

17.5.9 Flash_EraseChip

```
FlashStatus_t Flash_EraseChip()
```

Location: flash.h:262

Erase all data and code flash.

This function erases all code and data regions of all flash instances.
Returns:
Flash API status code

See: FlashStatus_t

NOTE: A blank check is not performed to ensure that the flash has been successfully erased. Flash_BlankCheck can be used by an application to verify if the erase has been successful.
NVR regions are not erased.
Interrupts are not disabled; operation is undefined if the calling application modifies the values of flash registers before returning from this function call.

Returns:

Flash API status code

See: FlashStatus_t

NOTE: A blank check is not performed to ensure that the flash has been successfully erased. Flash_BlankCheck can be used by an application to verify if the erase has been successful.
NVR regions are not erased.
Interrupts are not disabled; operation is undefined if the calling application modifies the values of flash registers before returning from this function call.

17.5.10 Flash_EraseSector

```
FlashStatus_t Flash_EraseSector(uint32_t addr, bool enb_endurance)
```

Location: flash.h:278

Erase a sector flash.

This function erases a flash sector (512 words for code, 64 words for data). addr An address within the flash sector to be erased. enb_endurance Set to 0 for default flash endurance;
Set to 1 to enable two-stage erase iteration for higher endurance of flash.
Returns:
Flash API status code. See FlashStatus_t

NOTE: Interrupts are not disabled; operation is undefined if the calling application modifies the values of flash registers before returning from this function call.

Returns:

Flash API status code. See FlashStatus_t

NOTE: Interrupts are not disabled; operation is undefined if the calling application modifies the values of flash registers before returning from this function call.

Parameters

Direction	Name	Description
`in`	*addr*	An address within the flash sector to be erased.
`in`	*enb_endurance*	Set to 0 for default flash endurance; Set to 1 to enable two-stage erase iteration for higher endurance of flash.

17.5.11 Flash_BlankCheck

```
FlashStatus_t Flash_BlankCheck(uint32_t addr, unsigned int word_length)
```

Location: flash.h:294

Check if flash region is blank.

This function uses the flash copier in comparator mode to verify if the flash contents are empty (i.e containing the erase value 0xFFFFFFFF). addr Address of the first word in flash to be verified. word_length Total number of words to be verified.
Returns:

Flash API status code FlashStatus_t

NOTE: Interrupts are not disabled; operation is undefined if the calling application modifies the values of flash registers before returning from this function call.
addr must be word aligned.

Returns:

Flash API status code FlashStatus_t

NOTE: Interrupts are not disabled; operation is undefined if the calling application modifies the values of flash registers before returning from this function call.
addr must be word aligned.

Parameters

Direction	Name	Description
`in`	*addr*	Address of the first word in flash to be verified.
`in`	*word_length*	Total number of words to be verified.

CHAPTER 18

CMSIS Drivers Reference

CMSIS Drivers Reference.

18.1 Summary

Typedefs

- ARM_DRIVER_VERSION : Driver Version.
- ARM_POWER_STATE : General power states.

Data Structures

- _ARM_DRIVER_VERSION : Driver Version.

Enumerations

- _ARM_POWER_STATE : General power states.

Macros

- ARM_DRIVER_VERSION_MAJOR_MINOR : Driver API Version.
- ARM_DRIVER_OK : General return codes
- ARM_DRIVER_ERROR : Unspecified error.
- ARM_DRIVER_ERROR_BUSY : Driver is busy.
- ARM_DRIVER_ERROR_TIMEOUT : Timeout occurred.
- ARM_DRIVER_ERROR_UNSUPPORTED : Operation not supported.
- ARM_DRIVER_ERROR_PARAMETER : Parameter error.
- ARM_DRIVER_ERROR_SPECIFIC : Start of driver specific errors.

18.2 CMSIS Drivers Reference Typedef Documentation

18.2.1 ARM_DRIVER_VERSION

```
typedef struct _ARM_DRIVER_VERSION ARM_DRIVER_VERSION
```

Location: Driver_Common.h:62

Driver Version.

18.2.2 ARM_POWER_STATE

```
typedef enum _ARM_POWER_STATE ARM_POWER_STATE
```

Location: Driver_Common.h:80

General power states.

18.3 CMSIS Drivers Reference Data Structures Type Documentation

18.3.1 _ARM_DRIVER_VERSION

Location: Driver_Common.h:59

Driver Version.

Data Fields

Type	*Name*	Description
uint16_t	*api*	API version.
uint16_t	*drv*	Driver version.

18.4 CMSIS Drivers Reference Enumeration Type Documentation

18.4.1 _ARM_POWER_STATE

Location: Driver_Common.h:76

General power states.

Members

- `ARM_POWER_OFF`

  Power off: No operation possible.

- `ARM_POWER_LOW`

  Low Power mode: Retain state, detect and signal wake-up events.

- `ARM_POWER_FULL`

  Power on: Full operation at maximum performance.

18.5 CMSIS Drivers Reference Macro Definition Documentation

18.5.1 ARM_DRIVER_VERSION_MAJOR_MINOR

```
#define ARM_DRIVER_VERSION_MAJOR_MINOR (((major) << 8) | (minor))
```

Location: Driver_Common.h:54

Driver API Version.

18.5.2 ARM_DRIVER_OK

```
#define ARM_DRIVER_OK 0
```

Location: Driver_Common.h:65

General return codes

Operation succeeded

18.5.3 ARM_DRIVER_ERROR

```
#define ARM_DRIVER_ERROR -1
```

Location: Driver_Common.h:66

Unspecified error.

18.5.4 ARM_DRIVER_ERROR_BUSY

```
#define ARM_DRIVER_ERROR_BUSY -2
```

Location: Driver_Common.h:67

Driver is busy.

18.5.5 ARM_DRIVER_ERROR_TIMEOUT

```
#define ARM_DRIVER_ERROR_TIMEOUT -3
```

Location: Driver_Common.h:68

Timeout occurred.

18.5.6 ARM_DRIVER_ERROR_UNSUPPORTED

```
#define ARM_DRIVER_ERROR_UNSUPPORTED -4
```

Location: Driver_Common.h:69

Operation not supported.

18.5.7 ARM_DRIVER_ERROR_PARAMETER

```
#define ARM_DRIVER_ERROR_PARAMETER -5
```

Location: Driver_Common.h:70

Parameter error.

18.5.8 ARM_DRIVER_ERROR_SPECIFIC

```
#define ARM_DRIVER_ERROR_SPECIFIC -6
```

Location: Driver_Common.h:71

Start of driver specific errors.

18.6 CMSIS DMA Driver

CMSIS DMA Driver Reference.

18.6.1 Summary

Typedefs

- DMA_SEL_t : Selects the DMA channel.
- DMA_TRG_t : Selects the DMA src/dst target interface.
- DMA_SRC_STEP_t : Selects the step size increment to the DMA channel source address.
- DMA_DST_STEP_t : Selects the step size increment to the DMA channel destination address.
- DMA_SRC_DST_TRANS_LENGHT_SEL_t : Selects whether the transfer length counter depends on either the source word counts or the destination word count.
- DMA_DATA_MODE_t : Selects how often data is transferred.
- DMA_BYTE_ORDER_t : Selects the order of the data bytes.
- DMA_WORD_SIZE_t : Selects the src/dst data word size.
- DMA_CH_PRI_t : Selects priority of DMA channels.
- ADC_EVENT_SRC_t : Selects DMA interrupt channel source.
- DMA_SignalEvent_t : Pointer to DMA_SignalEvent : Signal Timer event.
- DMA_CFG_t : DMA channel configuration.
- DMA_ADDR_CFG_t : DMA src/dst address configuration.
- DMA_PRI_CFG_t : DMA interrupt priority configuration.
- DMA_STATUS_t : DMA status.
- DRIVER_DMA_t : Access structure of the DMA Driver.

Data Structures

- _DMA_CFG_t : DMA channel configuration.
- _DMA_ADDR_CFG_t : DMA src/dst address configuration.
- _DMA_PRI_CFG_t : DMA interrupt priority configuration.
- _DMA_STATUS_t : DMA status.
- _DRIVER_DMA_t : Access structure of the DMA Driver.

Enumerations

- _DMA_SEL_t : Selects the DMA channel.
- _DMA_TRG_t : Selects the DMA src/dst target interface.
- _DMA_SRC_STEP_t : Selects the step size increment to the DMA channel source address.
- _DMA_DST_STEP_t : Selects the step size increment to the DMA channel destination address.
- _DMA_SRC_DST_TRANS_LENGHT_SEL_t : Selects whether the transfer length counter depends on either the source word counts or the destination word count.
- _DMA_DATA_MODE_t : Selects how often data is transferred.
- _DMA_BYTE_ORDER_t : Selects the order of the data bytes.
- _DMA_WORD_SIZE_t : Selects the src/dst data word size.
- _DMA_CH_PRI_t : Selects priority of DMA channels.
- _ADC_EVENT_SRC_t : Selects DMA interrupt channel source.

Macros

- ARM_DMA_API_VERSION : DMA API version.
- DMA_ERROR_UNCONFIGURED : DMA channel has not been configured yet.

Functions

- DMA_GetVersion : Get driver version.
- DMA_Initialize : Initialize DMA driver with default configuration.
- DMA_Configure : Configure particular DMA channel.
- DMA_ConfigureWord : Configure particular DMA channel.
- DMA_ConfigureAddr : Configure DMA channel source and destination addresses.
- DMA_SetInterruptPriority : Configure the DMA interrupt priority.
- DMA_CreateConfigWord : Create DMA channel configuration word.
- DMA_SetConfigWord : Quickly updates the DMA channel configuration.
- DMA_Stop : Stops the DMA transfer.
- DMA_GetCounterValue : Returns the current counter value of DMA channel.
- DMA_GetStatus : Returns the DMA channel status.
- DMA_SignalEvent : Signal DMA events.

18.6.2 CMSIS DMA Driver Typedef Documentation

18.6.2.1 DMA_SEL_t

```
typedef enum _DMA_SEL_t DMA_SEL_t
```

Location: Driver_DMA.h:49

Selects the DMA channel.

18.6.2.2 DMA_TRG_t

```
typedef enum _DMA_TRG_t DMA_TRG_t
```

Location: Driver_DMA.h:63

Selects the DMA src/dst target interface.

18.6.2.3 DMA_SRC_STEP_t

```
typedef enum _DMA_SRC_STEP_t DMA_SRC_STEP_t
```

Location: Driver_DMA.h:85

Selects the step size increment to the DMA channel source address.

18.6.2.4 DMA_DST_STEP_t

```
typedef enum _DMA_DST_STEP_t DMA_DST_STEP_t
```

Location: Driver_DMA.h:107

Selects the step size increment to the DMA channel destination address.

18.6.2.5 DMA_SRC_DST_TRANS_LENGHT_SEL_t

```
typedef enum _DMA_SRC_DST_TRANS_LENGHT_SEL_t DMA_SRC_DST_TRANS_LENGHT_SEL_t
```

Location: Driver_DMA.h:116

Selects whether the transfer length counter depends on either the source word counts or the destination word count.

18.6.2.6 DMA_DATA_MODE_t

```
typedef enum _DMA_DATA_MODE_t DMA_DATA_MODE_t
```

Location: Driver_DMA.h:124

Selects how often data is transferred.

18.6.2.7 DMA_BYTE_ORDER_t

```
typedef enum _DMA_BYTE_ORDER_t DMA_BYTE_ORDER_t
```

Location: Driver_DMA.h:132

Selects the order of the data bytes.

18.6.2.8 DMA_WORD_SIZE_t

```
typedef enum _DMA_WORD_SIZE_t DMA_WORD_SIZE_t
```

Location: Driver_DMA.h:161

Selects the src/dst data word size.

18.6.2.9 DMA_CH_PRI_t

```
typedef enum _DMA_CH_PRI_t DMA_CH_PRI_t
```

Location: Driver_DMA.h:171

Selects priority of DMA channels.

18.6.2.10 ADC_EVENT_SRC_t

```
typedef enum _ADC_EVENT_SRC_t ADC_EVENT_SRC_t
```

Location: Driver_DMA.h:181

Selects DMA interrupt channel source.

18.6.2.11 DMA_SignalEvent_t

```
typedef void(* DMA_SignalEvent_t
```

Location: Driver_DMA.h:263

Pointer to DMA_SignalEvent : Signal Timer event.

18.6.2.12 DMA_CFG_t

```
typedef struct _DMA_CFG_t DMA_CFG_t
```

Location: Driver_DMA.h:277

DMA channel configuration.

18.6.2.13 DMA_ADDR_CFG_t

```
typedef struct _DMA_ADDR_CFG_t DMA_ADDR_CFG_t
```

Location: Driver_DMA.h:288

DMA src/dst address configuration.

18.6.2.14 DMA_PRI_CFG_t

```
typedef struct _DMA_PRI_CFG_t DMA_PRI_CFG_t
```

Location: Driver_DMA.h:299

DMA interrupt priority configuration.

18.6.2.15 DMA_STATUS_t

```
typedef struct _DMA_STATUS_t DMA_STATUS_t
```

Location: Driver_DMA.h:311

DMA status.

18.6.2.16 DRIVER_DMA_t

```
typedef struct _DRIVER_DMA_t DRIVER_DMA_t
```

Location: Driver_DMA.h:331

Access structure of the DMA Driver.

18.6.3 CMSIS DMA Driver Data Structures Type Documentation

18.6.3.1 _DMA_CFG_t

Location: Driver_DMA.h:268

DMA channel configuration.

Data Fields

Type	*Name*	Description
DMA_TRG_t	*src_sel*	DMA source target.
DMA_SRC_STEP_t	*src_step*	Source step mode.
DMA_WORD_SIZE_t	*word_size*	Source word size.
DMA_TRG_t	*dst_sel*	DMA destination target.
DMA_DST_STEP_t	*dst_step*	Destination step mode.
DMA_CH_PRI_t	*ch_priority*	Channel priority.
uint32_t	*__pad0__*	Reserved.

18.6.3.2 _DMA_ADDR_CFG_t

Location: Driver_DMA.h:282

DMA src/dst address configuration.

Data Fields

Type	*Name*	Description
`const void *`	*src_addr*	Source address.
`const void *`	*dst_addr*	Destination address.
`uint32_t`	*counter_len*	Value which when reached triggers the counter event.
`uint32_t`	*transfer_len*	DMA transfer length.

18.6.3.3 _DMA_PRI_CFG_t

Location: Driver_DMA.h:293

DMA interrupt priority configuration.

Data Fields

Type	*Name*	Description
`uint32_t`	*preempt_pri*	Preempt priority.
`uint32_t`	*__pad0__*	Reserved.
`uint32_t`	*subgrp_pri*	Subgroup priority.
`uint32_t`	*__pad1__*	Reserved.

18.6.3.4 _DMA_STATUS_t

Location: Driver_DMA.h:304

DMA status.

Data Fields

Type	*Name*	Description
`uint32_t`	*active*	Transfer was started.
`uint32_t`	*completed*	Transfer was completed.

uint32_t	*counter_reached*	Counter value was reached.
uint32_t	*buffer_fill_lvl*	Error occurred.
uint32_t	*__pad0__*	Reserved.

18.6.3.5 _DRIVER_DMA_t

Location: Driver_DMA.h:316

Access structure of the DMA Driver.

Data Fields

Type	Name	Description
ARM_DRIVER_VERSION(*	*GetVersion)(void)*	Pointer to DMA_GetVersion : Get driver version.
int32_t(*	*Initialize)(DMA_SignalEvent_t cb_event)*	Pointer to DMA_Initialize : Initialize DMA driver.
int32_t(*	*Configure)(DMA_SEL_t sel, const DMA_CFG_t *cfg, DMA_SignalEvent_t cb)*	Pointer to DMA_Configure : Configure DMA channel.
int32_t(*	*ConfigureWord)(DMA_SEL_t sel, uint32_t cfg, DMA_SignalEvent_t cb)*	Pointer to DMA_ConfigureWord : Configure DMA channel.
int32_t(*	*ConfigureAddr)(DMA_SEL_t sel, const DMA_ADDR_CFG_t *pri)*	Pointer to DMA_ConfigureAddr : Configure DMA interrupt priority.
int32_t(*	*SetInterruptPriority)(DMA_SEL_t sel, const DMA_PRI_CFG_t *pri)*	Pointer to DMA_SetInterruptPriority : Configure DMA interrupt priority.
uint32_t(*	*CreateConfigWord)(const DMA_CFG_t *cfg)*	Pointer to DMA_CreateConfigWord : Create DMA channel configuration word.
void(*	*SetConfigWord)(DMA_SEL_t sel, uint32_t cfg)*	Pointer to DMA_SetConfigWord : Quickly update DMA channel configuration word.

`int32_t(*`	*Start)(DMA_SEL_t sel, uint32_t trigger, uint32_t trigger_source)*	Pointer to DMA_Start : Start DMA transfer.
`int32_t(*`	*Stop)(DMA_SEL_t sel)*	Pointer to DMA_Stop : Stop DMA transfer.
`uint32_t(*`	*GetCounterValue)(DMA_SEL_t sel)*	Pointer to DMA_GetCounterValue : Get the current channel transfer counter.
`DMA_STATUS_t(*`	*GetStatus)(DMA_SEL_t sel)*	Pointer to DMA_GetStatus : Returns DMA channel status.

18.6.4 CMSIS DMA Driver Enumeration Type Documentation

18.6.4.1 _DMA_SEL_t

Location: Driver_DMA.h:44

Selects the DMA channel.

Members

- `DMA_CH_0 = 0`

  DMA channel 0.

- `DMA_CH_1 = 1`

  DMA channel 1.

- `DMA_CH_2 = 2`

  DMA channel 2.

- `DMA_CH_3 = 3`

  DMA channel 3.

18.6.4.2 _DMA_TRG_t

Location: Driver_DMA.h:54

Selects the DMA src/dst target interface.

Members

- `DMA_TRG_MEM = 0`
- `DMA_TRG_SPI0 = 1`

  Source / destination target = SPI0.

- `DMA_TRG_SPI1 = 2`

  Source / destination target = SPI1.

- `DMA_TRG_I2C0 = 3`

  Source / destination target = I2C0.

- `DMA_TRG_I2C1 = 4`

  Source / destination target = I2C1.

- `DMA_TRG_UART = 5`

  Source / destination target = UART.

- `DMA_TRG_PCM = 6`

  Source / destination target = PCM.

- `DMA_TRG_TOF = 7`

  Source / destination target = TOF.

18.6.4.3 _DMA_SRC_STEP_t

Location: Driver_DMA.h:68

Selects the step size increment to the DMA channel source address.

Members

- `DMA_CFG0_SRC_ADDR_STATIC = 0x00`

  Do not increment the source address used by DMA channel.

- `DMA_CFG0_SRC_ADDR_INCR_1 = 0x01`

  Set the step size of DMA channel source address to 1.

- `DMA_CFG0_SRC_ADDR_INCR_2 = 0x02`

Set the step size of DMA channel source address to 2.

- `DMA_CFG0_SRC_ADDR_INCR_3 = 0x03`

  Set the step size of DMA channel source address to 3.

- `DMA_CFG0_SRC_ADDR_INCR_4 = 0x04`

  Set the step size of DMA channel source address to 4.

- `DMA_CFG0_SRC_ADDR_INCR_5 = 0x05`

  Set the step size of DMA channel source address to 5.

- `DMA_CFG0_SRC_ADDR_INCR_6 = 0x06`

  Set the step size of DMA channel source address to 6.

- `DMA_CFG0_SRC_ADDR_INCR_7 = 0x07`

Set the step size of DMA channel source address to 7.

- `DMA_CFG0_SRC_ADDR_DECR_8 = 0x08`

Set the step size of DMA channel source address to negative 8.

- `DMA_CFG0_SRC_ADDR_DECR_7 = 0x09`

Set the step size of DMA channel source address to negative 7.

- `DMA_CFG0_SRC_ADDR_DECR_6 = 0x0A`

Set the step size of DMA channel source address to negative 6.

- `DMA_CFG0_SRC_ADDR_DECR_5 = 0x0B`

Set the step size of DMA channel source address to negative 5.

- `DMA_CFG0_SRC_ADDR_DECR_4 = 0x0C`

Set the step size of DMA channel source address to negative 4.

- `DMA_CFG0_SRC_ADDR_DECR_3 = 0x0D`

Set the step size of DMA channel source address to negative 3.

- `DMA_CFG0_SRC_ADDR_DECR_2 = 0x0E`

Set the step size of DMA channel source address to negative 2.

- `DMA_CFG0_SRC_ADDR_DECR_1 = 0x0F`

Set the step size of DMA channel source address to negative 1.

18.6.4.4 _DMA_DST_STEP_t

Location: Driver_DMA.h:90

Selects the step size increment to the DMA channel destination address.

Members

- `DMA_CFG0_DEST_ADDR_STATIC = 0x00`

  Do not increment the destination address used by DMA channel.

- `DMA_CFG0_DEST_ADDR_INCR_1 = 0x01`

  Set the step size of DMA channel destination address to 1.

- `DMA_CFG0_DEST_ADDR_INCR_2 = 0x02`

  Set the step size of DMA channel destination address to 2.

- `DMA_CFG0_DEST_ADDR_INCR_3 = 0x03`

  Set the step size of DMA channel destination address to 3.

- `DMA_CFG0_DEST_ADDR_INCR_4 = 0x04`

  Set the step size of DMA channel destination address to 4.

- `DMA_CFG0_DEST_ADDR_INCR_5 = 0x05`

  Set the step size of DMA channel destination address to 5.

- `DMA_CFG0_DEST_ADDR_INCR_6 = 0x06`

  Set the step size of DMA channel destination address to 6.

- `DMA_CFG0_DEST_ADDR_INCR_7 = 0x07`

  Set the step size of DMA channel destination address to 7.

- `DMA_CFG0_DEST_ADDR_DECR_8 = 0x08`

  Set the step size of DMA channel destination address to negative 8.

- `DMA_CFG0_DEST_ADDR_DECR_7 = 0x09`

Set the step size of DMA channel destination address to negative 7.

- `DMA_CFG0_DEST_ADDR_DECR_6 = 0x0A`

Set the step size of DMA channel destination address to negative 6.

- `DMA_CFG0_DEST_ADDR_DECR_5 = 0x0B`

Set the step size of DMA channel destination address to negative 5.

- `DMA_CFG0_DEST_ADDR_DECR_4 = 0x0C`

Set the step size of DMA channel destination address to negative 4.

- `DMA_CFG0_DEST_ADDR_DECR_3 = 0x0D`

Set the step size of DMA channel destination address to negative 3.

- `DMA_CFG0_DEST_ADDR_DECR_2 = 0x0E`

Set the step size of DMA channel destination address to negative 2.

- `DMA_CFG0_DEST_ADDR_DECR_1 = 0x0F`

Set the step size of DMA channel destination address to negative 1.

18.6.4.5 _DMA_SRC_DST_TRANS_LENGHT_SEL_t

Location: Driver_DMA.h:112

Selects whether the transfer length counter depends on either the source word counts or the destination word count.

Members

- `DMA_CFG0_DEST_TRANS_LENGTH_SEL = 0x00`

  Transfer length counter depends on the destination word count.

- `DMA_CFG0_SRC_TRANS_LENGTH_SEL = 0x01`

  Transfer length counter depends on the size word count.

18.6.4.6 _DMA_DATA_MODE_t

Location: Driver_DMA.h:121

Selects how often data is transferred.

Members

- `DMA_REPEAT = 0x0U`

  Data to be transfered repeatedly.

- `DMA_SINGLE = 0x1U`

  Single data transfer.

18.6.4.7 _DMA_BYTE_ORDER_t

Location: Driver_DMA.h:129

Selects the order of the data bytes.

Members

- `DMA_ENDIANNESS_LITTLE = 0x0U`

  Little endian to be used.

- `DMA_ENDIANNESS_BIG = 0x1U`

  Big endian to be used.

18.6.4.8 _DMA_WORD_SIZE_t

Location: Driver_DMA.h:137

Selects the src/dst data word size.

Members

- `DMA_CFG0_DEST_WORD_SIZE_32BITS_TO_32BITS = 0x00`

  Source data uses 32-bit word and destination data uses 32-bit word.

- `DMA_CFG0_DEST_WORD_SIZE_32BITS_TO_4BITS = 0x01`

Source data uses 32-bit word and destination data uses 4-bit word.

- `DMA_CFG0_DEST_WORD_SIZE_32BITS_TO_8BITS = 0x02`

  Source data uses 32-bit word and destination data uses 8-bit word.

- `DMA_CFG0_DEST_WORD_SIZE_32BITS_TO_16BITS = 0x04`

  Source data uses 32-bit word and destination data uses 16-bit word.

- `DMA_CFG0_DEST_WORD_SIZE_4BITS_TO_32BITS = 0x08`

  Source data uses 4-bit word and destination data uses 32-bit word.

- `DMA_CFG0_DEST_WORD_SIZE_4BITS_TO_4BITS = 0x09`

  Source data uses 4-bit word and destination data uses 4-bit word.

- `DMA_CFG0_DEST_WORD_SIZE_4BITS_TO_8BITS = 0x0A`

  Source data uses 4-bit word and destination data uses 8-bit word.

- `DMA_CFG0_DEST_WORD_SIZE_4BITS_TO_16BITS = 0x0C`

  Source data uses 4-bit word and destination data uses 16-bit word.

- `DMA_CFG0_DEST_WORD_SIZE_4BITS_TO_24BITS = 0x0E`

  Source data uses 4-bit word and destination data uses 24-bit word.

- `DMA_CFG0_DEST_WORD_SIZE_8BITS_TO_32BITS = 0x10`

  Source data uses 8-bit word and destination data uses 32-bit word.

- `DMA_CFG0_DEST_WORD_SIZE_8BITS_TO_4BITS = 0x11`

  Source data uses 8-bit word and destination data uses 4-bit word.

- `DMA_CFG0_DEST_WORD_SIZE_8BITS_TO_8BITS = 0x12`

  Source data uses 8-bit word and destination data uses 8-bit word.

- `DMA_CFG0_DEST_WORD_SIZE_8BITS_TO_16BITS = 0x14`

Source data uses 8-bit word and destination data uses 16-bit word.

- `DMA_CFG0_DEST_WORD_SIZE_8BITS_TO_24BITS = 0x16`

  Source data uses 8-bit word and destination data uses 24-bit word.

- `DMA_CFG0_DEST_WORD_SIZE_16BITS_TO_32BITS = 0x20`

  Source data uses 16-bit word and destination data uses 32-bit word.

- `DMA_CFG0_DEST_WORD_SIZE_16BITS_TO_4BITS = 0x21`

  Source data uses 16-bit word and destination data uses 4-bit word.

- `DMA_CFG0_DEST_WORD_SIZE_16BITS_TO_8BITS = 0x22`

  Source data uses 16-bit word and destination data uses 8-bit word.

- `DMA_CFG0_DEST_WORD_SIZE_16BITS_TO_16BITS = 0x24`

  Source data uses 16-bit word and destination data uses 16-bit word.

- `DMA_CFG0_DEST_WORD_SIZE_16BITS_TO_24BITS = 0x26`

Source data uses 16-bit word and destination data uses 24-bit word.

- `DMA_CFG0_DEST_WORD_SIZE_24BITS_TO_4BITS = 0x31`

  Source data uses 24-bit word and destination data uses 4-bit word.

- `DMA_CFG0_DEST_WORD_SIZE_24BITS_TO_8BITS = 0x32`

  Source data uses 24-bit word and destination data uses 8-bit word.

- `DMA_CFG0_DEST_WORD_SIZE_24BITS_TO_16BITS = 0x34`

  Source data uses 24-bit word and destination data uses 16-bit word.

- `DMA_CFG0_DEST_WORD_SIZE_24BITS_TO_24BITS = 0x36`

  Source data uses 24-bit word and destination data uses 24-bit word.

18.6.4.9 _DMA_CH_PRI_t

Location: Driver_DMA.h:166

Selects priority of DMA channels.

Members

- `DMA_CH_PRI_0 = 0x0U`

  Channel priority = 0.

- `DMA_CH_PRI_1 = 0x1U`

  Channel priority = 1.

- `DMA_CH_PRI_2 = 0x2U`

  Channel priority = 2.

- `DMA_CH_PRI_3 = 0x3U`

  Channel priority = 3.

18.6.4.10 _ADC_EVENT_SRC_t

Location: Driver_DMA.h:176

Selects DMA interrupt channel source.

Members

- `DMA_DMA0_EVENT = 1 << DMA_CH_0`

  DMA channel 0 event.

- `DMA_DMA1_EVENT = 1 << DMA_CH_1`

  DMA channel 1 event.

- `DMA_DMA2_EVENT = 1 << DMA_CH_2`

DMA channel 2 event.

- `DMA_DMA3_EVENT = 1 << DMA_CH_3`

  DMA channel 3 event.

18.6.5 CMSIS DMA Driver Macro Definition Documentation

18.6.5.1 ARM_DMA_API_VERSION

```
#define ARM_DMA_API_VERSION ARM_DRIVER_VERSION_MAJOR_MINOR(1,0)
```

Location: Driver_DMA.h:37

DMA API version.

18.6.5.2 DMA_ERROR_UNCONFIGURED

```
#define DMA_ERROR_UNCONFIGURED (ARM_DRIVER_ERROR_SPECIFIC - 1)
```

Location: Driver_DMA.h:184

DMA channel has not been configured yet.

18.6.6 CMSIS DMA Driver Function Documentation

18.6.6.1 DMA_GetVersion

```
ARM_DRIVER_VERSION DMA_GetVersion()
```

Location: Driver_DMA.c:21

Get driver version.

Returns:

ARM_DRIVER_VERSION

18.6.6.2 DMA_Initialize

```
int32_t DMA_Initialize()
```

Location: Driver_DMA.c:22

Initialize DMA driver with default configuration.

Returns:

execution_status

18.6.6.3 DMA_Configure

```
int32_t DMA_Configure(DMA_SEL_t sel, const DMA_CFG_t * cfg, DMA_SignalEvent_t cb)
```

Location: Driver_DMA.c:23

Configure particular DMA channel.

Returns:

execution_status

Parameters

Direction	Name	Description
in	*sel*	DMA channel to be configured (DMA_SEL_t)
in	*cfg*	Pointer to DMA_CFG_t
in	*cb*	Pointer to DMA_SignalEvent_t

18.6.6.4 DMA_ConfigureWord

```
int32_t DMA_ConfigureWord(DMA_SEL_t sel, uint32_t cfg, DMA_SignalEvent_t cb)
```

Location: Driver_DMA.c:24

Configure particular DMA channel.

Returns:

execution_status

Parameters

Direction	Name	Description
in	*sel*	DMA channel to be configured (DMA_SEL_t)
in	*cfg*	Configuration word
in	*cb*	Pointer to DMA_SignalEvent_t

18.6.6.5 DMA_ConfigureAddr

```
int32_t DMA_ConfigureAddr(DMA_SEL_t sel, const DMA_ADDR_CFG_t * cfg)
```

Location: Driver_DMA.c:25

Configure DMA channel source and destination addresses.

Returns:

execution_status

Parameters

Direction	Name	Description
in	*sel*	DMA to be configured (DMA_SEL_t)
in	*cfg*	Pointer to DMA_ADDR_CFG_t

18.6.6.6 DMA_SetInterruptPriority

```
int32_t DMA_SetInterruptPriority(DMA_SEL_t sel, const DMA_PRI_CFG_t * cfg)
```

Location: Driver_DMA.c:26

Configure the DMA interrupt priority.

Returns:

execution_status

Parameters

Direction	Name	Description
in	*sel*	DMA channel to be configured (DMA_SEL_t)
in	*cfg*	Pointer to DMA_PRI_CFG_t

18.6.6.7 DMA_CreateConfigWord

```
uint32_t DMA_CreateConfigWord(const DMA_CFG_t * cfg)
```

Location: Driver_DMA.c:27

Create DMA channel configuration word.

Returns:

configuration word

Parameters

Direction	Name	Description
in	*cfg*	Pointer to DMA_CFG_t

18.6.6.8 DMA_SetConfigWord

```
int32_t DMA_SetConfigWord(DMA_SEL_t sel, uint32_t cfg)
```

Location: Driver_DMA.c:28

Quickly updates the DMA channel configuration.

Returns:

none

Parameters

Direction	Name	Description
in	*sel*	DMA channel to be configured (DMA_SEL_t)
in	*cfg*	configuration word

18.6.6.9 DMA_Stop

```
int32_t DMA_Stop(DMA_SEL_t sel)
```

Location: Driver_DMA.c:30

Stops the DMA transfer.

Returns:

execution_status

Parameters

Direction	Name	Description
in	*sel*	DMA channel number to be used(DMA_SEL_t)

18.6.6.10 DMA_GetCounterValue

```
uint32_t DMA_GetCounterValue(DMA_SEL_t sel)
```

Location: Driver_DMA.c:31

Returns the current counter value of DMA channel.

Returns:

DMA channel counter value

Parameters

Direction	Name	Description
in	*sel*	DMA channel value to be read (DMA_SEL_t)

18.6.6.11 DMA_GetStatus

```
DMA_STATUS_t DMA_GetStatus(DMA_SEL_t sel)
```

Location: Driver_DMA.c:32

Returns the DMA channel status.

Clears the status register on read. sel DMA channel value to be read (DMA_SEL_t)
Returns:

DMA channel status (DMA_STATUS_t)

Returns:

DMA channel status (DMA_STATUS_t)

Parameters

Direction	Name	Description
in	*sel*	DMA channel value to be read (DMA_SEL_t)

18.6.6.12 DMA_SignalEvent

```
void DMA_SignalEvent(uint32_t event)
```

Location: Driver_DMA.c:34

Signal DMA events.

Returns:

None

Parameters

Direction	Name	Description
in	*event*	Notification mask (_ADC_EVENT_SRC_t)

18.7 CMSIS GPIO Driver

CMSIS GPIO Driver Reference.

18.7.1 Summary

Typedefs

- GPIO_SEL_t : GPIO Control Codes: GPIO Selection.
- GPIO_DIR_t : GPIO Control Codes: GPIO direction.
- GPIO_INT_SEL_t : GPIO Control Codes: GPIO INT Selection.
- GPIO_DRIVE_t : GPIO Control Codes: Drive strength.
- GPIO_LPF_t : GPIO Control Codes: Low pass filter.
- GPIO_PULL_t : GPIO Control Codes: Pull control.
- GPIO_OUTPUT_LEVEL_t : GPIO Control Codes: Output Level.
- GPIO_MODE_t : GPIO Control Codes: IO Mode.
- GPIO_FUNC_REGISTERS_t : GPIO Control Codes: GPIO alternative function registers.
- GPIO_EN_DIS_t : GPIO Control Codes: Enable / Disable values.
- GPIO_EVENT_t : GPIO Control Codes: Interrupts events.
- GPIO_DBC_CLK_t : GPIO Control Codes: Debounce clock source.
- GPIO_DRIVE_STRENGTHS_t : GPIO Control Codes: Pads strength.
- GPIO_SignalEvent_t : Pointer to GPIO_SignalEvent : Signal GPIO Event.
- GPIO_DBF_CFG_t : Debounce filter configuration.
- GPIO_PRI_CFG_t : GPIO interrupt priority configuration.
- GPIO_CFG_t : GPIO Driver configuration.
- GPIO_PAD_CFG_t : GPIO PAD configuration.
- GPIO_INT_CFG_t : GPIO INT configuration.
- GPIO_EXTCLK_CFG_t : External clock pad configuration.
- GPIO_JTAG_SW_CFG_t : JTAG configuration.
- DRIVER_GPIO_t : Access structure of the GPIO Driver.

Data Structures

- _GPIO_DBF_CFG_t : Debounce filter configuration.
- _GPIO_PRI_CFG_t : GPIO interrupt priority configuration.
- _GPIO_CFG_t : GPIO Driver configuration.
- _GPIO_PAD_CFG_t : GPIO PAD configuration.
- _GPIO_INT_CFG_t : GPIO INT configuration.
- _GPIO_EXTCLK_CFG_t : External clock pad configuration.
- _GPIO_JTAG_SW_CFG_t : JTAG configuration.
- _DRIVER_GPIO_t : Access structure of the GPIO Driver.

Enumerations

- _GPIO_SEL_t : GPIO Control Codes: GPIO Selection.
- _GPIO_DIR_t : GPIO Control Codes: GPIO direction.
- _GPIO_INT_SEL_t : GPIO Control Codes: GPIO INT Selection.
- _GPIO_DRIVE_t : GPIO Control Codes: Drive strength.
- _GPIO_LPF_t : GPIO Control Codes: Low pass filter.
- _GPIO_PULL_t : GPIO Control Codes: Pull control.
- _GPIO_OUTPUT_LEVEL_t : GPIO Control Codes: Output Level.
- _GPIO_MODE_t : GPIO Control Codes: IO Mode.
- _GPIO_FUNC_REGISTERS_t : GPIO Control Codes: GPIO alternative function registers.
- _GPIO_EN_DIS_t : GPIO Control Codes: Enable / Disable values.
- _GPIO_EVENT_t : GPIO Control Codes: Interrupts events.

- _GPIO_DBC_CLK_t : GPIO Control Codes: Debounce clock source.
- _GPIO_DRIVE_STRENGTHS_t : GPIO Control Codes: Pads strength.

Macros

- ARM_GPIO_API_VERSION : GPIO API version.
- GPIO_EVENT_0_IRQ : GPIO Event.
- GPIO_EVENT_1_IRQ : GPIO1 interrupt event value.
- GPIO_EVENT_2_IRQ : GPIO2 interrupt event value.
- GPIO_EVENT_3_IRQ : GPIO3 interrupt event value.

Functions

- GPIO_GetVersion : Get driver version.
- GPIO_Initialize : Initialize the GPIO driver.
- GPIO_Configure : Configure common GPIO settings.
- GPIO_ConfigurePad : Configure the GPIO pad.
- GPIO_ConfigureInterrupt : Configure the GPIO interrupt.
- GPIO_SetInterruptPriority : Configure GPIO interrupt priority.
- GPIO_ConfigureJTAG : Configure the GPIO JTAG mode.
- GPIO_SetHigh : Set particular GPIO pad.
- GPIO_ToggleValue : Toggle particular GPIO pad.
- GPIO_SetLow : Reset particular GPIO pad.
- GPIO_ReadValue : Returns the selected GPIO pad value.
- GPIO_ResetAltFuncRegister : Reset the particular alternative function register.
- GPIO_SignalEvent : Signal GPIO events.

18.7.2 CMSIS GPIO Driver Typedef Documentation

18.7.2.1 GPIO_SEL_t

```
typedef enum _GPIO_SEL_t GPIO_SEL_t
```

Location: Driver_GPIO.h:59

GPIO Control Codes: GPIO Selection.

18.7.2.2 GPIO_DIR_t

```
typedef enum _GPIO_DIR_t GPIO_DIR_t
```

Location: Driver_GPIO.h:80

GPIO Control Codes: GPIO direction.

18.7.2.3 GPIO_INT_SEL_t

```
typedef enum _GPIO_INT_SEL_t GPIO_INT_SEL_t
```

Location: Driver_GPIO.h:88

GPIO Control Codes: GPIO INT Selection.

18.7.2.4 GPIO_DRIVE_t

```
typedef enum _GPIO_DRIVE_t GPIO_DRIVE_t
```

Location: Driver_GPIO.h:96

GPIO Control Codes: Drive strength.

18.7.2.5 GPIO_LPF_t

```
typedef enum _GPIO_LPF_t GPIO_LPF_t
```

Location: Driver_GPIO.h:102

GPIO Control Codes: Low pass filter.

18.7.2.6 GPIO_PULL_t

```
typedef enum _GPIO_PULL_t GPIO_PULL_t
```

Location: Driver_GPIO.h:110

GPIO Control Codes: Pull control.

18.7.2.7 GPIO_OUTPUT_LEVEL_t

```
typedef enum _GPIO_OUTPUT_LEVEL_t GPIO_OUTPUT_LEVEL_t
```

Location: Driver_GPIO.h:116

GPIO Control Codes: Output Level.

18.7.2.8 GPIO_MODE_t

```
typedef enum _GPIO_MODE_t GPIO_MODE_t
```

Location: Driver_GPIO.h:260

GPIO Control Codes: IO Mode.

18.7.2.9 GPIO_FUNC_REGISTERS_t

```
typedef enum _GPIO_FUNC_REGISTERS_t GPIO_FUNC_REGISTERS_t
```

Location: Driver_GPIO.h:278

GPIO Control Codes: GPIO alternative function registers.

18.7.2.10 GPIO_EN_DIS_t

```
typedef enum _GPIO_EN_DIS_t GPIO_EN_DIS_t
```

Location: Driver_GPIO.h:284

GPIO Control Codes: Enable / Disable values.

18.7.2.11 GPIO_EVENT_t

```
typedef enum _GPIO_EVENT_t GPIO_EVENT_t
```

Location: Driver_GPIO.h:295

GPIO Control Codes: Interrupts events.

18.7.2.12 GPIO_DBC_CLK_t

```
typedef enum _GPIO_DBC_CLK_t GPIO_DBC_CLK_t
```

Location: Driver_GPIO.h:301

GPIO Control Codes: Debounce clock source.

18.7.2.13 GPIO_DRIVE_STRENGTHS_t

```
typedef enum _GPIO_DRIVE_STRENGTHS_t GPIO_DRIVE_STRENGTHS_t
```

Location: Driver_GPIO.h:307

GPIO Control Codes: Pads strength.

18.7.2.14 GPIO_SignalEvent_t

```
typedef void(* GPIO_SignalEvent_t
```

Location: Driver_GPIO.h:392

Pointer to GPIO_SignalEvent : Signal GPIO Event.

18.7.2.15 GPIO_DBF_CFG_t

```
typedef struct _GPIO_DBF_CFG_t GPIO_DBF_CFG_t
```

Location: Driver_GPIO.h:402

Debounce filter configuration.

18.7.2.16 GPIO_PRI_CFG_t

```
typedef struct _GPIO_PRI_CFG_t GPIO_PRI_CFG_t
```

Location: Driver_GPIO.h:413

GPIO interrupt priority configuration.

18.7.2.17 GPIO_CFG_t

```
typedef struct _GPIO_CFG_t GPIO_CFG_t
```

Location: Driver_GPIO.h:423

GPIO Driver configuration.

18.7.2.18 GPIO_PAD_CFG_t

```
typedef struct _GPIO_PAD_CFG_t GPIO_PAD_CFG_t
```

Location: Driver_GPIO.h:436

GPIO PAD configuration.

18.7.2.19 GPIO_INT_CFG_t

```
typedef struct _GPIO_INT_CFG_t GPIO_INT_CFG_t
```

Location: Driver_GPIO.h:448

GPIO INT configuration.

18.7.2.20 GPIO_EXTCLK_CFG_t

```
typedef struct _GPIO_EXTCLK_CFG_t GPIO_EXTCLK_CFG_t
```

Location: Driver_GPIO.h:458

External clock pad configuration.

18.7.2.21 GPIO_JTAG_SW_CFG_t

```
typedef struct _GPIO_JTAG_SW_CFG_t GPIO_JTAG_SW_CFG_t
```

Location: Driver_GPIO.h:473

JTAG configuration.

18.7.2.22 DRIVER_GPIO_t

```
typedef struct _DRIVER_GPIO_t DRIVER_GPIO_t
```

Location: Driver_GPIO.h:492

Access structure of the GPIO Driver.

18.7.3 CMSIS GPIO Driver Data Structures Type Documentation

18.7.3.1 _GPIO_DBF_CFG_t

Location: Driver_GPIO.h:397

Debounce filter configuration.

Data Fields

Type	*Name*	Description
uint8_t	*count*	Debounce filter count value
GPIO_DBC_CLK_t	*clk_source*	Debounce filter clock source.
uint8_t	*__pad0__*	Reserved

18.7.3.2 _GPIO_PRI_CFG_t

Location: Driver_GPIO.h:407

GPIO interrupt priority configuration.

Data Fields

Type	*Name*	Description
uint32_t	*preempt_pri*	Preempt priority
uint32_t	*__pad0__*	Reserved
uint32_t	*subgrp_pri*	Subgroup priority.
uint32_t	*__pad1__*	Reserved

18.7.3.3 _GPIO_CFG_t

Location: Driver_GPIO.h:418

GPIO Driver configuration.

Data Fields

Type	*Name*	Description
GPIO_DRIVE_STRENGTHS_t	*drive_strengths*	Drive strengths configuration.
uint8_t	*__pad0__*	Reserved
GPIO_DBF_CFG_t	*debounce_cfg*	Debounce filter configuration.

18.7.3.4 _GPIO_PAD_CFG_t

Location: Driver_GPIO.h:428

GPIO PAD configuration.

Data Fields

Type	*Name*	Description
GPIO_PULL_t	*pull_mode*	Pull control.
GPIO_DRIVE_t	*drive_mode*	Drive mode.
GPIO_LPF_t	*lpf_en*	Low pass filter enable.

GPIO_MODE_t	*io_mode*	IO mode.
GPIO_OUTPUT_LEVEL_t	*output_level*	Output level.
uint8_t	*__pad0__*	Reserved.

18.7.3.5 _GPIO_INT_CFG_t

Location: Driver_GPIO.h:441

GPIO INT configuration.

Data Fields

Type	*Name*	Description
GPIO_SEL_t	*src_sel*	Interrupt source selection.
GPIO_EVENT_t	*event*	Event selection
GPIO_EN_DIS_t	*debounce_en*	Debouce filter enable
GPIO_EN_DIS_t	*interrup_en*	Interrupt enable flag
uint8_t	*__pad0__*	Reserved

18.7.3.6 _GPIO_EXTCLK_CFG_t

Location: Driver_GPIO.h:453

External clock pad configuration.

Data Fields

Type	*Name*	Description
GPIO_PULL_t	*pull_mode*	Pull control
GPIO_LPF_t	*lpf_en*	Low pass filter enable.
uint8_t	*__pad0__*	Reserved

18.7.3.7 _GPIO_JTAG_SW_CFG_t

Location: Driver_GPIO.h:463

JTAG configuration.

Data Fields

Type	*Name*	Description
GPIO_LPF_t	*swclk_jtck_lpf_en*	SWCLK/JTCK low pass filter enable
GPIO_LPF_t	*swdio_jtms_lpf_en*	SWDIO/JTMS low pass filter enable
GPIO_EN_DIS_t	*jtag_data_en*	JTAG data available on GPIO[2:3]
GPIO_EN_DIS_t	*jtag_trst_en*	JTAG trst available on GPIO4
GPIO_PULL_t	*swclk_jtck_pull*	SWCLK/JTCK pull mode
GPIO_PULL_t	*swdio_jtms_pull*	SWDIO/JTMS pull mode
GPIO_DRIVE_t	*swdio_jtms_drive*	SWDIO/JTMS drive mode
uint8_t	*__pad0__*	Reserved

18.7.3.8 _DRIVER_GPIO_t

Location: Driver_GPIO.h:478

Access structure of the GPIO Driver.

Data Fields

Type	*Name*	Description
ARM_DRIVER_VERSION(*	*GetVersion)(void)*	Pointer to GPIO_GetVersion : Get driver version.
int32_t(*	*Initialize)(GPIO_SignalEvent_t cb)*	Pointer to GPIO_Initialize : Initialize the GPIO driver.
int32_t(*	*Configure)(const GPIO_*	Pointer to GPIO_Configure : Configure common GPIO

	*CFG_t *cfg)*	settings.
`int32_t(*`	*ConfigurePad)(GPIO_SEL_t sel, const GPIO_PAD_CFG_t *cfg)*	Pointer to GPIO_ConfigurePad : Configure the GPIO pad.
`int32_t(*`	*ConfigureInterrupt)(GPIO_INT_SEL_t sel, const GPIO_INT_CFG_t *cfg)*	Pointer to GPIO_ConfigureInterrupt : Configure the GPIO interrupt.
`int32_t(*`	*SetInterruptPriority)(GPIO_INT_SEL_t sel, const GPIO_PRI_CFG_t *pri)*	Pointer to GPIO_SetInterruptPriority : Configure GPIO interrupt priority.
`int32_t(*`	*ConfigureJTAGSW)(const GPIO_JTAG_SW_CFG_t *cfg)*	Pointer to GPIO_ConfigureJTAG : Configure the GPIO JTAG mode.
`void(*`	*SetDir)(GPIO_SEL_t sel, GPIO_DIR_t dir)*	Pointer to GPIO_SetDir : Set particular GPIO pad direction.
`void(*`	*SetHigh)(GPIO_SEL_t sel)*	Pointer to GPIO_SetHigh : Set particular GPIO pad.
`void(*`	*ToggleValue)(GPIO_SEL_t sel)*	Pointer to GPIO_ToggleValue : Toggle particular GPIO pad.
`void(*`	*SetLow)(GPIO_SEL_t sel)*	Pointer to GPIO_SetLow : Reset particular GPIO pad.
`uint32_t(*`	*ReadValue)(GPIO_SEL_t sel)*	Pointer to GPIO_ReadValue : Return the selected GPIO value.
`int32_t(*`	*ResetAltFuncRegister)(GPIO_FUNC_REGISTERS_t sel)*	Pointer to GPIO_ResetAltFuncRegister : Reset GPIO alternative function register.

18.7.4 CMSIS GPIO Driver Enumeration Type Documentation

18.7.4.1 _GPIO_SEL_t

Location: Driver_GPIO.h:42

GPIO Control Codes: GPIO Selection.

Members

- `GPIO_0 = 0x0`

  GPIO pad 0

- `GPIO_1 = 0x1`

  GPIO pad 1

- `GPIO_2 = 0x2`

  GPIO pad 2

- `GPIO_3 = 0x3`

  GPIO pad 3

- `GPIO_4 = 0x4`

  GPIO pad 4

- `GPIO_5 = 0x5`

  GPIO pad 5

- `GPIO_6 = 0x6`

  GPIO pad 6

- `GPIO_7 = 0x7`

  GPIO pad 7

- `GPIO_8 = 0x8`

  GPIO pad 8

- `GPIO_9 = 0x9`

  GPIO pad 9

- `GPIO_10 = 0xA`

  GPIO pad 10.

- `GPIO_11 = 0xB`

  GPIO pad 11.

- `GPIO_12 = 0xC`

  GPIO pad 12.

- `GPIO_13 = 0xD`

  GPIO pad 13.

- `GPIO_14 = 0xE`

  GPIO pad 14.

- `GPIO_15 = 0xF`

  GPIO pad 15.

18.7.4.2 _GPIO_DIR_t

Location: Driver_GPIO.h:62

GPIO Control Codes: GPIO direction.

Members

- `GPIO_DIR_IN = 0x0`

  GPIO direction input

- `GPIO_DIR_OUT_0 = 0x1`

  GPIO direction output 0

- `GPIO_DIR_OUT_1 = 0x2`

  GPIO direction output 1

- `GPIO_DIR_OUT_2 = 0x4`

  GPIO direction output 2

- `GPIO_DIR_OUT_3 = 0x8`

  GPIO direction output 3

- `GPIO_DIR_OUT_4 = 0x10`

  GPIO direction output 4

- `GPIO_DIR_OUT_5 = 0x20`

  GPIO direction output 5

- `GPIO_DIR_OUT_6 = 0x40`

  GPIO direction output 6

- `GPIO_DIR_OUT_7 = 0x80`

  GPIO direction output 7

- `GPIO_DIR_OUT_8 = 0x100`

  GPIO direction output 8

- `GPIO_DIR_OUT_9 = 0x200`

  GPIO direction output 9

- `GPIO_DIR_OUT_10 = 0x400`

  GPIO direction output 10

- `GPIO_DIR_OUT_11 = 0x800`

  GPIO direction output 11

- `GPIO_DIR_OUT_12 = 0x1000`

  GPIO direction output 12

- `GPIO_DIR_OUT_13 = 0x2000`

  GPIO direction output 13

- `GPIO_DIR_OUT_14 = 0x4000`

  GPIO direction output 14

- `GPIO_DIR_OUT_15 = 0x8000`

  GPIO direction output 15

18.7.4.3 _GPIO_INT_SEL_t

Location: Driver_GPIO.h:83

GPIO Control Codes: GPIO INT Selection.

Members

- `GPIO_INT_0 = 0x0`

  GPIO interrupt 0.

- `GPIO_INT_1 = 0x1`

  GPIO interrupt 1.

- `GPIO_INT_2 = 0x2`

  GPIO interrupt 2.

- `GPIO_INT_3 = 0x3`

  GPIO interrupt 3.

18.7.4.4 _GPIO_DRIVE_t

Location: Driver_GPIO.h:91

GPIO Control Codes: Drive strength.

Members

- `GPIO_2X = 0x0`

  2x drive strength

- `GPIO_3X = 0x1`

  3x drive strength

- `GPIO_5X = 0x2`

  5x drive strength

- `GPIO_6X = 0x3`

  6x drive strength

18.7.4.5 _GPIO_LPF_t

Location: Driver_GPIO.h:99

GPIO Control Codes: Low pass filter.

Members

- `GPIO_LPF_DISABLED = 0x0`

  Low pass filter disabled.

- `GPIO_LPF_ENABLED = 0x1`

  Low pass filter enabled

18.7.4.6 _GPIO_PULL_t

Location: Driver_GPIO.h:105

GPIO Control Codes: Pull control.

Members

- `GPIO_PC_NO_PULL = 0x0`

  No pull selected

- `GPIO_PC_WEAK_PULL_UP = 0x1`

  Weak pull-up selected

- `GPIO_PC_WEAK_PULL_DOWN = 0x2`

  Weak pull-down selected.

- `GPIO_PC_STRONG_PULL_UP = 0x3`

  Strong pull-up selected.

18.7.4.7 _GPIO_OUTPUT_LEVEL_t

Location: Driver_GPIO.h:113

GPIO Control Codes: Output Level.

Members

- `GPIO_OUTPUT_LEVEL_LOW = 0x0`

  Initial output level set to LOW.

- `GPIO_OUTPUT_LEVEL_HIGH = 0x1`

  Initial output level set to HIGH.

18.7.4.8 _GPIO_MODE_t

Location: Driver_GPIO.h:119

GPIO Control Codes: IO Mode.

Members

- `MODE_GPIO_DISABLE = 0x000`
- `MODE_GPIO_INPUT = 0x001`
- `MODE_GPIO_GPIO_IN = 0x002`
- `MODE_GPIO_GPIO_OUT = 0x003`
- `MODE_GPIO_SLOWCLK = 0x004`
- `MODE_GPIO_SYSCLK = 0x005`
- `MODE_GPIO_USRCLK = 0x006`
- `MODE_GPIO_RCCLK = 0x007`
- `MODE_GPIO_SWCLK_DIV = 0x008`
- `MODE_GPIO_EXTCLK_DIV = 0x009`
- `MODE_GPIO_RFCLK = 0x00A`
- `MODE_GPIO_STANDBYCLK = 0x00B`
- `MODE_GPIO_SENSORCLK = 0x00C`

- MODE_GPIO_SPI0_IO0 = 0x00D
- MODE_GPIO_SPI0_IO1 = 0x00E
- MODE_GPIO_SPI0_IO2 = 0x00F
- MODE_GPIO_SPI0_IO3 = 0x010
- MODE_GPIO_SPI0_CS = 0x011
- MODE_GPIO_SPI0_CLK = 0x012
- MODE_GPIO_SPI1_IO0 = 0x013
- MODE_GPIO_SPI1_IO1 = 0x014
- MODE_GPIO_SPI1_IO2 = 0x015
- MODE_GPIO_SPI1_IO3 = 0x016
- MODE_GPIO_SPI1_CS = 0x017
- MODE_GPIO_SPI1_CLK = 0x018
- MODE_GPIO_UART0_TX = 0x019
- MODE_GPIO_I2C0_SCL = 0x01A
- MODE_GPIO_I2C0_SDA = 0x01B
- MODE_GPIO_I2C1_SCL = 0x01C
- MODE_GPIO_I2C1_SDA = 0x01D
- MODE_GPIO_PCM0_SERO = 0x01E
- MODE_GPIO_PCM0_FRAME = 0x01F
- MODE_GPIO_PWM0 = 0x020
- MODE_GPIO_PWM1 = 0x021
- MODE_GPIO_PWM2 = 0x022
- MODE_GPIO_PWM3 = 0x023
- MODE_GPIO_PWM4 = 0x024
- MODE_GPIO_PWM0_INV = 0x025
- MODE_GPIO_PWM1_INV = 0x026
- MODE_GPIO_PWM2_INV = 0x027
- MODE_GPIO_PWM3_INV = 0x028
- MODE_GPIO_PWM4_INV = 0x029
- MODE_GPIO_LIN0_TX = 0x02A

- MODE_GPIO_BB_TX_DATA = 0x02B
- MODE_GPIO_BB_TX_DATA_VALID = 0x02C
- MODE_GPIO_BB_SPI_CSN = 0x02D
- MODE_GPIO_BB_SPI_CLK = 0x02E
- MODE_GPIO_BB_SPI_MOSI = 0x02F
- MODE_GPIO_BB_DBG_0 = 0x030
- MODE_GPIO_BB_DBG_1 = 0x031
- MODE_GPIO_BB_DBG_2 = 0x032
- MODE_GPIO_BB_DBG_3 = 0x033
- MODE_GPIO_BB_DBG_4 = 0x034
- MODE_GPIO_BB_DBG_5 = 0x035
- MODE_GPIO_BB_DBG_6 = 0x036
- MODE_GPIO_BB_DBG_7 = 0x037
- MODE_GPIO_BB_BLE_SYNC = 0x038
- MODE_GPIO_BB_BLE_IN_PROCESS = 0x039
- MODE_GPIO_BB_BLE_TX = 0x03A
- MODE_GPIO_BB_BLE_RX = 0x03B
- MODE_GPIO_BB_BLE_PTI_0 = 0x03C
- MODE_GPIO_BB_BLE_PTI_1 = 0x03D
- MODE_GPIO_BB_BLE_PTI_2 = 0x03E
- MODE_GPIO_BB_BLE_PTI_3 = 0x03F
- MODE_GPIO_BB_ANT_SW_EN = 0x040
- MODE_GPIO_BB_ANT_SW_0 = 0x041
- MODE_GPIO_BB_ANT_SW_1 = 0x042
- MODE_GPIO_BB_ANT_SW_2 = 0x043
- MODE_GPIO_BB_ANT_SW_3 = 0x044
- MODE_GPIO_BB_ANT_SW_4 = 0x045
- MODE_GPIO_BB_ANT_SW_5 = 0x046
- MODE_GPIO_BB_ANT_SW_6 = 0x047
- MODE_GPIO_BB_CTE_MODE = 0x048

- MODE_GPIO_BB_CTE_SAMPLE_P = 0x049
- MODE_GPIO_RF_SPI_MISO = 0x04A
- MODE_GPIO_RF_GPIO0 = 0x04B
- MODE_GPIO_RF_GPIO1 = 0x04C
- MODE_GPIO_RF_GPIO2 = 0x04D
- MODE_GPIO_RF_GPIO3 = 0x04E
- MODE_GPIO_RF_GPIO4 = 0x04F
- MODE_GPIO_RF_GPIO5 = 0x050
- MODE_GPIO_RF_GPIO6 = 0x051
- MODE_GPIO_RF_GPIO7 = 0x052
- MODE_GPIO_RF_GPIO8 = 0x053
- MODE_GPIO_RF_GPIO9 = 0x054
- MODE_GPIO_RF_IQ_DATA_P = 0x055
- MODE_GPIO_RF_I_DATA_0 = 0x056
- MODE_GPIO_RF_I_DATA_1 = 0x057
- MODE_GPIO_RF_I_DATA_2 = 0x058
- MODE_GPIO_RF_I_DATA_3 = 0x059
- MODE_GPIO_RF_I_DATA_4 = 0x05A
- MODE_GPIO_RF_I_DATA_5 = 0x05B
- MODE_GPIO_RF_I_DATA_6 = 0x05C
- MODE_GPIO_RF_I_DATA_7 = 0x05D
- MODE_GPIO_RF_Q_DATA_0 = 0x05E
- MODE_GPIO_RF_Q_DATA_1 = 0x05F
- MODE_GPIO_RF_Q_DATA_2 = 0x060
- MODE_GPIO_RF_Q_DATA_3 = 0x061
- MODE_GPIO_RF_Q_DATA_4 = 0x062
- MODE_GPIO_RF_Q_DATA_5 = 0x063
- MODE_GPIO_RF_Q_DATA_6 = 0x064
- MODE_GPIO_RF_Q_DATA_7 = 0x065
- MODE_GPIO_RF_ANT_SW_0 = 0x066

- MODE_GPIO_RF_ANT_SW_1 = 0x067
- MODE_GPIO_RF_ANT_SW_2 = 0x068
- MODE_GPIO_RF_ANT_SW_3 = 0x069
- MODE_GPIO_TOF_START = 0x06A
- MODE_GPIO_TOF_STOP = 0x06B
- MODE_GPIO_PCM_SERI_IN = 0x100
- MODE_GPIO_PCM_FRAME_IN = 0x101
- MODE_GPIO_PCM_FRAME_OUT = 0x102
- MODE_GPIO_PCM_CLK_IN = 0x103
- MODE_GPIO_SPI0_CS_IN = 0x200
- MODE_GPIO_SPI0_CLK_IN = 0x201
- MODE_GPIO_SPI1_CS_IN = 0x202
- MODE_GPIO_SPI1_CLK_IN = 0x203
- MODE_GPIO_UART_RX_IN = 0x300
- MODE_GPIO_I2C0_SCL_IN = 0x400
- MODE_GPIO_I2C0_SDA_IN = 0x401
- MODE_GPIO_I2C1_SCL_IN = 0x402
- MODE_GPIO_I2C1_SDA_IN = 0x403
- MODE_GPIO_NMI_IN = 0x500
- MODE_GPIO_BB_RX_CLK_IN = 0x600
- MODE_GPIO_BB_RX_DATA_IN = 0x601
- MODE_GPIO_BB_SYNC_P_IN = 0x602
- MODE_GPIO_BB_SPI_MISO_IN = 0x603
- MODE_GPIO_RF_SPI_MOSI_IN = 0x700
- MODE_GPIO_RF_SPI_CSN_IN = 0x701
- MODE_GPIO_RF_SPI_CLK_IN = 0x702
- MODE_GPIO_RF_GPIO0_IN = 0x800
- MODE_GPIO_RF_GPIO1_IN = 0x801
- MODE_GPIO_RF_GPIO2_IN = 0x802
- MODE_GPIO_RF_GPIO3_IN = 0x803

- `MODE_GPIO_RF_GPIO4_IN = 0x804`
- `MODE_GPIO_RF_GPIO5_IN = 0x805`
- `MODE_GPIO_RF_GPIO6_IN = 0x806`
- `MODE_GPIO_RF_GPIO7_IN = 0x807`
- `MODE_GPIO_RF_GPIO8_IN = 0x808`
- `MODE_GPIO_RF_GPIO9_IN = 0x809`
- `MODE_GPIO_ADC_IN = 0x80A`

18.7.4.9 _GPIO_FUNC_REGISTERS_t

Location: Driver_GPIO.h:263

GPIO Control Codes: GPIO alternative function registers.

Members

- `GPIO_FUNC_REG_SPI0 = 0x00`

  SPI 0 register

- `GPIO_FUNC_REG_SPI1 = 0x01`

  SPI 1 register

- `GPIO_FUNC_REG_UART0 = 0x03`

  UART register

- `GPIO_FUNC_REG_I2C0 = 0x04`

  I2C 0 register

- `GPIO_FUNC_REG_I2C1 = 0x05`

  I2C 1 register

- `GPIO_FUNC_REG_PCM0 = 0x06`

  PCM register

- `GPIO_FUNC_REG_NMI = 0x07`

  NMI register

- `GPIO_FUNC_REG_BB_RX = 0x08`

  BB RX register

- `GPIO_FUNC_REG_BB_SPI = 0x09`

  BB SPI register

- `GPIO_FUNC_REG_RF_SPI = 0x0A`

  RF SPI register

- `GPIO_FUNC_REG_RF_GPIO03 = 0x0B`

  RF GPIO03 register

- `GPIO_FUNC_REG_RF_GPIO47 = 0x0C`

  RF GPIO47 register

- `GPIO_FUNC_REG_RF_GPIO89 = 0x0D`

  RF GPIO89 register

- `GPIO_FUNC_REG_JTAG_SW_PAD = 0x0E`

  JTAG SW pad register.

18.7.4.10 _GPIO_EN_DIS_t

Location: Driver_GPIO.h:281

GPIO Control Codes: Enable / Disable values.

Members

- `GPIO_DISABLE = 0`

  GPIO disable value.

- `GPIO_ENABLE = 1`

  GPIO enable value

18.7.4.11 _GPIO_EVENT_t

Location: Driver_GPIO.h:288

GPIO Control Codes: Interrupts events.

Members

- `GPIO_IN_EVENT_NONE = 0`

Interrupt event none

- `GPIO_IN_EVENT_HIGH_LEVEL = 1`

Interrupt event high level

- `GPIO_IN_EVENT_LOW_LEVEL = 2`

Interrupt event low level

- `GPIO_IN_EVENT_RISING_EDGE = 3`

Interrupt event rising edge

- `GPIO_IN_EVENT_FALLING_EDGE = 4`

Interrupt event falling edge.

- `GPIO_IN_EVENT_TRANSITION = 5`

Interrupt event transition

18.7.4.12 _GPIO_DBC_CLK_t

Location: Driver_GPIO.h:298

GPIO Control Codes: Debounce clock source.

Members

- `GPIO_DBC_CLK_SLOWCLK_DIV32 = 0`

Debounce clock source = slow clock / 32

- `GPIO_DBC_CLK_SLOWCLK_DIV1024 = 1`

  Debounce clock source = slow clock / 1024.

18.7.4.13 _GPIO_DRIVE_STRENGTHS_t

Location: Driver_GPIO.h:304

GPIO Control Codes: Pads strength.

Members

- `GPIO_LOW_DRIVE = 0`

  Regular drive strengths

- `GPIO_HIGH_DRIVE = 1`

  Drive strengths increased by ~50%.

18.7.5 CMSIS GPIO Driver Macro Definition Documentation

18.7.5.1 ARM_GPIO_API_VERSION

```
#define ARM_GPIO_API_VERSION ARM_DRIVER_VERSION_MAJOR_MINOR(1,0)
```

Location: Driver_GPIO.h:37

GPIO API version.

18.7.5.2 GPIO_EVENT_0_IRQ

```
#define GPIO_EVENT_0_IRQ (1UL << 0)
```

Location: Driver_GPIO.h:310

GPIO Event.

GPIO0 interrupt event value

18.7.5.3 GPIO_EVENT_1_IRQ

```
#define GPIO_EVENT_1_IRQ (1UL << 1)
```

Location: Driver_GPIO.h:311

GPIO1 interrupt event value.

18.7.5.4 GPIO_EVENT_2_IRQ

```
#define GPIO_EVENT_2_IRQ (1UL << 2)
```

Location: Driver_GPIO.h:312

GPIO2 interrupt event value.

18.7.5.5 GPIO_EVENT_3_IRQ

```
#define GPIO_EVENT_3_IRQ (1UL << 3)
```

Location: Driver_GPIO.h:313

GPIO3 interrupt event value.

18.7.6 CMSIS GPIO Driver Function Documentation

18.7.6.1 GPIO_GetVersion

```
ARM_DRIVER_VERSION GPIO_GetVersion()
```

Location: Driver_GPIO.c:21

Get driver version.

Returns:

ARM_DRIVER_VERSION

18.7.6.2 GPIO_Initialize

```
int32_t GPIO_Initialize(GPIO_SignalEvent_t cb)
```

Location: Driver_GPIO.c:22

Initialize the GPIO driver.

Returns:

execution_status

Parameters

Direction	Name	Description
in	*cb*	Pointer to GPIO_SignalEvent

18.7.6.3 GPIO_Configure

```
int32_t GPIO_Configure(const GPIO_CFG_t * cfg)
```

Location: Driver_GPIO.c:23

Configure common GPIO settings.

Returns:

execution_status

Parameters

Direction	Name	Description
in	*cfg*	Pointer to GPIO_CFG_t

18.7.6.4 GPIO_ConfigurePad

```
int32_t GPIO_ConfigurePad(GPIO_SEL_t sel, const GPIO_PAD_CFG_t * cfg)
```

Location: Driver_GPIO.c:24

Configure the GPIO pad.

Returns:

execution_status

Parameters

Direction	Name	Description
in	*sel*	Pad selection GPIO_SEL_t
in	*cfg*	Pointer to GPIO_PAD_CFG_t

18.7.6.5 GPIO_ConfigureInterrupt

```
int32_t GPIO_ConfigureInterrupt(GPIO_INT_SEL_t sel, const GPIO_INT_CFG_t * cfg)
```

Location: Driver_GPIO.c:25

Configure the GPIO interrupt.

Returns:

execution_status

Parameters

Direction	Name	Description
in	*sel*	Interrupt selection GPIO_INT_SEL_t
in	*cfg*	Pointer to GPIO_INT_CFG_t

18.7.6.6 GPIO_SetInterruptPriority

```
int32_t GPIO_SetInterruptPriority(GPIO_INT_SEL_t sel, const GPIO_PRI_CFG_t * cfg)
```

Location: Driver_GPIO.c:26

Configure GPIO interrupt priority.

Returns:

execution_status

Parameters

Direction	Name	Description
in	*sel*	Interrupt selection GPIO_INT_SEL_t
in	*cfg*	Pointer to GPIO_PRI_CFG_t

18.7.6.7 GPIO_ConfigureJTAG

```
int32_t GPIO_ConfigureJTAG(const GPIO_JTAG_SW_CFG_t * cfg)
```

Location: Driver_GPIO.c:27

Configure the GPIO JTAG mode.

Returns:

execution_status

Parameters

Direction	Name	Description
in	*cfg*	Pointer to GPIO_JTAG_SW_CFG_t

18.7.6.8 GPIO_SetHigh

```
void GPIO_SetHigh(GPIO_SEL_t sel)
```

Location: Driver_GPIO.c:29

Set particular GPIO pad.

Returns:

None

Parameters

Direction	Name	Description
in	*sel*	Pad selection GPIO_SEL_t

18.7.6.9 GPIO_ToggleValue

```
void GPIO_ToggleValue(GPIO_SEL_t sel)
```

Location: Driver_GPIO.c:30

Toggle particular GPIO pad.

Returns:

None

Parameters

Direction	Name	Description
in	*sel*	Pad selection GPIO_SEL_t

18.7.6.10 GPIO_SetLow

```
void GPIO_SetLow(GPIO_SEL_t sel)
```

Location: Driver_GPIO.c:31

Reset particular GPIO pad.

Returns:

None

Parameters

Direction	Name	Description
in	*sel*	Pad selection GPIO_SEL_t

18.7.6.11 GPIO_ReadValue

```
uint32_t GPIO_ReadValue(GPIO_SEL_t sel)
```

Location: Driver_GPIO.c:32

Returns the selected GPIO pad value.

Returns:

GPIO pad value

Parameters

Direction	Name	Description
in	*sel*	Pad selection GPIO_SEL_t

18.7.6.12 GPIO_ResetAltFuncRegister

```
int32_t GPIO_ResetAltFuncRegister(GPIO_FUNC_REGISTERS_t reg)
```

Location: Driver_GPIO.c:33

Reset the particular alternative function register.

Returns:

execution_status

Parameters

Direction	Name	Description
in	*reg*	Register selection GPIO_FUNC_REGISTERS_t

18.7.6.13 GPIO_SignalEvent

```
void GPIO_SignalEvent(uint32_t event)
```

Location: Driver_GPIO.c:35

Signal GPIO events.

Returns:

None

Parameters

Direction	Name	Description
in	*event*	notification mask

18.8 CMSIS I2C DRIVER

CMSIS I^{2}C Driver Reference.

18.8.1 Summary

Typedefs

- ARM_I2C_STATUS : I2C Status.
- ARM_I2C_SignalEvent_t : Pointer to ARM_I2C_SignalEvent : Signal I2C Event.
- ARM_I2C_CAPABILITIES : I2C Driver Capabilities.
- ARM_DRIVER_I2C : Access structure of the I2C Driver.

Data Structures

- _ARM_I2C_STATUS : I2C Status.
- _ARM_I2C_CAPABILITIES : I2C Driver Capabilities.
- _ARM_DRIVER_I2C : Access structure of the I2C Driver.

Macros

- ARM_I2C_API_VERSION : I2C API version.
- ARM_I2C_OWN_ADDRESS : Set Own Slave Address; arg = address.
- ARM_I2C_BUS_SPEED : Set Bus Speed; arg = speed.
- ARM_I2C_BUS_CLEAR : Execute Bus clear: send nine clock pulses.
- ARM_I2C_ABORT_TRANSFER : Abort Master/Slave Transmit/Receive.
- ARM_I2C_BUS_SPEED_STANDARD : Standard Speed (100kHz)
- ARM_I2C_BUS_SPEED_FAST : Fast Speed (400kHz)

- ARM_I2C_BUS_SPEED_FAST_PLUS : Fast+ Speed (1MHz)
- ARM_I2C_BUS_SPEED_HIGH : High Speed (3.4MHz)
- ARM_I2C_ADDRESS_10BIT : 10-bit address flag
- ARM_I2C_ADDRESS_GC : General Call flag.
- ARM_I2C_EVENT_TRANSFER_DONE : I2C Event
- ARM_I2C_EVENT_TRANSFER_INCOMPLETE : Master/Slave Transmit/Receive incomplete transfer.
- ARM_I2C_EVENT_SLAVE_TRANSMIT : Slave Transmit operation requested.
- ARM_I2C_EVENT_SLAVE_RECEIVE : Slave Receive operation requested.
- ARM_I2C_EVENT_ADDRESS_NACK : Address not acknowledged from Slave.
- ARM_I2C_EVENT_GENERAL_CALL : General Call indication.
- ARM_I2C_EVENT_ARBITRATION_LOST : Master lost arbitration.
- ARM_I2C_EVENT_BUS_ERROR : Bus error detected (START/STOP at illegal position)
- ARM_I2C_EVENT_BUS_CLEAR : Bus clear finished.

Functions

- ARM_I2C_GetVersion : Get driver version.
- ARM_I2C_GetCapabilities : Get driver capabilities.
- ARM_I2C_Initialize : Initialize I2C Interface.
- ARM_I2C_Uninitialize : De-initialize I2C Interface.
- ARM_I2C_PowerControl : Control I2C Interface Power.
- ARM_I2C_MasterTransmit : Start transmitting data as I2C Master.
- ARM_I2C_MasterReceive : Start receiving data as I2C Master.
- ARM_I2C_SlaveTransmit : Start transmitting data as I2C Slave.
- ARM_I2C_SlaveReceive : Start receiving data as I2C Slave.
- ARM_I2C_GetDataCount : Get transferred data count.
- ARM_I2C_Control : Control I2C Interface.
- ARM_I2C_GetStatus : Get I2C status.
- ARM_I2C_SignalEvent : Signal I2C Events.

18.8.2 CMSIS I2C Driver Typedef Documentation

18.8.2.1 ARM_I2C_STATUS

```
typedef struct _ARM_I2C_STATUS ARM_I2C_STATUS
```

Location: Driver_I2C.h:112

I2C Status.

18.8.2.2 ARM_I2C_SignalEvent_t

```
typedef void(* ARM_I2C_SignalEvent_t
```

Location: Driver_I2C.h:198

Pointer to ARM_I2C_SignalEvent : Signal I2C Event.

18.8.2.3 ARM_I2C_CAPABILITIES

```
typedef struct _ARM_I2C_CAPABILITIES ARM_I2C_CAPABILITIES
```

Location: Driver_I2C.h:207

I2C Driver Capabilities.

18.8.2.4 ARM_DRIVER_I2C

```
typedef struct _ARM_DRIVER_I2C ARM_DRIVER_I2C
```

Location: Driver_I2C.h:226

Access structure of the I2C Driver.

18.8.3 CMSIS I2C Driver Data Structures Type Documentation

18.8.3.1 _ARM_I2C_STATUS

Location: Driver_I2C.h:104

I2C Status.

Data Fields

Type	*Name*	Description
uint32_t	*busy*	Busy flag.
uint32_t	*mode*	Mode: 0=Slave, 1=Master.
uint32_t	*direction*	Direction: 0=Transmitter, 1=Receiver.

uint32_t	*general_call*	General Call indication (cleared on start of next Slave operation)
uint32_t	*arbitration_lost*	Master lost arbitration (cleared on start of next Master operation)
uint32_t	*bus_error*	Bus error detected (cleared on start of next Master/Slave operation)
uint32_t	*reserved*	(Reserved for future use)

18.8.3.2 _ARM_I2C_CAPABILITIES

Location: Driver_I2C.h:204

I2C Driver Capabilities.

Data Fields

Type	*Name*	Description
uint32_t	*address_10_bit*	Supports 10-bit addressing.
uint32_t	*reserved*	Reserved (must be zero)

18.8.3.3 _ARM_DRIVER_I2C

Location: Driver_I2C.h:213

Access structure of the I2C Driver.

Data Fields

Type	*Name*	Description
ARM_DRIVER_VERSION(*	*GetVersion)(void)*	Pointer to ARM_I2C_GetVersion : Get driver version.
ARM_I2C_CAPABILITIES(*	*GetCapabilities)(void)*	Pointer to ARM_I2C_GetCapabilities : Get driver capabilities.

int32_t(*	*Initialize)(ARM_I2C_SignalEvent_t cb_event)*	Pointer to ARM_I2C_Initialize : Initialize I2C Interface.
int32_t(*	*Uninitialize)(void)*	Pointer to ARM_I2C_Uninitialize : De-initialize I2C Interface.
int32_t(*	*PowerControl)(ARM_POWER_STATE state)*	Pointer to ARM_I2C_PowerControl : Control I2C Interface Power.
int32_t(*	*MasterTransmit)(uint32_t addr, const uint8_t *data, uint32_t num, bool xfer_pending)*	Pointer to ARM_I2C_MasterTransmit : Start transmitting data as I2C Master.
int32_t(*	*MasterReceive)(uint32_t addr, uint8_t *data, uint32_t num, bool xfer_pending)*	Pointer to ARM_I2C_MasterReceive : Start receiving data as I2C Master.
int32_t(*	*SlaveTransmit)(const uint8_t *data, uint32_t num)*	Pointer to ARM_I2C_SlaveTransmit : Start transmitting data as I2C Slave.
int32_t(*	*SlaveReceive)(uint8_t *data, uint32_t num)*	Pointer to ARM_I2C_SlaveReceive : Start receiving data as I2C Slave.
int32_t(*	*GetDataCount)(void)*	Pointer to ARM_I2C_GetDataCount : Get transferred data count.
int32_t(*	*Control)(uint32_t control, uint32_t arg)*	Pointer to ARM_I2C_Control : Control I2C Interface.
ARM_I2C_STATUS(*	*GetStatus)(void)*	Pointer to ARM_I2C_GetStatus : Get I2C status.

18.8.4 CMSIS I2C Driver Macro Definition Documentation

18.8.4.1 ARM_I2C_API_VERSION

```
#define ARM_I2C_API_VERSION ARM_DRIVER_VERSION_MAJOR_MINOR(2,3)
```

Location: Driver_I2C.h:78

I2C API version.

18.8.4.2 ARM_I2C_OWN_ADDRESS

```
#define ARM_I2C_OWN_ADDRESS (0x01)
```

Location: Driver_I2C.h:83

Set Own Slave Address; arg = address.

18.8.4.3 ARM_I2C_BUS_SPEED

```
#define ARM_I2C_BUS_SPEED (0x02)
```

Location: Driver_I2C.h:84

Set Bus Speed; arg = speed.

18.8.4.4 ARM_I2C_BUS_CLEAR

```
#define ARM_I2C_BUS_CLEAR (0x03)
```

Location: Driver_I2C.h:85

Execute Bus clear: send nine clock pulses.

18.8.4.5 ARM_I2C_ABORT_TRANSFER

```
#define ARM_I2C_ABORT_TRANSFER (0x04)
```

Location: Driver_I2C.h:86

Abort Master/Slave Transmit/Receive.

18.8.4.6 ARM_I2C_BUS_SPEED_STANDARD

```
#define ARM_I2C_BUS_SPEED_STANDARD (0x01)
```

Location: Driver_I2C.h:89

Standard Speed (100kHz)

18.8.4.7 ARM_I2C_BUS_SPEED_FAST

```
#define ARM_I2C_BUS_SPEED_FAST (0x02)
```

Location: Driver_I2C.h:90

Fast Speed (400kHz)

18.8.4.8 ARM_I2C_BUS_SPEED_FAST_PLUS

```
#define ARM_I2C_BUS_SPEED_FAST_PLUS (0x03)
```

Location: Driver_I2C.h:91

Fast+ Speed (1MHz)

18.8.4.9 ARM_I2C_BUS_SPEED_HIGH

```
#define ARM_I2C_BUS_SPEED_HIGH (0x04)
```

Location: Driver_I2C.h:92

High Speed (3.4MHz)

18.8.4.10 ARM_I2C_ADDRESS_10BIT

```
#define ARM_I2C_ADDRESS_10BIT (0x0400)
```

Location: Driver_I2C.h:97

10-bit address flag

18.8.4.11 ARM_I2C_ADDRESS_GC

```
#define ARM_I2C_ADDRESS_GC (0x8000)
```

Location: Driver_I2C.h:98

General Call flag.

18.8.4.12 ARM_I2C_EVENT_TRANSFER_DONE

```
#define ARM_I2C_EVENT_TRANSFER_DONE (1UL << 0)
```

Location: Driver_I2C.h:116

I2C Event

Master/Slave Transmit/Receive finished

18.8.4.13 ARM_I2C_EVENT_TRANSFER_INCOMPLETE

```
#define ARM_I2C_EVENT_TRANSFER_INCOMPLETE (1UL << 1)
```

Location: Driver_I2C.h:117

Master/Slave Transmit/Receive incomplete transfer.

18.8.4.14 ARM_I2C_EVENT_SLAVE_TRANSMIT

```
#define ARM_I2C_EVENT_SLAVE_TRANSMIT (1UL << 2)
```

Location: Driver_I2C.h:118

Slave Transmit operation requested.

18.8.4.15 ARM_I2C_EVENT_SLAVE_RECEIVE

```
#define ARM_I2C_EVENT_SLAVE_RECEIVE (1UL << 3)
```

Location: Driver_I2C.h:119

Slave Receive operation requested.

18.8.4.16 ARM_I2C_EVENT_ADDRESS_NACK

```
#define ARM_I2C_EVENT_ADDRESS_NACK (1UL << 4)
```

Location: Driver_I2C.h:120

Address not acknowledged from Slave.

18.8.4.17 ARM_I2C_EVENT_GENERAL_CALL

```
#define ARM_I2C_EVENT_GENERAL_CALL (1UL << 5)
```

Location: Driver_I2C.h:121

General Call indication.

18.8.4.18 ARM_I2C_EVENT_ARBITRATION_LOST

```
#define ARM_I2C_EVENT_ARBITRATION_LOST (1UL << 6)
```

Location: Driver_I2C.h:122

Master lost arbitration.

18.8.4.19 ARM_I2C_EVENT_BUS_ERROR

```
#define ARM_I2C_EVENT_BUS_ERROR (1UL << 7)
```

Location: Driver_I2C.h:123

Bus error detected (START/STOP at illegal position)

18.8.4.20 ARM_I2C_EVENT_BUS_CLEAR

```
#define ARM_I2C_EVENT_BUS_CLEAR (1UL << 8)
```

Location: Driver_I2C.h:124

Bus clear finished.

18.8.5 CMSIS I2C Driver Function Documentation

18.8.5.1 ARM_I2C_GetVersion

```
ARM_DRIVER_VERSION ARM_I2C_GetVersion()
```

Location: Driver_I2C.c:38

Get driver version.

Returns:

ARM_DRIVER_VERSION

18.8.5.2 ARM_I2C_GetCapabilities

```
ARM_I2C_CAPABILITIES ARM_I2C_GetCapabilities()
```

Location: Driver_I2C.c:43

Get driver capabilities.

Returns:

ARM_I2C_CAPABILITIES

18.8.5.3 ARM_I2C_Initialize

```
int32_t ARM_I2C_Initialize(ARM_I2C_SignalEvent_t cb_event)
```

Location: Driver_I2C.c:48

Initialize I2C Interface.

Returns:

execution_status

Parameters

Direction	Name	Description
in	*cb_event*	Pointer to ARM_I2C_SignalEvent

18.8.5.4 ARM_I2C_Uninitialize

```
int32_t ARM_I2C_Uninitialize()
```

Location: Driver_I2C.c:52

De-initialize I2C Interface.

Returns:

execution_status

18.8.5.5 ARM_I2C_PowerControl

```
int32_t ARM_I2C_PowerControl(ARM_POWER_STATE state)
```

Location: Driver_I2C.c:56

Control I2C Interface Power.

Returns:

execution_status

Parameters

Direction	Name	Description
in	*state*	Power state

18.8.5.6 ARM_I2C_MasterTransmit

```
int32_t ARM_I2C_MasterTransmit(uint32_t addr, const uint8_t * data, uint32_t num, bool
xfer_pending)
```

Location: Driver_I2C.c:72

Start transmitting data as I2C Master.

Returns:

execution_status

Parameters

Direction	Name	Description
in	*addr*	Slave address (7-bit or 10-bit)
in	*data*	Pointer to buffer with data to transmit to I2C Slave
in	*num*	Number of data bytes to transmit
in	*xfer_pending*	Transfer operation is pending - Stop condition will not be generated

18.8.5.7 ARM_I2C_MasterReceive

```
int32_t ARM_I2C_MasterReceive(uint32_t addr, uint8_t * data, uint32_t num, bool xfer_
pending)
```

Location: Driver_I2C.c:76

Start receiving data as I2C Master.

Returns:

execution_status

Parameters

Direction	Name	Description
in	*addr*	Slave address (7-bit or 10-bit)
out	*data*	Pointer to buffer for data to receive from I2C Slave
in	*num*	Number of data bytes to receive
in	*xfer_pending*	Transfer operation is pending - Stop condition will not be generated

18.8.5.8 ARM_I2C_SlaveTransmit

```
int32_t ARM_I2C_SlaveTransmit(const uint8_t * data, uint32_t num)
```

Location: Driver_I2C.c:80

Start transmitting data as I2C Slave.

Returns:

execution_status

Parameters

Direction	Name	Description
in	*data*	Pointer to buffer with data to transmit to I2C Master
in	*num*	Number of data bytes to transmit

18.8.5.9 ARM_I2C_SlaveReceive

```
int32_t ARM_I2C_SlaveReceive(uint8_t * data, uint32_t num)
```

Location: Driver_I2C.c:84

Start receiving data as I2C Slave.

Returns:

execution_status

Parameters

Direction	Name	Description
`out`	*data*	Pointer to buffer for data to receive from I2C Master
`in`	*num*	Number of data bytes to receive

18.8.5.10 ARM_I2C_GetDataCount

```
int32_t ARM_I2C_GetDataCount()
```

Location: Driver_I2C.c:88

Get transferred data count.

Returns:

number of data bytes transferred; -1 when Slave is not addressed by Master

18.8.5.11 ARM_I2C_Control

```
int32_t ARM_I2C_Control(uint32_t control, uint32_t arg)
```

Location: Driver_I2C.c:92

Control I2C Interface.

Returns:

execution_status

Parameters

Direction	Name	Description
in	*control*	Operation
in	*arg*	Argument of operation (optional)

18.8.5.12 ARM_I2C_GetStatus

```
ARM_I2C_STATUS ARM_I2C_GetStatus()
```

Location: Driver_I2C.c:124

Get I2C status.

Returns:

I2C status ARM_I2C_STATUS

18.8.5.13 ARM_I2C_SignalEvent

```
void ARM_I2C_SignalEvent(uint32_t event)
```

Location: Driver_I2C.c:128

Signal I2C Events.

Parameters

Direction	Name	Description
in	*event*	I2C_events notification mask

18.9 CMSIS PWM Driver

CMSIS PWM Driver Reference.

18.9.1 Summary

Typedefs

- PWM_SEL_t : PWM Control Codes: PWM Selection.
- PWM_CFG_t : PWM Driver configuration.
- DRIVER_PWM_t : Access structure of the PWM Driver.

Data Structures

- _PWM_CFG_t : PWM Driver configuration.
- _DRIVER_PWM_t : Access structure of the PWM Driver.

Enumerations

- _PWM_SEL_t : PWM Control Codes: PWM Selection.

Macros

- ARM_PWM_API_VERSION : PWM API version
- PWM_ERROR_UNCONFIGURED : Driver has not been configured yet.

Functions

- PWM_GetVersion : Get driver version.
- PWM_Initialize : Initialize PWM driver with default configuration.
- PWM_Configure : Configure PWM common setting.
- PWM_SelectClock : Select the PWM clock source.
- PWM_Reset : Reset the PWM.
- PWM_SetDithering : Set the PWM dithering.
- PWM_SetPeriod : Set the PWM period.
- PWM_SetDutyCycle : Set the PWM duty cycle (expressed in percentage)
- PWM_SetHighPeriod : Set the PWM duty cycle (expressed in cycles).
- PWM_SetOffset : Set the offset between PWM.
- PWM_Start : Start the PWM.
- PWM_Stop : Stop the PWM.

18.9.2 CMSIS PWM Driver Typedef Documentation

18.9.2.1 PWM_SEL_t

```
typedef enum _PWM_SEL_t PWM_SEL_t
```

Location: Driver_PWM.h:48

PWM Control Codes: PWM Selection.

18.9.2.2 PWM_CFG_t

```
typedef struct _PWM_CFG_t PWM_CFG_t
```

Location: Driver_PWM.h:131

PWM Driver configuration.

18.9.2.3 DRIVER_PWM_t

```
typedef struct _DRIVER_PWM_t DRIVER_PWM_t
```

Location: Driver_PWM.h:149

Access structure of the PWM Driver.

18.9.3 CMSIS PWM Driver Data Structures Type Documentation

18.9.3.1 _PWM_CFG_t

Location: Driver_PWM.h:125

PWM Driver configuration.

Data Fields

Type	*Name*	Description
uint32_t	*period*	Period value
uint32_t	*high_cycle*	High cycle value.
uint32_t	*dithering*	Dithering value
uint32_t	*offset*	Offset value

18.9.3.2 _DRIVER_PWM_t

Location: Driver_PWM.h:136

Access structure of the PWM Driver.

Data Fields

Type	*Name*	Description
ARM_DRIVER_VERSION(*	*GetVersion)(void)*	Pointer to PWM_GetVersion : Get driver version.
int32_t(*	*Initialize)(void)*	Pointer to PWM_Initialize : Initialize PWM driver.
int32_t(*	*Configure)(PWM_SEL_t sel, const PWM_CFG_t *pwm_cfg)*	Pointer to PWM_Configure : Configure PWM common setting.
int32_t(*	*SelectClock)(uint8_t clock_src, uint8_t slowclk_prescale)*	Pointer to PWM_SelectClock : Configure PWM clock source.
int32_t(*	*Reset)(PWM_SEL_t sel)*	Pointer to PWM_Reset : Reset PWM
int32_t(*	*SetDithering)(PWM_SEL_t sel, uint8_t dithering)*	Pointer to PWM_SetDithering : Add dithering to PWM channel.
int32_t(*	*SetPeriod)(PWM_SEL_t sel, uint16_t period)*	Pointer to PWM_SetPeriod : Set the period.
int32_t(*	*SetDutyCycle)(PWM_SEL_t sel, uint8_t duty_cycle)*	Pointer to PWM_SetDutyCycle : Set the duty cycle (percentage).

`int32_t(*`	*SetHighPeriod)(PWM_SEL_t sel, uint16_t high_period)*	Pointer to PWM_SetHighPeriod : Set the high cycle period.
`int32_t(*`	*SetOffset)(PWM_SEL_t sel, uint16_t offset)*	Pointer to PWM_SetOffset : Set offset between PWM.
`int32_t(*`	*Start)(PWM_SEL_t sel)*	Pointer to PWM_Start : Start the PWM.
`int32_t(*`	*Stop)(PWM_SEL_t sel)*	Pointer to PWM_Stop : Stop the PWM.

18.9.4 CMSIS PWM Driver Enumeration Type Documentation

18.9.4.1 _PWM_SEL_t

Location: Driver_PWM.h:42

PWM Control Codes: PWM Selection.

Members

- `PWM_0 = 0`

  PWM module 0.

- `PWM_1 = 1`

  PWM module 1.

- `PWM_2 = 2`

  PWM module 2.

- `PWM_3 = 3`

  PWM module 3.

- `PWM_4 = 4`

  PWM module 4.

18.9.5 CMSIS PWM Driver Macro Definition Documentation

18.9.5.1 ARM_PWM_API_VERSION

```
#define ARM_PWM_API_VERSION ARM_DRIVER_VERSION_MAJOR_MINOR(1,0)
```

Location: Driver_PWM.h:37

PWM API version

18.9.5.2 PWM_ERROR_UNCONFIGURED

```
#define PWM_ERROR_UNCONFIGURED (ARM_DRIVER_ERROR_SPECIFIC - 1)
```

Location: Driver_PWM.h:51

Driver has not been configured yet.

18.9.6 CMSIS PWM Driver Function Documentation

18.9.6.1 PWM_GetVersion

```
ARM_DRIVER_VERSION PWM_GetVersion()
```

Location: Driver_PWM.c:21

Get driver version.

Returns:

ARM_DRIVER_VERSION

18.9.6.2 PWM_Initialize

```
int32_t PWM_Initialize()
```

Location: Driver_PWM.c:22

Initialize PWM driver with default configuration.

Returns:

execution_status

18.9.6.3 PWM_Configure

```
int32_t PWM_Configure(PWM_SEL_t sel, const PWM_CFG_t * pwm_cfg)
```

Location: Driver_PWM.c:23

Configure PWM common setting.

Returns:

execution_status

Parameters

Direction	Name	Description
in	*sel*	PWM to be configured PWM_SEL_t
in	*pwm_cfg*	Pointer to PWM_CFG_t

18.9.6.4 PWM_SelectClock

```
int32_t PWM_SelectClock(uint8_t clock_src, uint8_t slowclk_prescale)
```

Location: Driver_PWM.c:24

Select the PWM clock source.

Returns:

execution_status

Parameters

Direction	Name	Description
in	*clock_src*	Clock source for the PWM block
in	*slowclk_prescale*	Prescale to divide SYSCLK into SLOWCLK

18.9.6.5 PWM_Reset

```
int32_t PWM_Reset(PWM_SEL_t sel)
```

Location: Driver_PWM.c:25

Reset the PWM.

Returns:

execution_status

Parameters

Direction	Name	Description
in	*sel*	PWM to be reset PWM_SEL_t

18.9.6.6 PWM_SetDithering

```
int32_t PWM_SetDithering(PWM_SEL_t sel, uint8_t dithering)
```

Location: Driver_PWM.c:26

Set the PWM dithering.

Returns:

execution_status

Parameters

Direction	Name	Description
in	*sel*	PWM to be configured PWM_SEL_t
in	*dithering*	PWM dithering value

18.9.6.7 PWM_SetPeriod

```
int32_t PWM_SetPeriod(PWM_SEL_t sel, uint16_t period)
```

Location: Driver_PWM.c:27

Set the PWM period.

Returns:

execution_status

Parameters

Direction	Name	Description
in	*sel*	PWM to be configured PWM_SEL_t
in	*period*	Period value

18.9.6.8 PWM_SetDutyCycle

```
int32_t PWM_SetDutyCycle(PWM_SEL_t sel, uint8_t duty_cycle)
```

Location: Driver_PWM.c:28

Set the PWM duty cycle (expressed in percentage)

Returns:

execution_status

Parameters

Direction	Name	Description
in	*sel*	PWM to be configured PWM_SEL_t
in	*duty_cycle*	Duty cycle value (expressed in percentage)

18.9.6.9 PWM_SetHighPeriod

```
int32_t PWM_SetHighPeriod(PWM_SEL_t sel, uint16_t high_period)
```

Location: Driver_PWM.c:29

Set the PWM duty cycle (expressed in cycles).

Returns:

execution_status

Parameters

Direction	Name	Description
in	*sel*	PWM to be configured PWM_SEL_t
in	*high_period*	Duty cycle value (expressed in cycles)

18.9.6.10 PWM_SetOffset

```
int32_t PWM_SetOffset(PWM_SEL_t sel, uint16_t offset)
```

Location: Driver_PWM.c:30

Set the offset between PWM.

Returns:

execution_status

Parameters

Direction	Name	Description
in	*sel*	PWM to be configured PWM_SEL_t
in	*offset*	Offset between PWM

18.9.6.11 PWM_Start

```
int32_t PWM_Start(PWM_SEL_t sel)
```

Location: Driver_PWM.c:31

Start the PWM.

Returns:

execution_status

Parameters

Direction	Name	Description
in	*sel*	PWM to be started PWM_SEL_t

18.9.6.12 PWM_Stop

```
int32_t PWM_Stop(PWM_SEL_t sel)
```

Location: Driver_PWM.c:32

Stop the PWM.

Returns:

execution_status

Parameters

Direction	Name	Description
in	*sel*	PWM to be stopped PWM_SEL_t

18.10 CMSIS SPI Driver

CMSIS SPI Driver Reference.

18.10.1 Summary

Typedefs

- ARM_SPI_STATUS : SPI Status .
- ARM_SPI_SignalEvent_t : Pointer to ARM_SPI_SignalEvent : Signal SPI Event.

- ARM_SPI_CAPABILITIES : SPI Driver Capabilities.
- ARM_DRIVER_SPI : Access structure of the SPI Driver.

Data Structures

- _ARM_SPI_STATUS : SPI Status .
- _ARM_SPI_CAPABILITIES : SPI Driver Capabilities.
- _ARM_DRIVER_SPI : Access structure of the SPI Driver.

Macros

- ARM_SPI_API_VERSION : SPI API version.
- ARM_SPI_CONTROL_Pos : Position of the 0th bit of the SPI Control field in the ARM_SPI structure.
- ARM_SPI_CONTROL_Msk : Positioning of SPI Control field in the ARM_SPI structure.
- ARM_SPI_MODE_INACTIVE : SPI Inactive.
- ARM_SPI_MODE_MASTER : SPI Master (Output on MOSI, Input on MISO); arg = Bus Speed in bps.
- ARM_SPI_MODE_SLAVE : SPI Slave (Output on MISO, Input on MOSI).
- ARM_SPI_MODE_MASTER_SIMPLEX : SPI Master (Output/Input on MOSI); arg = Bus Speed in bps.
- ARM_SPI_MODE_SLAVE_SIMPLEX : SPI Slave (Output/Input on MISO).
- ARM_SPI_FRAME_FORMAT_Pos : Position of the 0th bit of the Frame format field in the ARM_SPI structure.
- ARM_SPI_FRAME_FORMAT_Msk : Positioning of Frame format field in the ARM_SPI structure.
- ARM_SPI_CPOL0_CPHA0 : Clock Polarity 0, Clock Phase 0 (default).
- ARM_SPI_CPOL0_CPHA1 : Clock Polarity 0, Clock Phase 1.
- ARM_SPI_CPOL1_CPHA0 : Clock Polarity 1, Clock Phase 0.
- ARM_SPI_CPOL1_CPHA1 : Clock Polarity 1, Clock Phase 1.
- ARM_SPI_TI_SSI : Texas Instruments Frame Format.
- ARM_SPI_MICROWIRE : National Microwire Frame Format.
- ARM_SPI_DATA_BITS_Pos : Position of the 0th bit of the Data bits field in the ARM_SPI structure.
- ARM_SPI_DATA_BITS_Msk : Positioning of the Data bits field in the ARM_SPI structure.
- ARM_SPI_DATA_BITS : Number of Data bits.
- ARM_SPI_BIT_ORDER_Pos : Position of the 0th bit of the Bit order field in the ARM_SPI structure.
- ARM_SPI_BIT_ORDER_Msk : Positioning of the Bit order field in the ARM_SPI structure.
- ARM_SPI_MSB_LSB : SPI Bit order from MSB to LSB (default).
- ARM_SPI_LSB_MSB : SPI Bit order from LSB to MSB.
- ARM_SPI_SS_MASTER_MODE_Pos : Position of the 0th bit of the Slave Select Master Mode field in the ARM_SPI structure.
- ARM_SPI_SS_MASTER_MODE_Msk : Positioning of the Slave Select Master Mode field in the ARM_SPI structure.
- ARM_SPI_SS_MASTER_UNUSED : SPI Slave Select when Master: Not used (default).
- ARM_SPI_SS_MASTER_SW : SPI Slave Select when Master: Software controlled.
- ARM_SPI_SS_MASTER_HW_OUTPUT : SPI Slave Select when Master: Hardware controlled Output.
- ARM_SPI_SS_MASTER_HW_INPUT : SPI Slave Select when Master: Hardware monitored Input.
- ARM_SPI_SS_SLAVE_MODE_Pos : Position of the 0th bit of the Slave Select Slave Mode field in the ARM_SPI structure.
- ARM_SPI_SS_SLAVE_MODE_Msk : Positioning of the Slave Select Slave mode field in the ARM_SPI structure.

- ARM_SPI_SS_SLAVE_HW : SPI Slave Select when Slave: Hardware monitored (default).
- ARM_SPI_SS_SLAVE_SW : SPI Slave Select when Slave: Software controlled.
- ARM_SPI_SET_BUS_SPEED : Set Bus Speed in bps; arg = value.
- ARM_SPI_GET_BUS_SPEED : Get Bus Speed in bps.
- ARM_SPI_SET_DEFAULT_TX_VALUE : Set default Transmit value; arg = value.
- ARM_SPI_CONTROL_SS : Control Slave Select; arg: 0=inactive, 1=active.
- ARM_SPI_ABORT_TRANSFER : Abort current data transfer.
- ARM_SPI_SS_INACTIVE : SPI Slave Select Signal Inactive.
- ARM_SPI_SS_ACTIVE : SPI Slave Select Signal Active.
- ARM_SPI_ERROR_MODE : Specified Mode not supported.
- ARM_SPI_ERROR_FRAME_FORMAT : Specified Frame Format not supported.
- ARM_SPI_ERROR_DATA_BITS : Specified number of Data bits not supported.
- ARM_SPI_ERROR_BIT_ORDER : Specified Bit order not supported.
- ARM_SPI_ERROR_SS_MODE : Specified Slave Select Mode not supported.
- ARM_SPI_EVENT_TRANSFER_COMPLETE : SPI Event
- ARM_SPI_EVENT_DATA_LOST : Data lost: Receive overflow / Transmit underflow.
- ARM_SPI_EVENT_MODE_FAULT : Master Mode Fault (SS deactivated when Master).

Functions

- ARM_SPI_GetVersion : Get driver version.
- ARM_SPI_GetCapabilities : Get driver capabilities.
- ARM_SPI_Initialize : Initialize SPI Interface.
- ARM_SPI_Uninitialize : De-initialize SPI Interface.
- ARM_SPI_PowerControl : Control SPI Interface Power.
- ARM_SPI_Send : Start sending data to SPI transmitter.
- ARM_SPI_Receive : Start receiving data from SPI receiver.
- ARM_SPI_Transfer : Start sending/receiving data to/from SPI transmitter/receiver.
- ARM_SPI_GetDataCount : Get transferred data count.
- ARM_SPI_Control : Control SPI Interface.
- ARM_SPI_GetStatus : Get SPI status.
- ARM_SPI_SignalEvent : Signal SPI Events.

18.10.2 CMSIS SPI Driver Typedef Documentation

18.10.2.1 ARM_SPI_STATUS

```
typedef struct _ARM_SPI_STATUS ARM_SPI_STATUS
```

Location: Driver_SPI.h:148

SPI Status .

18.10.2.2 ARM_SPI_SignalEvent_t

```
typedef void(* ARM_SPI_SignalEvent_t
```

Location: Driver_SPI.h:222

Pointer to ARM_SPI_SignalEvent : Signal SPI Event.

18.10.2.3 ARM_SPI_CAPABILITIES

```
typedef struct _ARM_SPI_CAPABILITIES ARM_SPI_CAPABILITIES
```

Location: Driver_SPI.h:234

SPI Driver Capabilities.

18.10.2.4 ARM_DRIVER_SPI

```
typedef struct _ARM_DRIVER_SPI ARM_DRIVER_SPI
```

Location: Driver_SPI.h:253

Access structure of the SPI Driver.

18.10.3 CMSIS SPI Driver Data Structures Type Documentation

18.10.3.1 _ARM_SPI_STATUS

Location: Driver_SPI.h:143

SPI Status .

Data Fields

Type	*Name*	Description
uint32_t	*busy*	Transmitter/Receiver busy flag.
uint32_t	*data_lost*	Data lost: Receive overflow / Transmit underflow (cleared on start of transfer operation).
uint32_t	*mode_fault*	Mode fault detected; optional (cleared on start of transfer operation).
uint32_t	*reserved*	(Reserved for future use)

18.10.3.2 _ARM_SPI_CAPABILITIES

Location: Driver_SPI.h:228

SPI Driver Capabilities.

Data Fields

Type	*Name*	Description
uint32_t	*simplex*	supports Simplex Mode (Master and Slave).
uint32_t	*ti_ssi*	supports TI Synchronous Serial Interface.
uint32_t	*microwire*	supports Microwire Interface.
uint32_t	*event_mode_fault*	Signal Mode Fault event: ARM_SPI_EVENT_MODE_FAULT.
uint32_t	*reserved*	Reserved (must be zero).

18.10.3.3 _ARM_DRIVER_SPI

Location: Driver_SPI.h:239

Access structure of the SPI Driver.

Data Fields

Type	Name	Description
ARM_DRIVER_VERSION(*	*GetVersion)(void)*	Pointer to ARM_SPI_GetVersion : Get driver version.
ARM_SPI_CAPABILITIES(*	*GetCapabilities)(void)*	Pointer to ARM_SPI_GetCapabilities : Get driver capabilities.
int32_t(*	*Initialize)(ARM_SPI_SignalEvent_t cb_event)*	Pointer to ARM_SPI_Initialize : Initialize SPI Interface.
int32_t(*	*Uninitialize)(void)*	Pointer to ARM_SPI_Uninitialize : De-initialize SPI Interface.
int32_t(*	*PowerControl)(ARM_POWER_STATE state)*	Pointer to ARM_SPI_PowerControl : Control SPI Interface Power.
int32_t(*	*Send)(const void *data, uint32_t num)*	Pointer to ARM_SPI_Send : Start sending data to SPI Interface.
int32_t(*	*Receive)(void *data, uint32_t num)*	Pointer to ARM_SPI_Receive : Start receiving data from SPI Interface.
int32_t(*	*Transfer)(const void *data_out, void *data_in, uint32_t num)*	Pointer to ARM_SPI_Transfer : Start sending/receiving data to/from SPI.
uint32_t(*	*GetDataCount)(void)*	Pointer to ARM_SPI_GetDataCount : Get transferred data count.
int32_t(*	*Control)(uint32_t control, uint32_t arg)*	Pointer to ARM_SPI_Control : Control SPI Interface.
ARM_SPI_STATUS(*	*GetStatus)(void)*	Pointer to ARM_SPI_GetStatus : Get SPI status.

18.10.4 CMSIS SPI Driver Macro Definition Documentation

18.10.4.1 ARM_SPI_API_VERSION

```
#define ARM_SPI_API_VERSION ARM_DRIVER_VERSION_MAJOR_MINOR(2,2)
```

Location: Driver_SPI.h:69

SPI API version.

18.10.4.2 ARM_SPI_CONTROL_Pos

```
#define ARM_SPI_CONTROL_Pos 0
```

Location: Driver_SPI.h:74

Position of the 0th bit of the SPI Control field in the ARM_SPI structure.

18.10.4.3 ARM_SPI_CONTROL_Msk

```
#define ARM_SPI_CONTROL_Msk (0xFFUL << ARM_SPI_CONTROL_Pos)
```

Location: Driver_SPI.h:75

Positioning of SPI Control field in the ARM_SPI structure.

18.10.4.4 ARM_SPI_MODE_INACTIVE

```
#define ARM_SPI_MODE_INACTIVE (0x00UL << ARM_SPI_CONTROL_Pos)
```

Location: Driver_SPI.h:78

SPI Inactive.

18.10.4.5 ARM_SPI_MODE_MASTER

```
#define ARM_SPI_MODE_MASTER (0x01UL << ARM_SPI_CONTROL_Pos)
```

Location: Driver_SPI.h:79

SPI Master (Output on MOSI, Input on MISO); arg = Bus Speed in bps.

18.10.4.6 ARM_SPI_MODE_SLAVE

```
#define ARM_SPI_MODE_SLAVE (0x02UL << ARM_SPI_CONTROL_Pos)
```

Location: Driver_SPI.h:80

SPI Slave (Output on MISO, Input on MOSI).

18.10.4.7 ARM_SPI_MODE_MASTER_SIMPLEX

```
#define ARM_SPI_MODE_MASTER_SIMPLEX (0x03UL << ARM_SPI_CONTROL_Pos)
```

Location: Driver_SPI.h:81

SPI Master (Output/Input on MOSI); arg = Bus Speed in bps.

18.10.4.8 ARM_SPI_MODE_SLAVE_SIMPLEX

```
#define ARM_SPI_MODE_SLAVE_SIMPLEX (0x04UL << ARM_SPI_CONTROL_Pos)
```

Location: Driver_SPI.h:82

SPI Slave (Output/Input on MISO).

18.10.4.9 ARM_SPI_FRAME_FORMAT_Pos

```
#define ARM_SPI_FRAME_FORMAT_Pos 8
```

Location: Driver_SPI.h:85

Position of the 0th bit of the Frame format field in the ARM_SPI structure.

18.10.4.10 ARM_SPI_FRAME_FORMAT_Msk

```
#define ARM_SPI_FRAME_FORMAT_Msk (7UL << ARM_SPI_FRAME_FORMAT_Pos)
```

Location: Driver_SPI.h:86

Positioning of Frame format field in the ARM_SPI structure.

18.10.4.11 ARM_SPI_CPOL0_CPHA0

```
#define ARM_SPI_CPOL0_CPHA0 (0UL << ARM_SPI_FRAME_FORMAT_Pos)
```

Location: Driver_SPI.h:87

Clock Polarity 0, Clock Phase 0 (default).

18.10.4.12 ARM_SPI_CPOL0_CPHA1

```
#define ARM_SPI_CPOL0_CPHA1 (1UL << ARM_SPI_FRAME_FORMAT_Pos)
```

Location: Driver_SPI.h:88

Clock Polarity 0, Clock Phase 1.

18.10.4.13 ARM_SPI_CPOL1_CPHA0

```
#define ARM_SPI_CPOL1_CPHA0 (2UL << ARM_SPI_FRAME_FORMAT_Pos)
```

Location: Driver_SPI.h:89

Clock Polarity 1, Clock Phase 0.

18.10.4.14 ARM_SPI_CPOL1_CPHA1

```
#define ARM_SPI_CPOL1_CPHA1 (3UL << ARM_SPI_FRAME_FORMAT_Pos)
```

Location: Driver_SPI.h:90

Clock Polarity 1, Clock Phase 1.

18.10.4.15 ARM_SPI_TI_SSI

```
#define ARM_SPI_TI_SSI (4UL << ARM_SPI_FRAME_FORMAT_Pos)
```

Location: Driver_SPI.h:91

Texas Instruments Frame Format.

18.10.4.16 ARM_SPI_MICROWIRE

```
#define ARM_SPI_MICROWIRE (5UL << ARM_SPI_FRAME_FORMAT_Pos)
```

Location: Driver_SPI.h:92

National Microwire Frame Format.

18.10.4.17 ARM_SPI_DATA_BITS_Pos

```
#define ARM_SPI_DATA_BITS_Pos 12
```

Location: Driver_SPI.h:95

Position of the 0th bit of the Data bits field in the ARM_SPI structure.

18.10.4.18 ARM_SPI_DATA_BITS_Msk

```
#define ARM_SPI_DATA_BITS_Msk (0x3FUL << ARM_SPI_DATA_BITS_Pos)
```

Location: Driver_SPI.h:96

Positioning of the Data bits field in the ARM_SPI structure.

18.10.4.19 ARM_SPI_DATA_BITS

```
#define ARM_SPI_DATA_BITS (((n) & 0x3F) << ARM_SPI_DATA_BITS_Pos)
```

Location: Driver_SPI.h:97

Number of Data bits.

18.10.4.20 ARM_SPI_BIT_ORDER_Pos

```
#define ARM_SPI_BIT_ORDER_Pos 18
```

Location: Driver_SPI.h:100

Position of the 0th bit of the Bit order field in the ARM_SPI structure.

18.10.4.21 ARM_SPI_BIT_ORDER_Msk

```
#define ARM_SPI_BIT_ORDER_Msk (1UL << ARM_SPI_BIT_ORDER_Pos)
```

Location: Driver_SPI.h:101

Positioning of the Bit order field in the ARM_SPI structure.

18.10.4.22 ARM_SPI_MSB_LSB

```
#define ARM_SPI_MSB_LSB (0UL << ARM_SPI_BIT_ORDER_Pos)
```

Location: Driver_SPI.h:102

SPI Bit order from MSB to LSB (default).

18.10.4.23 ARM_SPI_LSB_MSB

```
#define ARM_SPI_LSB_MSB (1UL << ARM_SPI_BIT_ORDER_Pos)
```

Location: Driver_SPI.h:103

SPI Bit order from LSB to MSB.

18.10.4.24 ARM_SPI_SS_MASTER_MODE_Pos

```
#define ARM_SPI_SS_MASTER_MODE_Pos 19
```

Location: Driver_SPI.h:106

Position of the 0th bit of the Slave Select Master Mode field in the ARM_SPI structure.

18.10.4.25 ARM_SPI_SS_MASTER_MODE_Msk

```
#define ARM_SPI_SS_MASTER_MODE_Msk (3UL << ARM_SPI_SS_MASTER_MODE_Pos)
```

Location: Driver_SPI.h:107

Positioning of the Slave Select Master Mode field in the ARM_SPI structure.

18.10.4.26 ARM_SPI_SS_MASTER_UNUSED

```
#define ARM_SPI_SS_MASTER_UNUSED (0UL << ARM_SPI_SS_MASTER_MODE_Pos)
```

Location: Driver_SPI.h:108

SPI Slave Select when Master: Not used (default).

18.10.4.27 ARM_SPI_SS_MASTER_SW

```
#define ARM_SPI_SS_MASTER_SW (1UL << ARM_SPI_SS_MASTER_MODE_Pos)
```

Location: Driver_SPI.h:109

SPI Slave Select when Master: Software controlled.

18.10.4.28 ARM_SPI_SS_MASTER_HW_OUTPUT

```
#define ARM_SPI_SS_MASTER_HW_OUTPUT (2UL << ARM_SPI_SS_MASTER_MODE_Pos)
```

Location: Driver_SPI.h:110

SPI Slave Select when Master: Hardware controlled Output.

18.10.4.29 ARM_SPI_SS_MASTER_HW_INPUT

```
#define ARM_SPI_SS_MASTER_HW_INPUT (3UL << ARM_SPI_SS_MASTER_MODE_Pos)
```

Location: Driver_SPI.h:111

SPI Slave Select when Master: Hardware monitored Input.

18.10.4.30 ARM_SPI_SS_SLAVE_MODE_Pos

```
#define ARM_SPI_SS_SLAVE_MODE_Pos 21
```

Location: Driver_SPI.h:112

Position of the 0th bit of the Slave Select Slave Mode field in the ARM_SPI structure.

18.10.4.31 ARM_SPI_SS_SLAVE_MODE_Msk

```
#define ARM_SPI_SS_SLAVE_MODE_Msk (1UL << ARM_SPI_SS_SLAVE_MODE_Pos)
```

Location: Driver_SPI.h:113

Positioning of the Slave Select Slave mode field in the ARM_SPI structure.

18.10.4.32 ARM_SPI_SS_SLAVE_HW

```
#define ARM_SPI_SS_SLAVE_HW (0UL << ARM_SPI_SS_SLAVE_MODE_Pos)
```

Location: Driver_SPI.h:114

SPI Slave Select when Slave: Hardware monitored (default).

18.10.4.33 ARM_SPI_SS_SLAVE_SW

```
#define ARM_SPI_SS_SLAVE_SW (1UL << ARM_SPI_SS_SLAVE_MODE_Pos)
```

Location: Driver_SPI.h:115

SPI Slave Select when Slave: Software controlled.

18.10.4.34 ARM_SPI_SET_BUS_SPEED

```
#define ARM_SPI_SET_BUS_SPEED (0x10UL << ARM_SPI_CONTROL_Pos)
```

Location: Driver_SPI.h:119

Set Bus Speed in bps; arg = value.

18.10.4.35 ARM_SPI_GET_BUS_SPEED

```
#define ARM_SPI_GET_BUS_SPEED (0x11UL << ARM_SPI_CONTROL_Pos)
```

Location: Driver_SPI.h:120

Get Bus Speed in bps.

18.10.4.36 ARM_SPI_SET_DEFAULT_TX_VALUE

```
#define ARM_SPI_SET_DEFAULT_TX_VALUE (0x12UL << ARM_SPI_CONTROL_Pos)
```

Location: Driver_SPI.h:121

Set default Transmit value; arg = value.

18.10.4.37 ARM_SPI_CONTROL_SS

```
#define ARM_SPI_CONTROL_SS (0x13UL << ARM_SPI_CONTROL_Pos)
```

Location: Driver_SPI.h:122

Control Slave Select; arg: 0=inactive, 1=active.

18.10.4.38 ARM_SPI_ABORT_TRANSFER

```
#define ARM_SPI_ABORT_TRANSFER (0x14UL << ARM_SPI_CONTROL_Pos)
```

Location: Driver_SPI.h:123

Abort current data transfer.

18.10.4.39 ARM_SPI_SS_INACTIVE

```
#define ARM_SPI_SS_INACTIVE 0
```

Location: Driver_SPI.h:127

SPI Slave Select Signal Inactive.

18.10.4.40 ARM_SPI_SS_ACTIVE

```
#define ARM_SPI_SS_ACTIVE 1
```

Location: Driver_SPI.h:128

SPI Slave Select Signal Active.

18.10.4.41 ARM_SPI_ERROR_MODE

```
#define ARM_SPI_ERROR_MODE (ARM_DRIVER_ERROR_SPECIFIC - 1)
```

Location: Driver_SPI.h:132

Specified Mode not supported.

18.10.4.42 ARM_SPI_ERROR_FRAME_FORMAT

```
#define ARM_SPI_ERROR_FRAME_FORMAT (ARM_DRIVER_ERROR_SPECIFIC - 2)
```

Location: Driver_SPI.h:133

Specified Frame Format not supported.

18.10.4.43 ARM_SPI_ERROR_DATA_BITS

```
#define ARM_SPI_ERROR_DATA_BITS (ARM_DRIVER_ERROR_SPECIFIC - 3)
```

Location: Driver_SPI.h:134

Specified number of Data bits not supported.

18.10.4.44 ARM_SPI_ERROR_BIT_ORDER

```
#define ARM_SPI_ERROR_BIT_ORDER (ARM_DRIVER_ERROR_SPECIFIC - 4)
```

Location: Driver_SPI.h:135

Specified Bit order not supported.

18.10.4.45 ARM_SPI_ERROR_SS_MODE

```
#define ARM_SPI_ERROR_SS_MODE (ARM_DRIVER_ERROR_SPECIFIC - 5)
```

Location: Driver_SPI.h:136

Specified Slave Select Mode not supported.

18.10.4.46 ARM_SPI_EVENT_TRANSFER_COMPLETE

```
#define ARM_SPI_EVENT_TRANSFER_COMPLETE (1UL << 0)
```

Location: Driver_SPI.h:152

SPI Event

Data Transfer completed.

18.10.4.47 ARM_SPI_EVENT_DATA_LOST

```
#define ARM_SPI_EVENT_DATA_LOST (1UL << 1)
```

Location: Driver_SPI.h:153

Data lost: Receive overflow / Transmit underflow.

18.10.4.48 ARM_SPI_EVENT_MODE_FAULT

```
#define ARM_SPI_EVENT_MODE_FAULT (1UL << 2)
```

Location: Driver_SPI.h:154

Master Mode Fault (SS deactivated when Master).

18.10.5 CMSIS SPI Driver Function Documentation

18.10.5.1 ARM_SPI_GetVersion

```
ARM_DRIVER_VERSION ARM_SPI_GetVersion()
```

Location: Driver_SPI.c:42

Get driver version.

Returns:

ARM_DRIVER_VERSION

18.10.5.2 ARM_SPI_GetCapabilities

```
ARM_SPI_CAPABILITIES ARM_SPI_GetCapabilities()
```

Location: Driver_SPI.c:47

Get driver capabilities.

Returns:

ARM_SPI_CAPABILITIES

18.10.5.3 ARM_SPI_Initialize

```
int32_t ARM_SPI_Initialize(ARM_SPI_SignalEvent_t cb_event)
```

Location: Driver_SPI.c:52

Initialize SPI Interface.

Returns:

execution_status

Parameters

Direction	Name	Description
in	*cb_event*	Pointer to ARM_SPI_SignalEvent

18.10.5.4 ARM_SPI_Uninitialize

```
int32_t ARM_SPI_Uninitialize()
```

Location: Driver_SPI.c:56

De-initialize SPI Interface.

Returns:

execution_status

18.10.5.5 ARM_SPI_PowerControl

```
int32_t ARM_SPI_PowerControl(ARM_POWER_STATE state)
```

Location: Driver_SPI.c:60

Control SPI Interface Power.

Returns:

execution_status

Parameters

Direction	Name	Description
in	*state*	Power state

18.10.5.6 ARM_SPI_Send

```
int32_t ARM_SPI_Send(const void * data, uint32_t num)
```

Location: Driver_SPI.c:76

Start sending data to SPI transmitter.

Returns:

execution_status

Parameters

Direction	Name	Description
in	*data*	Pointer to buffer with data to send to SPI transmitter
in	*num*	Number of data items to send

18.10.5.7 ARM_SPI_Receive

```
int32_t ARM_SPI_Receive(void * data, uint32_t num)
```

Location: Driver_SPI.c:80

Start receiving data from SPI receiver.

Returns:

execution_status

Parameters

Direction	Name	Description
out	*data*	Pointer to buffer for data to receive from SPI receiver
in	*num*	Number of data items to receive

18.10.5.8 ARM_SPI_Transfer

```
int32_t ARM_SPI_Transfer(const void * data_out, void * data_in, uint32_t num)
```

Location: Driver_SPI.c:84

Start sending/receiving data to/from SPI transmitter/receiver.

Returns:

execution_status

Parameters

Direction	Name	Description
`in`	*data_out*	Pointer to buffer with data to send to SPI transmitter
`out`	*data_in*	Pointer to buffer for data to receive from SPI receiver
`in`	*num*	Number of data items to transfer

18.10.5.9 ARM_SPI_GetDataCount

```
uint32_t ARM_SPI_GetDataCount()
```

Location: Driver_SPI.c:88

Get transferred data count.

Returns:

number of data items transferred

18.10.5.10 ARM_SPI_Control

```
int32_t ARM_SPI_Control(uint32_t control, uint32_t arg)
```

Location: Driver_SPI.c:92

Control SPI Interface.

Returns:

common execution_status and driver specific spi_execution_status

Parameters

Direction	Name	Description
in	*control*	Operation
in	*arg*	Argument of operation (optional)

18.10.5.11 ARM_SPI_GetStatus

```
ARM_SPI_STATUS ARM_SPI_GetStatus()
```

Location: Driver_SPI.c:125

Get SPI status.

Returns:

SPI status ARM_SPI_STATUS

18.10.5.12 ARM_SPI_SignalEvent

```
void ARM_SPI_SignalEvent(uint32_t event)
```

Location: Driver_SPI.c:129

Signal SPI Events.

Returns:

none

Parameters

Direction	Name	Description
in	*event*	SPI_events notification mask

18.11 CMSIS Timer Driver

CMSIS Timer Driver Reference.

18.11.1 Summary

Typedefs

- TIMER_SEL_t : Timer selection.
- TIMER_MODE_t : Timer mode selection.
- TIMER_CLKSRC_t : Timer clock source selection.
- TIMER_PRESCALE_t : Timer prescale values selection.
- TIMER_MULTI_COUNT_t : Timer multi-count values selection.
- TIMER_GPIO_STATUS_t : Timer GPIO status.
- TIMER_GPIO_INT_MODE_t : Timer GPIO capture mode.
- TIMER_GPIO_t : Timer GPIO interrupt selection.
- TIMER_SYSTICK_CLKSRC_t : Timer SysTick Clock sources.
- ADC_EVENT_t : Timer interrupt events selection.
- TIMER_SignalEvent_t : Pointer to TIMER_SignalEvent : Signal Timer event.
- TIMER_t : Timer Driver configuration.
- SYSTICK_t : SysTick Driver configuration.
- TIMER_CFG_t : Common TIMER driver configuration.
- TIMER_PRI_CFG_t : Timer interrupt priority configuration.
- DRIVER_TIMER_t : Access structure of the TIMER Driver.

Data Structures

- _TIMER_t : Timer Driver configuration.
- _SYSTICK_t : SysTick Driver configuration.
- _TIMER_PRI_CFG_t : Timer interrupt priority configuration.
- _DRIVER_TIMER_t : Access structure of the TIMER Driver.

Enumerations

- _TIMER_SEL_t : Timer selection.
- _TIMER_MODE_t : Timer mode selection.
- _TIMER_CLKSRC_t : Timer clock source selection.
- _TIMER_PRESCALE_t : Timer prescale values selection.
- _TIMER_MULTI_COUNT_t : Timer multi-count values selection.
- _TIMER_GPIO_STATUS_t : Timer GPIO status.

- _TIMER_GPIO_INT_MODE_t : Timer GPIO capture mode.
- _TIMER_GPIO_t : Timer GPIO interrupt selection.
- _TIMER_SYSTICK_CLKSRC_t : Timer SysTick Clock sources.
- _ADC_EVENT_t : Timer interrupt events selection.

Macros

- ARM_TIMER_API_VERSION : Timer API version.
- TIMER_ERROR_UNCONFIGURED : Driver has not been configured yet.

Functions

- TIMER_GetVersion : Get driver version.
- TIMER_Initialize : Initialize Timer driver with default configuration.
- TIMER_Configure : Configure particular Timer.
- TIMER_SetInterruptPriority : Configure the Timer interrupt priority.
- TIMER_Start : Starts the Timer.
- TIMER_Stop : Stops the Timer.
- TIMER_SetValue : Sets the timeout / reload value of the selected Timer.
- TIMER_GetValue : Returns the current value of Timer.
- TIMER_GetValueCapture : Returns the current value of Timer.
- TIMER_GetSysTickState : Returns 1 if SysTick has already reached 0.
- TIMER_SignalEvent : Signal Timer events.
- TIMER_SetGPIOInterrupt : Set GPIO interrupt capture status.

18.11.2 CMSIS Timer Driver Typedef Documentation

18.11.2.1 TIMER_SEL_t

```
typedef enum _TIMER_SEL_t TIMER_SEL_t
```

Location: Driver_TIMER.h:49

Timer selection.

18.11.2.2 TIMER_MODE_t

```
typedef enum _TIMER_MODE_t TIMER_MODE_t
```

Location: Driver_TIMER.h:57

Timer mode selection.

18.11.2.3 TIMER_CLKSRC_t

```
typedef enum _TIMER_CLKSRC_t TIMER_CLKSRC_t
```

Location: Driver_TIMER.h:65

Timer clock source selection.

18.11.2.4 TIMER_PRESCALE_t

```
typedef enum _TIMER_PRESCALE_t TIMER_PRESCALE_t
```

Location: Driver_TIMER.h:79

Timer prescale values selection.

18.11.2.5 TIMER_MULTI_COUNT_t

```
typedef enum _TIMER_MULTI_COUNT_t TIMER_MULTI_COUNT_t
```

Location: Driver_TIMER.h:93

Timer multi-count values selection.

18.11.2.6 TIMER_GPIO_STATUS_t

```
typedef enum _TIMER_GPIO_STATUS_t TIMER_GPIO_STATUS_t
```

Location: Driver_TIMER.h:101

Timer GPIO status.

18.11.2.7 TIMER_GPIO_INT_MODE_t

```
typedef enum _TIMER_GPIO_INT_MODE_t TIMER_GPIO_INT_MODE_t
```

Location: Driver_TIMER.h:109

Timer GPIO capture mode.

18.11.2.8 TIMER_GPIO_t

```
typedef enum _TIMER_GPIO_t TIMER_GPIO_t
```

Location: Driver_TIMER.h:119

Timer GPIO interrupt selection.

18.11.2.9 TIMER_SYSTICK_CLKSRC_t

```
typedef enum _TIMER_SYSTICK_CLKSRC_t TIMER_SYSTICK_CLKSRC_t
```

Location: Driver_TIMER.h:127

Timer SysTick Clock sources.

18.11.2.10 ADC_EVENT_t

```
typedef enum _ADC_EVENT_t ADC_EVENT_t
```

Location: Driver_TIMER.h:138

Timer interrupt events selection.

18.11.2.11 TIMER_SignalEvent_t

```
typedef void(* TIMER_SignalEvent_t
```

Location: Driver_TIMER.h:207

Pointer to TIMER_SignalEvent : Signal Timer event.

18.11.2.12 TIMER_t

```
typedef struct _TIMER_t TIMER_t
```

Location: Driver_TIMER.h:223

Timer Driver configuration.

18.11.2.13 SYSTICK_t

```
typedef struct _SYSTICK_t SYSTICK_t
```

Location: Driver_TIMER.h:233

SysTick Driver configuration.

18.11.2.14 TIMER_CFG_t

```
typedef union _TIMER_CFG_t TIMER_CFG_t
```

Location: Driver_TIMER.h:242

Common TIMER driver configuration.

18.11.2.15 TIMER_PRI_CFG_t

```
typedef struct _TIMER_PRI_CFG_t TIMER_PRI_CFG_t
```

Location: Driver_TIMER.h:253

Timer interrupt priority configuration.

18.11.2.16 DRIVER_TIMER_t

```
typedef struct _DRIVER_TIMER_t DRIVER_TIMER_t
```

Location: Driver_TIMER.h:270

Access structure of the TIMER Driver.

18.11.3 CMSIS Timer Driver Data Structures Type Documentation

18.11.3.1 _TIMER_t

Location: Driver_TIMER.h:212

Timer Driver configuration.

Data Fields

Type	*Name*	Description
TIMER_MODE_t	*mode*	Timer mode to be used.
TIMER_CLKSRC_t	*clk_src*	Clock source to be used.
TIMER_GPIO_INT_MODE_t	*gpio_mode*	GPIO capture mode to be used.
uint32_t	*__pad0__*	Reserved.
TIMER_PRESCALE_t	*prescale_val*	Timer prescale value.
TIMER_MULTI_COUNT_t	*multi_cnt*	Multi count value.
uint32_t	*__pad1__*	Reserved.
TIMER_GPIO_t	*gpio_int*	GPIO value.
uint32_t	*timeout_val*	Timer timeout value.

18.11.3.2 _SYSTICK_t

Location: Driver_TIMER.h:228

SysTick Driver configuration.

Data Fields

Type	Name	Description
TIMER_SYSTICK_CLKSRC_t	*clk_src*	Clock source to be used.
uint32_t	*__pad0__*	Reserved.
uint32_t	*reload_val*	SysTick value.

18.11.3.3 _TIMER_PRI_CFG_t

Location: Driver_TIMER.h:247

Timer interrupt priority configuration.

Data Fields

Type	Name	Description
uint32_t	*preempt_pri*	Preempt priority.
uint32_t	*__pad0__*	Reserved.
uint32_t	*subgrp_pri*	Subgroup priority.
uint32_t	*__pad1__*	Reserved.

18.11.3.4 _DRIVER_TIMER_t

Location: Driver_TIMER.h:258

Access structure of the TIMER Driver.

Data Fields

Type	Name	Description
ARM_DRIVER_VERSION(*	*GetVersion)(void)*	Pointer to TIMER_GetVersion : Get driver version.
int32_t(*	*Initialize)(TIMER_*	Pointer to TIMER_Initialize : Initialize Timer driver.

	SignalEvent_t cb)	
`int32_t(*`	*Configure)(TIMER_SEL_t sel, const TIMER_CFG_t *cfg)*	Pointer to TIMER_Configure : Configure driver.
`int32_t(*`	*SetInterruptPriority)(TIMER_SEL_t sel, const TIMER_PRI_CFG_t *pri)*	Pointer to TIMER_SetInterruptPriority : Configure Timer interrupt priority.
`int32_t(*`	*Start)(TIMER_SEL_t sel)*	Pointer to TIMER_Start : Start particular Timer.
`int32_t(*`	*Stop)(TIMER_SEL_t sel)*	Pointer to TIMER_Stop : Stop particular Timer.
`int32_t(*`	*SetValue)(TIMER_SEL_t sel, uint32_t value)*	Pointer to TIMER_SetValue : Set the particular Timer value.
`int32_t(*`	*SetGPIOInterrupt)(TIMER_SEL_t sel)*	Pointer to TIMER_SetGPIOInterrupt : Set GPIO interrupt capture status.
`uint32_t(*`	*GetValue)(TIMER_SEL_t sel)*	Pointer to TIMER_GetValue : Get the particular Timer value.
`uint32_t(*`	*GetValueCapture)(TIMER_SEL_t sel)*	Pointer to TIMER_GetValueCapture : Get the Timer GPIO Interrupt Captured Value.
`uint32_t(*`	*GetSysTickState)(void)*	Pointer to TIMER_GetSysTickState : Returns 1 if SysTick has already reached 0.

18.11.4 CMSIS Timer Driver Enumeration Type Documentation

18.11.4.1 _TIMER_SEL_t

Location: Driver_TIMER.h:43

Timer selection.

Members

- `TIMER_0 = 0`

  Timer module 0.

- `TIMER_1 = 1`

  Timer module 1.

- `TIMER_2 = 2`

  Timer module 2.

- `TIMER_3 = 3`

  Timer module 3.

- `TIMER_SYSTICK = 4`

  Timer module SysTick.

18.11.4.2 _TIMER_MODE_t

Location: Driver_TIMER.h:54

Timer mode selection.

Members

- `TIMER_MODE_SHOT = 0x0U`

  Timer mode = TIMER_SHOT_MODE_BITBAND.

- `TIMER_MODE_FREE_RUN = 0x1U`

  Timer mode = TIMER_FREE_RUN_BITBAND.

18.11.4.3 _TIMER_CLKSRC_t

Location: Driver_TIMER.h:62

Timer clock source selection.

Members

- `TIMER_SLOWCLOCK_DIV32 = 0x0U`

  Timer src = SLOWCLOCK DIV32.

- `TIMER_SLOWCLOCK_DIV2 = 0x1U`

  Timer src = SLOWCLOCK DIV2.

18.11.4.4 _TIMER_PRESCALE_t

Location: Driver_TIMER.h:70

Timer prescale values selection.

Members

- `TIMER_PRESCALE_VAL_1 = 0x0U`

  Timer prescale = 1.

- `TIMER_PRESCALE_VAL_2 = 0x1U`

  Timer prescale = 2.

- `TIMER_PRESCALE_VAL_4 = 0x2U`

  Timer prescale = 4.

- `TIMER_PRESCALE_VAL_8 = 0x3U`

  Timer prescale = 8.

- `TIMER_PRESCALE_VAL_16 = 0x4U`

  Timer prescale = 16.

- `TIMER_PRESCALE_VAL_32 = 0x5U`

  Timer prescale = 32.

- `TIMER_PRESCALE_VAL_64 = 0x6U`

  Timer prescale = 64.

- `TIMER_PRESCALE_VAL_128 = 0x7U`

  Timer prescale = 128.

18.11.4.5 _TIMER_MULTI_COUNT_t

Location: Driver_TIMER.h:84

Timer multi-count values selection.

Members

- `TIMER_MULTI_COUNT_VAL_1 = 0x0U`

  Timer multiCount = 1.

- `TIMER_MULTI_COUNT_VAL_2 = 0x1U`

  Timer multiCount = 2.

- `TIMER_MULTI_COUNT_VAL_3 = 0x2U`

  Timer multiCount = 3.

- `TIMER_MULTI_COUNT_VAL_4 = 0x3U`

  Timer multiCount = 4.

- `TIMER_MULTI_COUNT_VAL_5 = 0x4U`

  Timer multiCount = 5.

- `TIMER_MULTI_COUNT_VAL_6 = 0x5U`

  Timer multiCount = 6.

- `TIMER_MULTI_COUNT_VAL_7 = 0x6U`

  Timer multiCount = 7.

- `TIMER_MULTI_COUNT_VAL_8 = 0x7U`

  Timer multiCount = 8.

18.11.4.6 _TIMER_GPIO_STATUS_t

Location: Driver_TIMER.h:98

Timer GPIO status.

Members

- `TIMER_GPIO_INT_DISABLE_STATUS = 0x0U`

  Timer GPIO status = disable.

- `TIMER_GPIO_INT_ENABLE_STATUS = 0x1U`

  Timer GPIO status = enable.

18.11.4.7 _TIMER_GPIO_INT_MODE_t

Location: Driver_TIMER.h:106

Timer GPIO capture mode.

Members

- `TIMER_GPIO_SINGLE_MODE = 0x0U`

  Timer capture mode = single.

- `TIMER_GPIO_CONTINUOUS_MODE = 0x1U`

  Timer capture mode = continuous.

18.11.4.8 _TIMER_GPIO_t

Location: Driver_TIMER.h:114

Timer GPIO interrupt selection.

Members

- `TIMER_GPIO_0 = 0x0U`

  Timer GPIO interrupt 0.

- `TIMER_GPIO_1 = 0x1U`

  Timer GPIO interrupt 1.

- `TIMER_GPIO_2 = 0x2U`

  Timer GPIO interrupt 2.

- `TIMER_GPIO_3 = 0x3U`

  Timer GPIO interrupt 3.

18.11.4.9 _TIMER_SYSTICK_CLKSRC_t

Location: Driver_TIMER.h:124

Timer SysTick Clock sources.

Members

- `SYSTICK_CLKSOURCE_EXTREFCLK = 0x0U`

  SysTick Timer CLK src = external ref.

- `SYSTICK_CLKSOURCE_CORECLK = 0x1U`

  SysTick Timer CLK src = core CLK.

18.11.4.10 _ADC_EVENT_t

Location: Driver_TIMER.h:132

Timer interrupt events selection.

Members

- `TIMER_TIMER0_EVENT = 1 << TIMER_0`

  Timer0 event.

- `TIMER_TIMER1_EVENT = 1 << TIMER_1`

Timer1 event.

- `TIMER_TIMER2_EVENT = 1 << TIMER_2`

  Timer2 event.

- `TIMER_TIMER3_EVENT = 1 << TIMER_3`

  Timer3 event.

- `TIMER_SYSTICK_EVENT = 1 << TIMER_SYSTICK`

  SysTick event.

18.11.5 CMSIS Timer Driver Macro Definition Documentation

18.11.5.1 ARM_TIMER_API_VERSION

```
#define ARM_TIMER_API_VERSION ARM_DRIVER_VERSION_MAJOR_MINOR(1,0)
```

Location: Driver_TIMER.h:36

Timer API version.

18.11.5.2 TIMER_ERROR_UNCONFIGURED

```
#define TIMER_ERROR_UNCONFIGURED (ARM_DRIVER_ERROR_SPECIFIC - 1)
```

Location: Driver_TIMER.h:141

Driver has not been configured yet.

18.11.6 CMSIS Timer Driver Function Documentation

18.11.6.1 TIMER_GetVersion

```
ARM_DRIVER_VERSION TIMER_GetVersion()
```

Location: Driver_Timer.c:21

Get driver version.

Returns:

ARM_DRIVER_VERSION

18.11.6.2 TIMER_Initialize

```
int32_t TIMER_Initialize(TIMER_SignalEvent_t cb)
```

Location: Driver_Timer.c:22

Initialize Timer driver with default configuration.

Returns:

execution_status

Parameters

Direction	Name	Description
in	*cb*	Pointer to TIMER_SignalEvent

18.11.6.3 TIMER_Configure

```
int32_t TIMER_Configure(TIMER_SEL_t sel, const TIMER_CFG_t * cfg)
```

Location: Driver_Timer.c:23

Configure particular Timer.

Returns:

execution_status

Parameters

Direction	Name	Description
in	*sel*	Timer to be configured (TIMER_SEL_t)
in	*cfg*	Pointer to TIMER_CFG_t

18.11.6.4 TIMER_SetInterruptPriority

```
int32_t TIMER_SetInterruptPriority(TIMER_SEL_t sel, const TIMER_PRI_CFG_t * cfg)
```

Location: Driver_Timer.c:24

Configure the Timer interrupt priority.

Returns:

execution_status

Parameters

Direction	Name	Description
in	*sel*	Timer to be configured (TIMER_SEL_t)
in	*cfg*	Pointer to TIMER_PRI_CFG_t

18.11.6.5 TIMER_Start

```
int32_t TIMER_Start(TIMER_SEL_t sel)
```

Location: Driver_Timer.c:25

Starts the Timer.

Returns:

execution_status

Parameters

Direction	Name	Description
in	*sel*	Timer number to be started (TIMER_SEL_t)

18.11.6.6 TIMER_Stop

```
int32_t TIMER_Stop(TIMER_SEL_t sel)
```

Location: Driver_Timer.c:26

Stops the Timer.

Returns:

execution_status

Parameters

Direction	Name	Description
in	*sel*	Timer number to be stopped (TIMER_SEL_t)

18.11.6.7 TIMER_SetValue

```
int32_t TIMER_SetValue(TIMER_SEL_t sel, uint32_t val)
```

Location: Driver_Timer.c:27

Sets the timeout / reload value of the selected Timer.

Returns:

execution_status of error status

Parameters

Direction	Name	Description
in	*sel*	Timer value to be read (TIMER_SEL_t)
in	*val*	Timer value to be set

18.11.6.8 TIMER_GetValue

```
uint32_t TIMER_GetValue(TIMER_SEL_t sel)
```

Location: Driver_Timer.c:29

Returns the current value of Timer.

Returns:

Timer value or 0 if Timer was not enabled

Parameters

Direction	Name	Description
in	*sel*	Timer value to be read (TIMER_SEL_t)

18.11.6.9 TIMER_GetValueCapture

```
uint32_t TIMER_GetValueCapture(TIMER_SEL_t sel)
```

Location: Driver_Timer.c:30

Returns the current value of Timer.

Returns:

Timer capture value or 0 if Timer was not enabled

Parameters

Direction	Name	Description
in	*sel*	Timer value to be read (TIMER_SEL_t)

18.11.6.10 TIMER_GetSysTickState

```
uint32_t TIMER_GetSysTickState()
```

Location: Driver_Timer.c:31

Returns 1 if SysTick has already reached 0.

Returns:

SysTick status or 0 if SysTick was not enabled

18.11.6.11 TIMER_SignalEvent

```
void TIMER_SignalEvent(uint32_t event)
```

Location: Driver_Timer.c:33

Signal Timer events.

Returns:

none

Parameters

Direction	Name	Description
in	*event*	Notification mask

18.11.6.12 TIMER_SetGPIOInterrupt

```
int32_t TIMER_SetGPIOInterrupt(TIMER_SEL_t sel)
```

Location: Driver_Timer.c:28

Set GPIO interrupt capture status.

Returns:

execution_status

Parameters

Direction	Name	Description
in	*sel*	Timer value to be read (TIMER_SEL_t)

18.12 CMSIS USART DRIVER

CMSIS USART Driver Reference.

18.12.1 Summary

Typedefs

- ARM_USART_STATUS : USART Status.
- ARM_USART_MODEM_CONTROL : USART Modem Control.
- ARM_USART_MODEM_STATUS : USART Modem Status.
- ARM_USART_SignalEvent_t : Pointer to ARM_USART_SignalEvent : Signal USART Event.
- ARM_USART_CAPABILITIES : USART Device Driver Capabilities.
- ARM_DRIVER_USART : Access structure of the USART Driver.

Data Structures

- _ARM_USART_STATUS : USART Status.
- _ARM_USART_MODEM_STATUS : USART Modem Status.
- _ARM_USART_CAPABILITIES : USART Device Driver Capabilities.
- _ARM_DRIVER_USART : Access structure of the USART Driver.

Enumerations

- _ARM_USART_MODEM_CONTROL : USART Modem Control.

Macros

- ARM_USART_API_VERSION : API version.
- ARM_USART_CONTROL_Pos : Position of the 0th bit of the USART control field in the ARM_USART structure.
- ARM_USART_CONTROL_Msk : Positioning of USART control field in the ARM_USART structure.
- ARM_USART_MODE_ASYNCHRONOUS : UART (Asynchronous); arg = Baudrate.
- ARM_USART_MODE_SYNCHRONOUS_MASTER : Synchronous Master (generates clock signal); arg = Baudrate.
- ARM_USART_MODE_SYNCHRONOUS_SLAVE : Synchronous Slave (external clock signal).
- ARM_USART_MODE_SINGLE_WIRE : UART Single-wire (half-duplex); arg = Baudrate.
- ARM_USART_MODE_IRDA : UART IrDA; arg = Baudrate.

- ARM_USART_MODE_SMART_CARD : UART Smart Card; arg = Baudrate.
- ARM_USART_DATA_BITS_Pos : Position of the 0th bit of the Data bits field in the ARM_USART structure.
- ARM_USART_DATA_BITS_Msk : Positioning of the Data bits field in the ARM_USART structure.
- ARM_USART_DATA_BITS_5 : 5 data bits.
- ARM_USART_DATA_BITS_6 : 6 data bits.
- ARM_USART_DATA_BITS_7 : 7 data bits.
- ARM_USART_DATA_BITS_8 : 8 data bits (default).
- ARM_USART_DATA_BITS_9 : 9 data bits.
- ARM_USART_PARITY_Pos : Position of the 0th bit of the Mode parameters Parity field in the ARM_USART structure.
- ARM_USART_PARITY_Msk : Positioning of the Mode parameters Parity field in the ARM_USART structure.
- ARM_USART_PARITY_NONE : No parity (default).
- ARM_USART_PARITY_EVEN : Even parity.
- ARM_USART_PARITY_ODD : Odd parity.
- ARM_USART_STOP_BITS_Pos : Position of the 0th bit of the Mode parameters Stop bits field in the ARM_USART structure.
- ARM_USART_STOP_BITS_Msk : Positioning of the Mode parameters Stop bits field in the ARM_USART structure.
- ARM_USART_STOP_BITS_1 : 1 stop bit (default).
- ARM_USART_STOP_BITS_2 : 2 stop bits.
- ARM_USART_STOP_BITS_1_5 : 1.5 stop bits.
- ARM_USART_STOP_BITS_0_5 : 0.5 stop bits.
- ARM_USART_FLOW_CONTROL_Pos : Position of the 0th bit of the Mode parameters Flow control field in the ARM_USART structure.
- ARM_USART_FLOW_CONTROL_Msk : Positioning of the Mode parameters Flow control field in the ARM_USART structure.
- ARM_USART_FLOW_CONTROL_NONE : No flow control (default).
- ARM_USART_FLOW_CONTROL_RTS : RTS flow control.
- ARM_USART_FLOW_CONTROL_CTS : CTS flow control.
- ARM_USART_FLOW_CONTROL_RTS_CTS : RTS/CTS flow control.
- ARM_USART_CPOL_Pos : Position of the 0th bit of the Mode parameters Clock polarity field in the ARM_USART structure.
- ARM_USART_CPOL_Msk : Positioning of the Mode parameters Clock polarity field in the ARM_USART structure.
- ARM_USART_CPOL0 : CPOL = 0 (default).
- ARM_USART_CPOL1 : CPOL = 1.
- ARM_USART_CPHA_Pos : Position of the 0th bit of the Mode parameters Clock phase field in the ARM_USART structure.
- ARM_USART_CPHA_Msk : Positioning of the Mode parameters Clock phase field in the ARM_USART structure.
- ARM_USART_CPHA0 : CPHA = 0 (default).
- ARM_USART_CPHA1 : CPHA = 1.
- ARM_USART_SET_DEFAULT_TX_VALUE : Set default transmit value (synchronous receive only); arg = value.
- ARM_USART_SET_IRDA_PULSE : Set IrDA Pulse in ns; arg: 0=3/16 of bit period.
- ARM_USART_SET_SMART_CARD_GUARD_TIME : Set smart card guard time; arg = number of bit periods.
- ARM_USART_SET_SMART_CARD_CLOCK : Set smart card clock in Hz; arg: 0=Clock not generated.

- ARM_USART_CONTROL_SMART_CARD_NACK : Smart card NACK generation; arg: 0=disabled, 1=enabled.
- ARM_USART_CONTROL_TX : Transmitter; arg: 0=disabled, 1=enabled.
- ARM_USART_CONTROL_RX : Receiver; arg: 0=disabled, 1=enabled.
- ARM_USART_CONTROL_BREAK : Continuous break transmission; arg: 0=disabled, 1=enabled.
- ARM_USART_ABORT_SEND : Abort ARM_USART_Send.
- ARM_USART_ABORT_RECEIVE : Abort ARM_USART_Receive.
- ARM_USART_ABORT_TRANSFER : Abort ARM_USART_Transfer.
- ARM_USART_ERROR_MODE : Specified mode not supported.
- ARM_USART_ERROR_BAUDRATE : Specified baudrate not supported.
- ARM_USART_ERROR_DATA_BITS : Specified number of data bits not supported.
- ARM_USART_ERROR_PARITY : Specified parity not supported.
- ARM_USART_ERROR_STOP_BITS : Specified number of stop bits not supported.
- ARM_USART_ERROR_FLOW_CONTROL : Specified flow control not supported.
- ARM_USART_ERROR_CPOL : Specified clock polarity not supported.
- ARM_USART_ERROR_CPHA : Specified clock phase not supported.
- ARM_USART_EVENT_SEND_COMPLETE : USART Event
- ARM_USART_EVENT_RECEIVE_COMPLETE : Receive completed.
- ARM_USART_EVENT_TRANSFER_COMPLETE : Transfer completed.
- ARM_USART_EVENT_TX_COMPLETE : Transmit completed (optional).
- ARM_USART_EVENT_TX_UNDERFLOW : Transmit data not available (synchronous slave).
- ARM_USART_EVENT_RX_OVERFLOW : Receive data overflow.
- ARM_USART_EVENT_RX_TIMEOUT : Receive character timeout (optional).
- ARM_USART_EVENT_RX_BREAK : Break detected on receive.
- ARM_USART_EVENT_RX_FRAMING_ERROR : Framing error detected on receive.
- ARM_USART_EVENT_RX_PARITY_ERROR : Parity error detected on receive.
- ARM_USART_EVENT_CTS : CTS state changed (optional).
- ARM_USART_EVENT_DSR : DSR state changed (optional).
- ARM_USART_EVENT_DCD : DCD state changed (optional).
- ARM_USART_EVENT_RI : RI state changed (optional).

Functions

- ARM_USART_GetVersion : Get driver version.
- ARM_USART_GetCapabilities : Get driver capabilities.
- ARM_USART_Initialize : Initialize USART Interface.
- ARM_USART_Uninitialize : De-initialize USART Interface.
- ARM_USART_PowerControl : Control USART Interface Power.
- ARM_USART_Send : Start sending data to USART transmitter.
- ARM_USART_Receive : Start receiving data from USART receiver.
- ARM_USART_Transfer : Start sending/receiving data to/from USART transmitter/receiver.
- ARM_USART_GetTxCount : Get transmitted data count.
- ARM_USART_GetRxCount : Get received data count.
- ARM_USART_Control : Control USART Interface.
- ARM_USART_GetStatus : Get USART status.
- ARM_USART_SetModemControl : Set USART Modem Control line state.
- ARM_USART_GetModemStatus : Get USART Modem Status lines state.
- ARM_USART_SignalEvent : Signal USART Events.

18.12.2 CMSIS USART Driver Typedef Documentation

18.12.2.1 ARM_USART_STATUS

```
typedef struct _ARM_USART_STATUS ARM_USART_STATUS
```

Location: Driver_USART.h:176

USART Status.

18.12.2.2 ARM_USART_MODEM_CONTROL

```
typedef enum _ARM_USART_MODEM_CONTROL ARM_USART_MODEM_CONTROL
```

Location: Driver_USART.h:186

USART Modem Control.

18.12.2.3 ARM_USART_MODEM_STATUS

```
typedef struct _ARM_USART_MODEM_STATUS ARM_USART_MODEM_STATUS
```

Location: Driver_USART.h:197

USART Modem Status.

18.12.2.4 ARM_USART_SignalEvent_t

```
typedef void(* ARM_USART_SignalEvent_t
```

Location: Driver_USART.h:295

Pointer to ARM_USART_SignalEvent : Signal USART Event.

18.12.2.5 ARM_USART_CAPABILITIES

```
typedef struct _ARM_USART_CAPABILITIES ARM_USART_CAPABILITIES
```

Location: Driver_USART.h:324

USART Device Driver Capabilities.

18.12.2.6 ARM_DRIVER_USART

```
typedef struct _ARM_DRIVER_USART ARM_DRIVER_USART
```

Location: Driver_USART.h:347

Access structure of the USART Driver.

18.12.3 CMSIS USART Driver Data Structures Type Documentation

18.12.3.1 _ARM_USART_STATUS

Location: Driver_USART.h:167

USART Status.

Data Fields

Type	*Name*	Description
uint32_t	*tx_busy*	Transmitter busy flag.
uint32_t	*rx_busy*	Receiver busy flag.
uint32_t	*tx_underflow*	Transmit data underflow detected (cleared on start of next send operation).
uint32_t	*rx_overflow*	Receive data overflow detected (cleared on start of next receive operation).
uint32_t	*rx_break*	Break detected on receive (cleared on start of next receive operation).

uint32_t	*rx_framing_error*	Framing error detected on receive (cleared on start of next receive operation).
uint32_t	*rx_parity_error*	Parity error detected on receive (cleared on start of next receive operation).
uint32_t	*reserved*	(Reserved for future use)

18.12.3.2 _ARM_USART_MODEM_STATUS

Location: Driver_USART.h:191

USART Modem Status.

Data Fields

Type	*Name*	Description
uint32_t	*cts*	CTS state: 1=Active, 0=Inactive.
uint32_t	*dsr*	DSR state: 1=Active, 0=Inactive.
uint32_t	*dcd*	DCD state: 1=Active, 0=Inactive.
uint32_t	*ri*	RI state: 1=Active, 0=Inactive.
uint32_t	*reserved*	(Reserved for future use)

18.12.3.3 _ARM_USART_CAPABILITIES

Location: Driver_USART.h:301

USART Device Driver Capabilities.

Data Fields

Type	*Name*	Description
uint32_t	*asynchronous*	supports UART (asynchronous) mode.
uint32_t	*synchronous_master*	supports synchronous master mode.

uint32_t	*synchronous_slave*	supports synchronous slave mode.
uint32_t	*single_wire*	supports UART single-wire mode.
uint32_t	*irda*	supports UART IrDA mode.
uint32_t	*smart_card*	supports UART smart card mode.
uint32_t	*smart_card_clock*	Smart card clock generator available.
uint32_t	*flow_control_rts*	RTS flow control available.
uint32_t	*flow_control_cts*	CTS flow control available.
uint32_t	*event_tx_complete*	Transmit completed event: ARM_USART_EVENT_TX_COMPLETE.
uint32_t	*event_rx_timeout*	Signal receive character timeout event: ARM_USART_EVENT_RX_TIMEOUT.
uint32_t	*rts*	RTS Line: 0=not available, 1=available.
uint32_t	*cts*	CTS Line: 0=not available, 1=available.
uint32_t	*dtr*	DTR Line: 0=not available, 1=available.
uint32_t	*dsr*	DSR Line: 0=not available, 1=available.
uint32_t	*dcd*	DCD Line: 0=not available, 1=available.
uint32_t	*ri*	RI Line: 0=not available, 1=available.
uint32_t	*event_cts*	Signal CTS change event: ARM_USART_EVENT_CTS.
uint32_t	*event_dsr*	Signal DSR change event: ARM_USART_EVENT_DSR.
uint32_t	*event_dcd*	Signal DCD change event: ARM_USART_EVENT_DCD.
uint32_t	*event_ri*	Signal RI change event: ARM_USART_EVENT_RI.
uint32_t	*reserved*	Reserved (must be zero).

18.12.3.4 _ARM_DRIVER_USART

Location: Driver_USART.h:330

Access structure of the USART Driver.

Data Fields

Type	Name	Description
ARM_DRIVER_VERSION(*	*GetVersion)(void)*	Pointer to ARM_USART_GetVersion : Get driver version.
ARM_USART_CAPABILITIES(*	*GetCapabilities)(void)*	Pointer to ARM_USART_GetCapabilities : Get driver capabilities.
int32_t(*	*Initialize)(ARM_USART_SignalEvent_t cb_event)*	Pointer to ARM_USART_Initialize : Initialize USART Interface.
int32_t(*	*Uninitialize)(void)*	Pointer to ARM_USART_Uninitialize : De-initialize USART Interface.
int32_t(*	*PowerControl)(ARM_POWER_STATE state)*	Pointer to ARM_USART_PowerControl : Control USART Interface Power.
int32_t(*	*Send)(const void *data, uint32_t num)*	Pointer to ARM_USART_Send : Start sending data to USART transmitter.
int32_t(*	*Receive)(void *data, uint32_t num)*	Pointer to ARM_USART_Receive : Start receiving data from USART receiver.
int32_t(*	*Transfer)(const void *data_out, void *data_in, uint32_t num)*	Pointer to ARM_USART_Transfer : Start sending/receiving data to/from USART.
uint32_t(*	*GetTxCount)(void)*	Pointer to ARM_USART_GetTxCount : Get transmitted data count.
uint32_t(*	*GetRxCount)(void)*	Pointer to ARM_USART_GetRxCount : Get received data count.
int32_t(*	*Control)(uint32_t control, uint32_t arg)*	Pointer to ARM_USART_Control : Control USART Interface.
ARM_USART_STATUS(*	*GetStatus)(void)*	Pointer to ARM_USART_GetStatus : Get USART status.
int32_t(*	*SetModemControl)(ARM_USART_MODEM_CONTROL control)*	Pointer to ARM_USART_SetModemControl : Set USART modem control line state.
ARM_USART_MODEM_STATUS(*	*GetModemStatus)(void)*	Pointer to ARM_USART_GetModemStatus : Get USART modem status lines state.

18.12.4 CMSIS USART Driver Enumeration Type Documentation

18.12.4.1 _ARM_USART_MODEM_CONTROL

Location: Driver_USART.h:181

USART Modem Control.

Members

- `ARM_USART_RTS_CLEAR`

  Deactivate RTS.

- `ARM_USART_RTS_SET`

  Activate RTS.

- `ARM_USART_DTR_CLEAR`

  Deactivate DTR.

- `ARM_USART_DTR_SET`

  Activate DTR.

18.12.5 CMSIS USART Driver Macro Definition Documentation

18.12.5.1 ARM_USART_API_VERSION

```
#define ARM_USART_API_VERSION ARM_DRIVER_VERSION_MAJOR_MINOR(2,3)
```

Location: Driver_USART.h:78

API version.

18.12.5.2 ARM_USART_CONTROL_Pos

```
#define ARM_USART_CONTROL_Pos 0
```

Location: Driver_USART.h:81

Position of the 0th bit of the USART control field in the ARM_USART structure.

18.12.5.3 ARM_USART_CONTROL_Msk

```
#define ARM_USART_CONTROL_Msk (0xFFUL << ARM_USART_CONTROL_Pos)
```

Location: Driver_USART.h:82

Positioning of USART control field in the ARM_USART structure.

18.12.5.4 ARM_USART_MODE_ASYNCHRONOUS

```
#define ARM_USART_MODE_ASYNCHRONOUS (0x01UL << ARM_USART_CONTROL_Pos)
```

Location: Driver_USART.h:85

UART (Asynchronous); arg = Baudrate.

18.12.5.5 ARM_USART_MODE_SYNCHRONOUS_MASTER

```
#define ARM_USART_MODE_SYNCHRONOUS_MASTER (0x02UL << ARM_USART_CONTROL_Pos)
```

Location: Driver_USART.h:86

Synchronous Master (generates clock signal); arg = Baudrate.

18.12.5.6 ARM_USART_MODE_SYNCHRONOUS_SLAVE

```
#define ARM_USART_MODE_SYNCHRONOUS_SLAVE (0x03UL << ARM_USART_CONTROL_Pos)
```

Location: Driver_USART.h:87

Synchronous Slave (external clock signal).

18.12.5.7 ARM_USART_MODE_SINGLE_WIRE

```
#define ARM_USART_MODE_SINGLE_WIRE (0x04UL << ARM_USART_CONTROL_Pos)
```

Location: Driver_USART.h:88

UART Single-wire (half-duplex); arg = Baudrate.

18.12.5.8 ARM_USART_MODE_IRDA

```
#define ARM_USART_MODE_IRDA (0x05UL << ARM_USART_CONTROL_Pos)
```

Location: Driver_USART.h:89

UART IrDA; arg = Baudrate.

18.12.5.9 ARM_USART_MODE_SMART_CARD

```
#define ARM_USART_MODE_SMART_CARD (0x06UL << ARM_USART_CONTROL_Pos)
```

Location: Driver_USART.h:90

UART Smart Card; arg = Baudrate.

18.12.5.10 ARM_USART_DATA_BITS_Pos

```
#define ARM_USART_DATA_BITS_Pos 8
```

Location: Driver_USART.h:93

Position of the 0th bit of the Data bits field in the ARM_USART structure.

18.12.5.11 ARM_USART_DATA_BITS_Msk

```
#define ARM_USART_DATA_BITS_Msk (7UL << ARM_USART_DATA_BITS_Pos)
```

Location: Driver_USART.h:94

Positioning of the Data bits field in the ARM_USART structure.

18.12.5.12 ARM_USART_DATA_BITS_5

```
#define ARM_USART_DATA_BITS_5 (5UL << ARM_USART_DATA_BITS_Pos)
```

Location: Driver_USART.h:95

5 data bits.

18.12.5.13 ARM_USART_DATA_BITS_6

```
#define ARM_USART_DATA_BITS_6 (6UL << ARM_USART_DATA_BITS_Pos)
```

Location: Driver_USART.h:96

6 data bits.

18.12.5.14 ARM_USART_DATA_BITS_7

```
#define ARM_USART_DATA_BITS_7 (7UL << ARM_USART_DATA_BITS_Pos)
```

Location: Driver_USART.h:97

7 data bits.

18.12.5.15 ARM_USART_DATA_BITS_8

```
#define ARM_USART_DATA_BITS_8 (0UL << ARM_USART_DATA_BITS_Pos)
```

Location: Driver_USART.h:98

8 data bits (default).

18.12.5.16 ARM_USART_DATA_BITS_9

```
#define ARM_USART_DATA_BITS_9 (1UL << ARM_USART_DATA_BITS_Pos)
```

Location: Driver_USART.h:99

9 data bits.

18.12.5.17 ARM_USART_PARITY_Pos

```
#define ARM_USART_PARITY_Pos 12
```

Location: Driver_USART.h:102

Position of the 0th bit of the Mode parameters Parity field in the ARM_USART structure.

18.12.5.18 ARM_USART_PARITY_Msk

```
#define ARM_USART_PARITY_Msk (3UL << ARM_USART_PARITY_Pos)
```

Location: Driver_USART.h:103

Positioning of the Mode parameters Parity field in the ARM_USART structure.

18.12.5.19 ARM_USART_PARITY_NONE

```
#define ARM_USART_PARITY_NONE (0UL << ARM_USART_PARITY_Pos)
```

Location: Driver_USART.h:104

No parity (default).

18.12.5.20 ARM_USART_PARITY_EVEN

```
#define ARM_USART_PARITY_EVEN (1UL << ARM_USART_PARITY_Pos)
```

Location: Driver_USART.h:105

Even parity.

18.12.5.21 ARM_USART_PARITY_ODD

```
#define ARM_USART_PARITY_ODD (2UL << ARM_USART_PARITY_Pos)
```

Location: Driver_USART.h:106

Odd parity.

18.12.5.22 ARM_USART_STOP_BITS_Pos

```
#define ARM_USART_STOP_BITS_Pos 14
```

Location: Driver_USART.h:109

Position of the 0th bit of the Mode parameters Stop bits field in the ARM_USART structure.

18.12.5.23 ARM_USART_STOP_BITS_Msk

```
#define ARM_USART_STOP_BITS_Msk (3UL << ARM_USART_STOP_BITS_Pos)
```

Location: Driver_USART.h:110

Positioning of the Mode parameters Stop bits field in the ARM_USART structure.

18.12.5.24 ARM_USART_STOP_BITS_1

```
#define ARM_USART_STOP_BITS_1 (0UL << ARM_USART_STOP_BITS_Pos)
```

Location: Driver_USART.h:111

1 stop bit (default).

18.12.5.25 ARM_USART_STOP_BITS_2

```
#define ARM_USART_STOP_BITS_2 (1UL << ARM_USART_STOP_BITS_Pos)
```

Location: Driver_USART.h:112

2 stop bits.

18.12.5.26 ARM_USART_STOP_BITS_1_5

```
#define ARM_USART_STOP_BITS_1_5 (2UL << ARM_USART_STOP_BITS_Pos)
```

Location: Driver_USART.h:113

1.5 stop bits.

18.12.5.27 ARM_USART_STOP_BITS_0_5

```
#define ARM_USART_STOP_BITS_0_5 (3UL << ARM_USART_STOP_BITS_Pos)
```

Location: Driver_USART.h:114

0.5 stop bits.

18.12.5.28 ARM_USART_FLOW_CONTROL_Pos

```
#define ARM_USART_FLOW_CONTROL_Pos 16
```

Location: Driver_USART.h:117

Position of the 0th bit of the Mode parameters Flow control field in the ARM_USART structure.

18.12.5.29 ARM_USART_FLOW_CONTROL_Msk

```
#define ARM_USART_FLOW_CONTROL_Msk (3UL << ARM_USART_FLOW_CONTROL_Pos)
```

Location: Driver_USART.h:118

Positioning of the Mode parameters Flow control field in the ARM_USART structure.

18.12.5.30 ARM_USART_FLOW_CONTROL_NONE

```
#define ARM_USART_FLOW_CONTROL_NONE (0UL << ARM_USART_FLOW_CONTROL_Pos)
```

Location: Driver_USART.h:119

No flow control (default).

18.12.5.31 ARM_USART_FLOW_CONTROL_RTS

```
#define ARM_USART_FLOW_CONTROL_RTS (1UL << ARM_USART_FLOW_CONTROL_Pos)
```

Location: Driver_USART.h:120

RTS flow control.

18.12.5.32 ARM_USART_FLOW_CONTROL_CTS

```
#define ARM_USART_FLOW_CONTROL_CTS (2UL << ARM_USART_FLOW_CONTROL_Pos)
```

Location: Driver_USART.h:121

CTS flow control.

18.12.5.33 ARM_USART_FLOW_CONTROL_RTS_CTS

```
#define ARM_USART_FLOW_CONTROL_RTS_CTS (3UL << ARM_USART_FLOW_CONTROL_Pos)
```

Location: Driver_USART.h:122

RTS/CTS flow control.

18.12.5.34 ARM_USART_CPOL_Pos

```
#define ARM_USART_CPOL_Pos 18
```

Location: Driver_USART.h:125

Position of the 0th bit of the Mode parameters Clock polarity field in the ARM_USART structure.

18.12.5.35 ARM_USART_CPOL_Msk

```
#define ARM_USART_CPOL_Msk (1UL << ARM_USART_CPOL_Pos)
```

Location: Driver_USART.h:126

Positioning of the Mode parameters Clock polarity field in the ARM_USART structure.

18.12.5.36 ARM_USART_CPOL0

```
#define ARM_USART_CPOL0 (0UL << ARM_USART_CPOL_Pos)
```

Location: Driver_USART.h:127

CPOL = 0 (default).

18.12.5.37 ARM_USART_CPOL1

```
#define ARM_USART_CPOL1 (1UL << ARM_USART_CPOL_Pos)
```

Location: Driver_USART.h:128

CPOL = 1.

18.12.5.38 ARM_USART_CPHA_Pos

```
#define ARM_USART_CPHA_Pos 19
```

Location: Driver_USART.h:131

Position of the 0th bit of the Mode parameters Clock phase field in the ARM_USART structure.

18.12.5.39 ARM_USART_CPHA_Msk

```
#define ARM_USART_CPHA_Msk (1UL << ARM_USART_CPHA_Pos)
```

Location: Driver_USART.h:132

Positioning of the Mode parameters Clock phase field in the ARM_USART structure.

18.12.5.40 ARM_USART_CPHA0

```
#define ARM_USART_CPHA0 (0UL << ARM_USART_CPHA_Pos)
```

Location: Driver_USART.h:133

CPHA = 0 (default).

18.12.5.41 ARM_USART_CPHA1

```
#define ARM_USART_CPHA1 (1UL << ARM_USART_CPHA_Pos)
```

Location: Driver_USART.h:134

CPHA = 1.

18.12.5.42 ARM_USART_SET_DEFAULT_TX_VALUE

```
#define ARM_USART_SET_DEFAULT_TX_VALUE (0x10UL << ARM_USART_CONTROL_Pos)
```

Location: Driver_USART.h:138

Set default transmit value (synchronous receive only); arg = value.

18.12.5.43 ARM_USART_SET_IRDA_PULSE

```
#define ARM_USART_SET_IRDA_PULSE (0x11UL << ARM_USART_CONTROL_Pos)
```

Location: Driver_USART.h:139

Set IrDA Pulse in ns; arg: 0=3/16 of bit period.

18.12.5.44 ARM_USART_SET_SMART_CARD_GUARD_TIME

```
#define ARM_USART_SET_SMART_CARD_GUARD_TIME (0x12UL << ARM_USART_CONTROL_Pos)
```

Location: Driver_USART.h:140

Set smart card guard time; arg = number of bit periods.

18.12.5.45 ARM_USART_SET_SMART_CARD_CLOCK

```
#define ARM_USART_SET_SMART_CARD_CLOCK (0x13UL << ARM_USART_CONTROL_Pos)
```

Location: Driver_USART.h:141

Set smart card clock in Hz; arg: 0=Clock not generated.

18.12.5.46 ARM_USART_CONTROL_SMART_CARD_NACK

```
#define ARM_USART_CONTROL_SMART_CARD_NACK (0x14UL << ARM_USART_CONTROL_Pos)
```

Location: Driver_USART.h:142

Smart card NACK generation; arg: 0=disabled, 1=enabled.

18.12.5.47 ARM_USART_CONTROL_TX

```
#define ARM_USART_CONTROL_TX (0x15UL << ARM_USART_CONTROL_Pos)
```

Location: Driver_USART.h:143

Transmitter; arg: 0=disabled, 1=enabled.

18.12.5.48 ARM_USART_CONTROL_RX

```
#define ARM_USART_CONTROL_RX (0x16UL << ARM_USART_CONTROL_Pos)
```

Location: Driver_USART.h:144

Receiver; arg: 0=disabled, 1=enabled.

18.12.5.49 ARM_USART_CONTROL_BREAK

```
#define ARM_USART_CONTROL_BREAK (0x17UL << ARM_USART_CONTROL_Pos)
```

Location: Driver_USART.h:145

Continuous break transmission; arg: 0=disabled, 1=enabled.

18.12.5.50 ARM_USART_ABORT_SEND

```
#define ARM_USART_ABORT_SEND (0x18UL << ARM_USART_CONTROL_Pos)
```

Location: Driver_USART.h:146

Abort ARM_USART_Send.

18.12.5.51 ARM_USART_ABORT_RECEIVE

```
#define ARM_USART_ABORT_RECEIVE (0x19UL << ARM_USART_CONTROL_Pos)
```

Location: Driver_USART.h:147

Abort ARM_USART_Receive.

18.12.5.52 ARM_USART_ABORT_TRANSFER

```
#define ARM_USART_ABORT_TRANSFER (0x1AUL << ARM_USART_CONTROL_Pos)
```

Location: Driver_USART.h:148

Abort ARM_USART_Transfer.

18.12.5.53 ARM_USART_ERROR_MODE

```
#define ARM_USART_ERROR_MODE (ARM_DRIVER_ERROR_SPECIFIC - 1)
```

Location: Driver_USART.h:153

Specified mode not supported.

18.12.5.54 ARM_USART_ERROR_BAUDRATE

```
#define ARM_USART_ERROR_BAUDRATE (ARM_DRIVER_ERROR_SPECIFIC - 2)
```

Location: Driver_USART.h:154

Specified baudrate not supported.

18.12.5.55 ARM_USART_ERROR_DATA_BITS

```
#define ARM_USART_ERROR_DATA_BITS (ARM_DRIVER_ERROR_SPECIFIC - 3)
```

Location: Driver_USART.h:155

Specified number of data bits not supported.

18.12.5.56 ARM_USART_ERROR_PARITY

```
#define ARM_USART_ERROR_PARITY (ARM_DRIVER_ERROR_SPECIFIC - 4)
```

Location: Driver_USART.h:156

Specified parity not supported.

18.12.5.57 ARM_USART_ERROR_STOP_BITS

```
#define ARM_USART_ERROR_STOP_BITS (ARM_DRIVER_ERROR_SPECIFIC - 5)
```

Location: Driver_USART.h:157

Specified number of stop bits not supported.

18.12.5.58 ARM_USART_ERROR_FLOW_CONTROL

```
#define ARM_USART_ERROR_FLOW_CONTROL (ARM_DRIVER_ERROR_SPECIFIC - 6)
```

Location: Driver_USART.h:158

Specified flow control not supported.

18.12.5.59 ARM_USART_ERROR_CPOL

```
#define ARM_USART_ERROR_CPOL (ARM_DRIVER_ERROR_SPECIFIC - 7)
```

Location: Driver_USART.h:159

Specified clock polarity not supported.

18.12.5.60 ARM_USART_ERROR_CPHA

```
#define ARM_USART_ERROR_CPHA (ARM_DRIVER_ERROR_SPECIFIC - 8)
```

Location: Driver_USART.h:160

Specified clock phase not supported.

18.12.5.61 ARM_USART_EVENT_SEND_COMPLETE

```
#define ARM_USART_EVENT_SEND_COMPLETE (1UL << 0)
```

Location: Driver_USART.h:201

USART Event

Send completed; however USART may still transmit data.

18.12.5.62 ARM_USART_EVENT_RECEIVE_COMPLETE

```
#define ARM_USART_EVENT_RECEIVE_COMPLETE (1UL << 1)
```

Location: Driver_USART.h:202

Receive completed.

18.12.5.63 ARM_USART_EVENT_TRANSFER_COMPLETE

```
#define ARM_USART_EVENT_TRANSFER_COMPLETE (1UL << 2)
```

Location: Driver_USART.h:203

Transfer completed.

18.12.5.64 ARM_USART_EVENT_TX_COMPLETE

```
#define ARM_USART_EVENT_TX_COMPLETE (1UL << 3)
```

Location: Driver_USART.h:204

Transmit completed (optional).

18.12.5.65 ARM_USART_EVENT_TX_UNDERFLOW

```
#define ARM_USART_EVENT_TX_UNDERFLOW (1UL << 4)
```

Location: Driver_USART.h:205

Transmit data not available (synchronous slave).

18.12.5.66 ARM_USART_EVENT_RX_OVERFLOW

```
#define ARM_USART_EVENT_RX_OVERFLOW (1UL << 5)
```

Location: Driver_USART.h:206

Receive data overflow.

18.12.5.67 ARM_USART_EVENT_RX_TIMEOUT

```
#define ARM_USART_EVENT_RX_TIMEOUT (1UL << 6)
```

Location: Driver_USART.h:207

Receive character timeout (optional).

18.12.5.68 ARM_USART_EVENT_RX_BREAK

```
#define ARM_USART_EVENT_RX_BREAK (1UL << 7)
```

Location: Driver_USART.h:208

Break detected on receive.

18.12.5.69 ARM_USART_EVENT_RX_FRAMING_ERROR

```
#define ARM_USART_EVENT_RX_FRAMING_ERROR (1UL << 8)
```

Location: Driver_USART.h:209

Framing error detected on receive.

18.12.5.70 ARM_USART_EVENT_RX_PARITY_ERROR

```
#define ARM_USART_EVENT_RX_PARITY_ERROR (1UL << 9)
```

Location: Driver_USART.h:210

Parity error detected on receive.

18.12.5.71 ARM_USART_EVENT_CTS

```
#define ARM_USART_EVENT_CTS (1UL << 10)
```

Location: Driver_USART.h:211

CTS state changed (optional).

18.12.5.72 ARM_USART_EVENT_DSR

```
#define ARM_USART_EVENT_DSR (1UL << 11)
```

Location: Driver_USART.h:212

DSR state changed (optional).

18.12.5.73 ARM_USART_EVENT_DCD

```
#define ARM_USART_EVENT_DCD (1UL << 12)
```

Location: Driver_USART.h:213

DCD state changed (optional).

18.12.5.74 ARM_USART_EVENT_RI

```
#define ARM_USART_EVENT_RI (1UL << 13)
```

Location: Driver_USART.h:214

RI state changed (optional).

18.12.6 CMSIS USART Driver Function Documentation

18.12.6.1 ARM_USART_GetVersion

```
ARM_DRIVER_VERSION ARM_USART_GetVersion()
```

Location: Driver_USART.c:59

Get driver version.

Returns:

ARM_DRIVER_VERSION

18.12.6.2 ARM_USART_GetCapabilities

```
ARM_USART_CAPABILITIES ARM_USART_GetCapabilities()
```

Location: Driver_USART.c:64

Get driver capabilities.

Returns:

ARM_USART_CAPABILITIES

18.12.6.3 ARM_USART_Initialize

```
int32_t ARM_USART_Initialize(ARM_USART_SignalEvent_t cb_event)
```

Location: Driver_USART.c:69

Initialize USART Interface.

Returns:

execution_status

Parameters

Direction	Name	Description
in	*cb_event*	Pointer to ARM_USART_SignalEvent

18.12.6.4 ARM_USART_Uninitialize

```
int32_t ARM_USART_Uninitialize()
```

Location: Driver_USART.c:73

De-initialize USART Interface.

Returns:

execution_status

18.12.6.5 ARM_USART_PowerControl

```
int32_t ARM_USART_PowerControl(ARM_POWER_STATE state)
```

Location: Driver_USART.c:77

Control USART Interface Power.

Returns:

execution_status

Parameters

Direction	Name	Description
in	*state*	Power state

18.12.6.6 ARM_USART_Send

```
int32_t ARM_USART_Send(const void * data, uint32_t num)
```

Location: Driver_USART.c:93

Start sending data to USART transmitter.

Returns:

execution_status

Parameters

Direction	Name	Description
in	*data*	Pointer to buffer with data to send to USART transmitter
in	*num*	Number of data items to send

18.12.6.7 ARM_USART_Receive

```
int32_t ARM_USART_Receive(void * data, uint32_t num)
```

Location: Driver_USART.c:97

Start receiving data from USART receiver.

Returns:

execution_status

Parameters

Direction	Name	Description
`out`	*data*	Pointer to buffer for data to receive from USART receiver
`in`	*num*	Number of data items to receive

18.12.6.8 ARM_USART_Transfer

```
int32_t ARM_USART_Transfer(const void * data_out, void * data_in, uint32_t num)
```

Location: Driver_USART.c:101

Start sending/receiving data to/from USART transmitter/receiver.

Returns:

execution_status

Parameters

Direction	Name	Description
`in`	*data_out*	Pointer to buffer with data to send to USART transmitter
`out`	*data_in*	Pointer to buffer for data to receive from USART receiver
`in`	*num*	Number of data items to transfer

18.12.6.9 ARM_USART_GetTxCount

```
uint32_t ARM_USART_GetTxCount()
```

Location: Driver_USART.c:105

Get transmitted data count.

Returns:

number of data items transmitted

18.12.6.10 ARM_USART_GetRxCount

```
uint32_t ARM_USART_GetRxCount()
```

Location: Driver_USART.c:109

Get received data count.

Returns:

number of data items received

18.12.6.11 ARM_USART_Control

```
int32_t ARM_USART_Control(uint32_t control, uint32_t arg)
```

Location: Driver_USART.c:113

Control USART Interface.

Returns:

common execution_status and driver specific usart_execution_status

Parameters

Direction	Name	Description
in	*control*	Operation
in	*arg*	Argument of operation (optional)

18.12.6.12 ARM_USART_GetStatus

```
ARM_USART_STATUS ARM_USART_GetStatus()
```

Location: Driver_USART.c:117

Get USART status.

Returns:

USART status ARM_USART_STATUS

18.12.6.13 ARM_USART_SetModemControl

```
int32_t ARM_USART_SetModemControl(ARM_USART_MODEM_CONTROL control)
```

Location: Driver_USART.c:121

Set USART Modem Control line state.

Returns:

execution_status

Parameters

Direction	Name	Description
in	*control*	ARM_USART_MODEM_CONTROL

18.12.6.14 ARM_USART_GetModemStatus

```
ARM_USART_MODEM_STATUS ARM_USART_GetModemStatus()
```

Location: Driver_USART.c:125

Get USART Modem Status lines state.

Returns:

modem status ARM_USART_MODEM_STATUS

18.12.6.15 ARM_USART_SignalEvent

```
void ARM_USART_SignalEvent(uint32_t event)
```

Location: Driver_USART.c:129

Signal USART Events.

Returns:

none

Parameters

Direction	Name	Description
in	*event*	USART_events notification mask

CHAPTER 19

swmTrace Reference

The swmTrace library functions and macros provide a logging utility that helps you to debug an application running on the Arm Cortex-M33 core.

19.1 Summary

Macros

- swmLogVerbose : Shortcut macro for verbose logging.
- swmLogInfo : Shortcut macro for informational logging.
- swmLogWarn : Shortcut macro for warnings.
- swmLogError : Shortcut macro for errors.
- swmLogFatal : Shortcut macro for fatal errors.
- swmLogTestPass : Shortcut macro for test PASS indicators.
- swmLogTestFail : Shortcut macro for test FAIL indicators.

Functions

- swmTrace_init : Trace initialization function.
- swmTrace_txInProgress : Provides indication if transmission is in progress.
- swmTrace_printf : This provides a printf-like implementation for all possible trace mechanisms.
- swmTrace_vprintf : This provides a vprintf-like implementation for all possible trace mechanisms.
- swmTrace_getch : A method to allow characters to be passed from the logging target to the traced application.
- swmLog : A general logging method that allows us to output only trace messages if a particular log level has been selected.

19.2 swmTrace Reference Macro Definition Documentation

19.2.1 swmLogVerbose

```
#define swmLogVerbose swmLog(SWM_LOG_LEVEL_VERBOSE, __VA_ARGS__)
```

Location: swmTrace_api.h:125

Shortcut macro for verbose logging.

19.2.2 swmLogInfo

```
#define swmLogInfo swmLog(SWM_LOG_LEVEL_INFO, __VA_ARGS__)
```

Location: swmTrace_api.h:130

Shortcut macro for informational logging.

19.2.3 swmLogWarn

```
#define swmLogWarn swmLog(SWM_LOG_LEVEL_WARNING, __VA_ARGS__)
```

Location: swmTrace_api.h:135

Shortcut macro for warnings.

19.2.4 swmLogError

```
#define swmLogError swmLog(SWM_LOG_LEVEL_ERROR, __VA_ARGS__)
```

Location: swmTrace_api.h:140

Shortcut macro for errors.

19.2.5 swmLogFatal

```
#define swmLogFatal swmLog(SWM_LOG_LEVEL_FATAL, __VA_ARGS__)
```

Location: swmTrace_api.h:145

Shortcut macro for fatal errors.

19.2.6 swmLogTestPass

```
#define swmLogTestPass swmLog(SWM_LOG_TEST_PASS, __VA_ARGS__)
```

Location: swmTrace_api.h:150

Shortcut macro for test PASS indicators.

19.2.7 swmLogTestFail

```
#define swmLogTestFail swmLog(SWM_LOG_TEST_FAIL, __VA_ARGS__)
```

Location: swmTrace_api.h:155

Shortcut macro for test FAIL indicators.

19.3 SWMTRACE REFERENCE FUNCTION DOCUMENTATION

19.3.1 swmTrace_init

```
void swmTrace_init(const uint32_t * configuration, uint32_t size)
```

Location: swmTrace_api.h:79

Trace initialization function.

This method allows the tracing functions to be initialized in a general way, allowing different trace options to be supplied, depending on the type of trace library selected. configuration Consists of an array of 32-bit words that define the selected initialization options. size Indicates the number of options provided.

NOTE: The list of options can be set up as a superset of all the possible options, and only the ones required for a given trace library are used. The possible options are defined in the swmTrace_options.h file.

NOTE: The list of options can be set up as a superset of all the possible options, and only the ones required for a given trace library are used. The possible options are defined in the swmTrace_options.h file.

Parameters

Direction	Name	Description
in	*configuration*	Consists of an array of 32-bit words that define the selected initialization options.
in	*size*	Indicates the number of options provided.

19.3.2 swmTrace_txInProgress

```
bool swmTrace_txInProgress()
```

Location: swmTrace_api.h:85

Provides indication if transmission is in progress.

Returns:

True if a string is being transmitted; false otherwise.

19.3.3 swmTrace_printf

```
void swmTrace_printf(const char * sFormat, ... )
```

Location: swmTrace_api.h:93

This provides a printf-like implementation for all possible trace mechanisms.

Parameters

Direction	Name	Description
in	*sFormat*	Defines the format of the string to print. This is followed by a variable number of arguments.

19.3.4 swmTrace_vprintf

```
void swmTrace_vprintf(const char * sFormat, va_list * pParamList)
```

Location: swmTrace_api.h:102

This provides a vprintf-like implementation for all possible trace mechanisms.

Parameters

Direction	Name	Description
in	*sFormat*	Defines the format of the string to print.
in	*pParamList*	Defines a pointer to a va_list object in a form similar to vprintf.

19.3.5 swmTrace_getch

```
bool swmTrace_getch(char * ch)
```

Location: swmTrace_api.h:111

A method to allow characters to be passed from the logging target to the traced application.

Returns:

True if a valid character has been returned; false otherwise.

Parameters

Direction	Name	Description
in	*ch*	A pointer to a character object which holds the returned character.

19.3.6 swmLog

```
void swmLog(uint32_t level, const char * sFormat, ... )
```

Location: swmTrace_api.h:120

A general logging method that allows us to output only trace messages if a particular log level has been selected.

Parameters

Direction	Name	Description
`in`	*level*	The level of this log message. Only messages which have a level equal to or higher than the currently selected level are output.
`in`	*sFormat*	The format of the output string, as per printf.

CHAPTER 20

Bluetooth Low Energy Stack Abstraction

Bluetooth Low Energy Stack Abstraction Reference.

20.1 SUMMARY

Typedefs

- MsgHandlerCallback_t : BLE abstraction message handler function type.

Variables

- ble_dev_params : Structure to save BLE device parameters provided by the application.

Data Structures

- GAPM_ActivityStatus_t : GAPM activity status.
- GAP_Env_t : GAP environment.
- att_db_desc : Custom service attribute database description.
- cust_svc_desc : Custom service descriptor.
- GATT_Env_t : GATT environment.
- low_power_clock : Application defined low power clocks and sources.
- ble_device_parameter : Application defined BLE device parameters used by BLE stack.
- BASC_Env_t : Battery service client environment.
- BASS_Env_t : Battery service server environment.
- DISS_DeviceInfoField_t : Global DISS specific info structure.
- DISS_DeviceInfo_t : Global DISS info compilation structure.
- DISS_Env_t : DISS state structure.

Enumerations

- gapm_activity_state : GAPM activity state.
- gapm_state : GAPM state.
- cfm_msg_instr_enum : Instructions to send or withhold confirmation message.
- basc_app_msg_id : BASC application task messages.
- bass_app_msg_id : BASS application task messages.

Macros

- GAPM_CFG_ADDR_PUBLIC : GAPM privacy configuration identity address public.
- GAPM_CFG_ADDR_PRIVATE : GAPM privacy configuration identity address private.
- GAPM_CFG_HOST_PRIVACY : GAPM privacy configuration set to host privacy.

- GAPM_CFG_CONTROLLER_PRIVACY : GAPM privacy configuration set to controller privacy.
- GAPM_DEFAULT_ROLE : GAPM default device configuration (used in GAPM_SET_DEV_CONFIG_CMD)
- GAPM_DEFAULT_RENEW_DUR : Default duration for regenerating device address when privacy is enabled (in seconds)
- GAPM_DEFAULT_GAP_START_HDL : Default start handle for GAP service (dynamically allocated)
- GAPM_DEFAULT_GATT_START_HDL : Default start handle for GATT service (dynamically allocated)
- GAPM_DEFAULT_ATT_CFG : Default attribute database configuration.
- GAPM_DEFAULT_TX_OCT_MAX : Default value for the controller's maximum transmitted number of payload octets to be used.
- GAPM_DEFAULT_TX_TIME_MAX : Suggested value for the controller's maximum packet transmission time to be used.
- GAPM_DEFAULT_MTU_MAX : Default maximum MTU acceptable for device (L2CAP)
- GAPM_DEFAULT_MPS_MAX : Default maximum MPS (L2CAP)
- GAPM_DEFAULT_MAX_NB_LECB : Default maximum number of LE Credit based connection that can be established (L2CAP)
- GAPM_DEFAULT_AUDIO_CFG : Default LE Audio Mode Configuration.
- GAP_ROLE_MASTER : GAP master.
- GAP_ROLE_SLAVE : GAP slave.
- GAPM_DEFAULT_ADV_DATA : Default advertisement data (length, data)
- GAPM_DEFAULT_ADV_DATA_LEN : Default length of advertisement data.
- GAPM_DEFAULT_SCANRSP_DATA : Default scan response data.
- GAPM_DEFAULT_SCANRSP_DATA_LEN : Default length of scan response data.
- GAPM_DEFAULT_SCAN_INTERVAL : GAPM default StartConnectionCmd configuration (GAPM_START_CONNECTION_CMD)
- GAPM_DEFAULT_SCAN_WINDOW : Scan window to 50% of the interval.
- GAPM_DEFAULT_CON_INTV_MIN : Default connection interval and slave latency.
- GAPM_DEFAULT_CON_INTV_MAX : 25 ms (units of 1.25ms)
- GAPM_DEFAULT_CON_LATENCY : Slave latency: number of events that can be missed while maintaining a connection.
- GAPM_DEFAULT_SUPERV_TO : Default supervisory timeout.
- GAPM_DEFAULT_CE_LEN_MIN : Recommended minimum duration of connection events 40 ms.
- GAPM_DEFAULT_CE_LEN_MAX : Recommended maximum duration of connection events 40 ms.
- GAPM_DEFAULT_ADV_INTV_MIN : Advertising minimum and maximum interval.
- GAPM_DEFAULT_ADV_INTV_MAX : 40ms (64*0.625ms)
- GAPM_DEFAULT_ADV_CHMAP : Advertising channel map - 37, 38, 39.
- GATTC_DEFAULT_START_HDL : Default GATTC handles.
- GATTC_DEFAULT_END_HDL : GATTC default end handle.
- CS_SERVICE_UUID_16 : Macros to declare a (custom) service with 16, 32 and 128 bit UUID srvidx Service index uuid Service UUID.
- CS_SERVICE_UUID_32 : Custom service with 32 bit UUID.
- CS_SERVICE_UUID_128 : Custom service with 128 bit UUID.
- CS_ATT_SERVICE_128 : Standard declaration/description UUIDs in 16-byte format.
- CS_ATT_CHARACTERISTIC_128 : Custom service characteristic.
- CS_ATT_CLIENT_CHAR_CFG_128 : Custom service attribute for client characteristic configuration.
- CS_ATT_CHAR_USER_DESC_128 : Custom service attribute characteristic user description.
- CS_CHAR_UUID_16 : Macros to define characteristics with 16, 32 and 128 bit UUID attidx_char: Characteristic attribute index attidx_val: Value attribute index uuid UUID perm Permissions (see gattm_att_

desc) length Maximum length value (in bytes) data Pointer to the data structure in the application callback Function to transfer the data between the application and the GATTM.

- CS_CHAR_UUID_32 : Custom service characteristic with 32 bit UUID.
- CS_CHAR_UUID_128 : Custom service characteristic with 128 bit UUID.
- CS_CHAR_CCC : Macro to add to the characteristic a CCC attidx CCC attribute index data Pointer to the 2-byte CCC data value in the application callback Function to transfer the CCC data between the application and the GATTM.
- CS_CHAR_USER_DESC : Macro to add to the characteristic a user description attidx Description attribute index length Description length (in bytes) data Pointer to the description string (constant) callback Function to transfer the description string to the GATTM.
- MIN : Macro to Find Minimum.
- HCI_VS_RF_CW_ENABLE_CMD_CODE : Vendor specific to enable CW (RX or TX)
- HCI_VS_RF_CW_DISABLE_CMD_CODE : Vendor specific to enable CW (RX or TX)
- HCI_VS_RF_OUTPUT_PWR_CMD_CODE : Vendor specific to set desired output power.

Functions

- GAP_Initialize : Initialize GAP environment.
- GAP_GetEnv : Get GAP environment.
- GAP_GetProfileAddedTaskId : Get GAP environment added profiles task identifiers.
- GAP_IsAddrPrivateResolvable : Check address if its private resolvable.
- GAP_AddAdvData : Add GAP advertise data.
- GAPM_ResetCmd : Perform GAPM reset with specified GAPM operation.
- GAPM_SoftwareReset : Perform GAPM software reset.
- GAPM_PlatformReset : Perform GAPM platform reset.
- GAPM_SetDevConfigCmd : Prepare and send set the device configuration.
- GAPM_GetDeviceConfig : get Device configuration
- GAPM_ProfileTaskAddCmd : Prepare and Send GAPM_PROFILE_TASK_ADD_CMD to the Bluetooth stack.
- GAPM_GetProfileAddedCount : Get number of profiles added by the Bluetooth stack.
- GAPM_LepsmRegisterCmd : Prepare and send GAPM_LEPSM_REGISTER_CMD to the Bluetooth stack.
- GAPM_GenRandAddrCmd : Prepare and send GAPM_GEN_RAND_ADDR_CMD to the Bluetooth stack.
- GAPM_ResolvAddrCmd : Prepare and send GAPM_RESOLV_ADDR_CMD to the Bluetooth stack.
- GAPM_ActivityCreateAdvCmd : Prepare and send GAPM_ACTIVITY_CREATE_CMD to create an advertising activity.
- GAPM_ActivityCreateScanCmd : Prepare and send GAPM_ACTIVITY_CREATE_CMD to create scan activity.
- GAPM_ActivityCreateInitCmd : Prepare and send GAPM_ACTIVITY_CREATE_CMD to create an initiating activity.
- GAPM_ActivityCreatePeriodSyncCmd : Prepare and send GAPM_ACTIVITY_CREATE_CMD to create a periodic sync activity.
- GAPM_AdvActivityStart : Prepare and send GAPM_ACTIVITY_START_CMD to start an advertising activity.
- GAPM_InitActivityStart : Prepare and send GAPM_ACTIVITY_START_CMD to start an initiating activity.
- GAPM_ScanActivityStart : Prepare and send GAPM_ACTIVITY_START_CMD to start a scanning activity.
- GAPM_PerSyncActivityStart : Prepare and send GAPM_ACTIVITY_START_CMD to start a periodic sync activity.
- GAPM_ActivityStartCmd : Prepare and send GAPM_ACTIVITY_START_CMD to start an activity.
- GAPM_ActivityStop : Prepare and send GAPM_ACTIVITY_STOP_CMD to stop a specified activity.
- GAPM_ActivityStopAll : Prepare and send GAPM_ACTIVITY_STOP_CMD to stop all existing activities.

- GAPM_DeleteActivity : Prepare and send GAPM_ACTIVITY_DELETE_CMD to delete a specified activity.
- GAPM_DeleteAllActivities : Prepare and send GAPM_ACTIVITY_DELETE_CMD to delete all the activities.
- GAPM_ActivityStopCmd : Prepare and send GAPM_ACTIVITY_STOP_CMD to stop a specified activity.
- GAPM_DeleteActivityCmd : Prepare and send GAPM_ACTIVITY_DELETE_CMD to delete activity/activities.
- GAPM_SetAdvDataCmd : Prepare and send GAPM_SET_ADV_DATA_CMD to set an advertising data or scan response data or periodic advertising data.
- GAPM_MsgHandler : Message handler for GAPM events.
- GAPC_ParamUpdateCmd : Prepare and send GAPC_PARAM_UPDATE_CMD to perform an update on parameters.
- GAPC_ParamUpdateCfm : Prepare and send GAPC_PARAM_UPDATE_CFM to accept or reject connection parameters.
- GAPC_ConnectionCfm : Prepare and send GAPC_CONNECTION_CFM in a response to connection request.
- GAPC_DisconnectCmd : Prepare and send GAPC_DISCONNECT_CMD to disconnect link.
- GAPC_IsConnectionActive : Check if the specified connection is active.
- GAPC_DisconnectAll : Prepare and send GAPC_DISCONNECT_CMD to disconnect all the links.
- GAPC_ConnectionCount : Get the number of active connection count.
- GAPC_MasterConnectionCount : Get the master connection count.
- GAPC_SlaveConnectionCount : Get the slave connection count.
- GAPC_GetConnectionInfo : Get the connection information of a specified connection ID.
- GAPC_GetDevInfoCfm_Name : Prepare and send GAPC_GET_DEV_INFO_CFM to send the device name.
- GAPC_GetDevInfoCfm_Appearance : Prepare and send GAPC_GET_DEV_INFO_CFM to send the device appearance.
- GAPC_GetDevInfoCfm_SlvPrefParams : Prepare and send GAPC_GET_DEV_INFO_CFM to send device slave preferred parameters.
- GAPC_GetDevInfoCfm : Prepare and send GAPC_GET_DEV_INFO_CFM to send requested information.
- GAPC_SetDevInfoCfm : Prepare and send GAPC_SET_DEV_INFO_CFM to confirm if the requested device information was written.
- GAPC_BondCfm : GAPC bond and encryption operations.
- GAPC_EncryptCmd : Prepare and send GAPC_ENCRYPT_CMD to request encrypting the connection link.
- GAPC_EncryptCfm : Prepare and send GAPC_ENCRYPT_CFM to confirm an encryption request.
- GAPC_IsBonded : Check if the connection is bonded.
- GAPC_GetBondInfo : Get bond information for the connection.
- GAPC_BondCmd : Prepare and send GAPC_BOND_CMD to initiate a bond procedure.
- GAPC_AddRecordToBondList : Add record to bond list.
- GAPC_MsgHandler : GAPC message handler.
- GAPM_ListSetWlCmd : Prepare and send GAPM_LIST_SET_CMD to set the white list content.
- GAPM_ListSetRalCmd : Prepare and send GAPM_LIST_SET_CMD to set the resolving list content.
- GAPM_IsIRKValid : Check if IRK is valid.
- WhiteList_ResolvableList_Update : Update the white list and the resolve List.
- GAPM_GetWhitelistNumDev : Get number of devices in whitelist.
- GAPM_SetWhitelistNumDev : Set the number of devices in whitelist.
- GAPC_SetPhyCmd : GAPC PHY management operations.
- GAPC_CteTxCfgCmd : GAPC constant Tone extension operations.
- GAPC_CteRxCfgCmd : Prepare and send GAPC_CTE_RX_CFG_CMD to configure CTE RX.
- GAPC_CteReqCtrlCmd : Prepare and send GAPC_CTE_REQ_CTRL_CMD to control CTE requests.
- GAPC_CteRspCtrlCmd : Prepare and send GAPC_CTE_RSP_CTRL_CMD to control CTE reception.
- GAPC_GetInfoCmd : GAPC local and peer device information operations.
- GAPM_PerAdvCteTxCmd : Prepare and send GAPM_PER_ADV_CTE_TX_CTL_CMD to control CTE transmission in a periodic advertising activity.

- GAPM_PerAdvReportCtrlCmd : Prepare and send GAPM_PER_ADV_REPORT_CTRL_CMD to control reception of periodic advertising report in a periodic advertising sync activity.
- GAPM_PerSyncIQSamplingCtrlCmd : Prepare and send GAPM_PER_SYNC_IQ_SAMPLING_CTRL_CMD to control capturing IQ samples from the constant tone extension of periodic advertising packets.
- GATT_Initialize : GATT initialization.
- GATT_GetEnv : Get GATT environment.
- GATT_SetEnvData : Set GATT environment data.
- GATT_GetMaxCustomServiceNumber : Get the maximum number of custom services in GATT.
- GATTM_GetServiceAddedCount : Get GATT added services count.
- GATTM_ResetServiceAttributeDatabaseID : Resets GATT services attribute database ID.
- GATTM_AddAttributeDatabase : Add service and attributes to the Bluetooth stack.
- GATTM_GetHandle : Get the handle for an attribute.
- GATTM_MsgHandler : Handle GATTM messages.
- GATTC_Initialize : GATTC initialization.
- GATTC_DiscByUUIDSvc : GATTC services discovery by UUID.
- GATTC_DiscAllSvc : GATTC all services discovery.
- GATTC_DiscAllChar : GATTC all characteristics discovery.
- GATTC_SendEvtCmd : GATTC send event.
- GATTC_SendEvtCfm : GATTC send event confirmation.
- GATTC_MtuExchange : Prepare and send GATTC_EXC_MTU_CMD to start MTU exchange procedure.
- GATTC_ReadReqInd : GATTC handle read request indication.
- GATTC_WriteReqInd : GATTC handle write request indication.
- GATTC_AttInfoReqInd : GATTC request attribute information.
- GATTC_MsgHandler : Handle GATTM messages.
- Device_BLE_Public_Address_Read : Read the Bluetooth public address to ble_public_addr.
- Device_BLE_Param_Get : Read BLE device parameters.
- rand_func : Generate a pseudo-random number.
- set_app_rand_func : Sets random number generator function to be used by the BLE stack when calling rand_func()
- srand_func : Seeds pseudo-random number generator used by function rand()
- set_app_rand_seed_val : Sets seed value to be used by the BLE stack when calling srand_func()
- default_rf_rssi_func : Default RF to RSSI conversion function to be used by the BLE stack when calling Device_RF_RSSI_Convert().
- set_app_rf_rssi_func : Sets RF to RSSI conversion function to be used by the BLE stack when calling Device_RF_RSSI_Convert()
- MsgHandler_GetTaskAppDesc : Provide an application task descriptor with default handler set to the MsgHandler_Notify function.
- MsgHandler_Add : Add/subscribe a message handler callback function.
- MsgHandler_Notify : Notify the callback functions associated with the msg_id.
- BASC_Initialize : Initialize BASC environment.
- BASC_EnableReq : Enable the client role of the BAS.
- BASC_ReadInfoReq : Prepare and send BASC_READ_INFO_REQ to read value from peer.
- BASC_BattLevelNtfCfgReq : Send a request to change notification configuration.
- BASC_RequestBattLevelOnTimeout : Send a request for periodic battery level update.
- BASC_GetLastBatteryLevel : Read the most recent battery level.
- BASC_GetEnv : Get the BASC environment.
- BASC_MsgHandler : Handle the BASC events.
- BASS_Initialize : Initialize BASS environment.
- BASS_NotifyOnTimeout : BASS configure periodic notification.

- BASS_NotifyOnBattLevelChange : BASS notify peers on battery level change.
- BASS_ProfileTaskAddCmd : Add a battery profile in kernel.
- BASS_EnableReq : Enable the server role in BAS.
- BASS_BattLevelUpdReq : Prepare and send BASS_BATT_LEVEL_UPD_REQ for battery level notification.
- BASS_GetEnv : Get the BASS environment.
- BASS_IsAdded : Check if the BASS is added successfully.
- BASS_MsgHandler : Handle BASS messages.
- DISS_Initialize : Initialize the device information server service environment and configure message handlers.
- DISS_ProfileTaskAddCmd : Send a request to add the device information service in kernel and database.
- DISS_EnvGet : Return a reference to the internal DISS environment structure.
- DISS_IsAdded : Return a boolean indication if the DISS server has been successfully added to the services database.
- DISS_MsgHandler : Handle all the events related to the device information service server.
- DISS_DeviceInfoValueReqInd : Handle the indication when the peer device requests a value.

20.2 Detailed Description

This reference chapter presents a detailed description of all the support provided for the Bluetooth Low Energy Stack Abstraction. This reference includes calling parameters, returned values, and assumptions.

20.3 Bluetooth Low Energy Stack Abstraction Typedef Documentation

20.3.1 MsgHandlerCallback_t

```
typedef void(* MsgHandlerCallback_t
```

Location: msg_handler.h:33

BLE abstraction message handler function type.

20.4 Bluetooth Low Energy Stack Abstraction Variable Documentation

20.4.1 ble_dev_params

```
struct ble_device_parameter ble_dev_params
```

Location: ble_protocol_support.h:1

Structure to save BLE device parameters provided by the application.

20.5 Bluetooth Low Energy Stack Abstraction Data Structures Type Documentation

20.5.1 GAPM_ActivityStatus_t

Location: ble_gap.h:137

GAPM activity status.

Data Fields

Type	*Name*	Description
uint8_t	*actv_idx*	Activity identifier.
enum gapm_actv_type	*type*	GAPM activity type.
enum gapm_activity_state	*state*	GAPM activity state.
bool	*advDataSet*	True if advertisement data is set.
bool	*scanRspDataSet*	True if scan response data is set.

20.5.2 GAP_Env_t

Location: ble_gap.h:149

GAP environment.

Data Fields

Type	*Name*	Description
enum gapm_state	*gapmState*	State of GAPM.
uint16_t	*profileAddedCount*	**of standard BLE profile added**
struct gapm_set_dev_config_cmd	*deviceConfig*	Device configuration for GAPM_SET_DEV_CONFIG.

struct gapc_connection_req_ind	*connection[APP_MAX_NB_CON]*	GAPC_CONNECTION_REQ_IND information of connections.
BondInfo_t	*bondInfo[APP_MAX_NB_CON]*	Bond information of connections.
GAPM_ActivityStatus_t *	*actv[APP_MAX_NB_ACTIVITY]*	GAPM activity status of activities.
uint16_t	*profileAddedTaskId [APP_MAX_NB_PROFILES]*	Task IDs for added profiles.

20.5.3 att_db_desc

Location: ble_gatt.h:151

Custom service attribute database description.

Data Fields

Type	*Name*	Description
uint16_t	*att_idx*	Convert into uint8_t.
struct gattm_att_desc	*att*	GATTM Attribute description structure.
bool	*is_service*	True if it is service.
uint16_t	*length*	Total number of characteristics to add.
void *	*data*	Pointer to data.
uint8_t(*	*callback)(uint8_t conidx, uint16_t attidx, uint16_t handle, uint8_t *toData, const uint8_t *fromData, uint16_t lenData, uint16_t operation, uint8_t hl_status, uint8_t *cfm_msg_instr)*	Pointer to callback function.

20.5.4 cust_svc_desc

Location: ble_gatt.h:170

Custom service descriptor.

Data Fields

Type	*Name*	Description
uint16_t	*cust_svc_start_hdl*	Start handle array of custom services in the stack's attribute database.
const struct att_db_desc *	*cust_svc_att_db*	Custom services attribute database.
uint16_t	*cust_svc_att_db_len*	Custom service attribute database length.

20.5.5 GATT_Env_t

Location: ble_gatt.h:181

GATT environment.

Data Fields

Type	*Name*	Description
uint16_t	*addedSvcCount*	Counter of successfully added custom service/s in stack.
uint16_t *	*discSvcCount*	Counter of discovered service/s per connection id.
cust_svc_desc *	*cust_svc_db*	Custom service database array.
uint16_t	*max_cust_svc*	Maximum number of custom services.
const struct att_db_desc *	*att_db*	Attribute database specific to related custom service.
uint16_t	*att_db_len*	Length of attribute database specific to related custom service.

20.5.6 low_power_clock

Location: ble_protocol_support.h:46

Application defined low power clocks and sources.

Data Fields

Type	*Name*	Description
uint8_t	*low_pwr_clk_xtal32*	Value of LPCLK_SRC_XTAL32 from application.
uint8_t	*low_pwr_clk_rc32*	Value of LPCLK_SRC_RC32 from application.
uint8_t	*low_pwr_standby_clk_src*	Value of LPCLK_STANDBYCLK_SRC from application.

20.5.7 ble_device_parameter

Location: ble_protocol_support.h:56

Application defined BLE device parameters used by BLE stack.

Data Fields

Type	*Name*	Description
uint32_t	*low_pwr_clk_accuracy*	Low power clock accuracy in ppm.
uint32_t	*twosc*	TWOSC us.
uint32_t	*rf_tx_power_dbm*	The default TX power in dbm.
struct low_power_clock	*low_pwr_clk*	Low power clock selection.

20.5.8 BASC_Env_t

Location: ble_basc.h:36

Battery service client environment.

Data Fields

Type	*Name*	Description
struct gapm_profile_added_ind	*profile_added_ind*	Profile service handle.
uint32_t	*battLevelReqTimeout*	Timeout for periodic battery level request.
uint8_t	*bas_nb[APP_MAX_NB_CON]*	Number of battery instances [1,BASC_NB_BAS_INSTANCES_MAX].
bool	*enabled[APP_MAX_NB_CON]*	The flag that indicates that service has been enabled.
uint8_t	*batt_lvl[APP_MAX_NB_CON][BASC_NB_BAS_INSTANCES_MAX]*	Battery level characteristic value.
uint8_t	*ntf_cfg[APP_MAX_NB_CON][BASC_NB_BAS_INSTANCES_MAX]*	Notification configuration.
uint8_t	*req_ntf_cfg[APP_MAX_NB_CON][BASC_NB_BAS_INSTANCES_MAX]*	Control notification is required or not.
struct prf_char_pres_fmt	*char_pres_format[APP_MAX_NB_CON][BASC_NB_BAS_INSTANCES_MAX]*	Battery level characteristic presentation format descriptor value structure
struct bas_content	*bas[APP_MAX_NB_CON][BASC_NB_BAS_INSTANCES_MAX]*	Battery service content structure.

20.5.9 BASS_Env_t

Location: ble_bass.h:37

Battery service server environment.

Data Fields

Type	*Name*	Description
uint8_t	*bas_nb*	Number of battery instances [1,BASS_NB_BAS_INSTANCES_MAX].
uint8_t	*batt_ntf_cfg[APP_MAX_NB_CON]*	The current value of CCCD of battery value that has been set by the client device.
bool	*enabled[APP_MAX_NB_CON]*	The flag that indicates that service has been enabled.
uint32_t	*battLevelNotificationTimeout*	Timeout value for sending batter level notification.
bool	*battLevelNotificationTimerEnqueued*	Flag that tells whether the timer for battery level notification is enabled or not.
uint32_t	*battLevelMonitoringTimeout*	Timeout value for battery level monitoring.
bool	*battLevelMonitoringTimerEnqueued*	Flag that tells whether the timer for monitoring battery level is enabled or not.
uint8_t	*lastBattLevel[BASS_NB_BAS_INSTANCES_MAX]*	Most recent value of battery level.
uint8_t(*	*readBattLevelCallback)(uint8_t bas_nb)*	Pointer to an application function that returns the battery level.
struct gapm_profile_added_ind	*profile_added_ind*	Profile service handle.

20.5.10 DISS_DeviceInfoField_t

Location: ble_diss.h:37

Global DISS specific info structure.

Data Fields

Type	*Name*	Description
uint8_t	*len*	Length of data.
uint8_t *	*data*	Pointer to data.

20.5.11 DISS_DeviceInfo_t

Location: ble_diss.h:46

Global DISS info compilation structure.

Data Fields

Type	*Name*	Description
struct DISS_DeviceInfoField_t	*MANUFACTURER_NAME*	Manufacturer name characteristic.
struct DISS_DeviceInfoField_t	*MODEL_NB_STR*	Model number string characteristic.
struct DISS_DeviceInfoField_t	*SERIAL_NB_STR*	Serial number string characteristic.
struct DISS_DeviceInfoField_t	*FIRM_REV_STR*	Firmware revision string characteristic.
struct DISS_DeviceInfoField_t	*SYSTEM_ID*	System ID characteristic.
struct DISS_DeviceInfoField_t	*HARD_REV_STR*	Hardware revision string characteristic.
struct DISS_DeviceInfoField_t	*SW_REV_STR*	Software revision string characteristic.
struct DISS_DeviceInfoField_t	*IEEE*	IEEE 11073-20601 regulatory certification data characteristic.
struct DISS_DeviceInfoField_t	*PNP*	PnP ID characteristic.

20.5.12 DISS_Env_t

Location: ble_diss.h:63

DISS state structure.

Data Fields

Type	*Name*	Description
uint16_t	*features*	DISS features that will be enabled. See diss_task.h for reference
bool	*serviceAdded*	The flag that indicates that service has been added.
const struct DISS_DeviceInfo_t *	*deviceInfo*	DISS device information compilation structure arm.

20.6 Bluetooth Low Energy Stack Abstraction Enumeration Type Documentation

20.6.1 gapm_activity_state

Location: ble_gap.h:115

GAPM activity state.

Members

- ACTIVITY_STATE_NOT_CREATED = 0

  GAPM Activity state value when activity state not created.

- ACTIVITY_STATE_CREATING

  GAPM Activity state value when start creating activity.

- ACTIVITY_STATE_NOT_STARTED

  GAPM Activity state value when not started.

- ACTIVITY_STATE_STARTING

GAPM Activity state value when activity state is starting.

- `ACTIVITY_STATE_STARTED`

  GAPM Activity state value when activity state is started.

20.6.2 gapm_state

Location: ble_gap.h:127

GAPM state.

Members

- `GAPM_STATE_INITIAL = 0`

  Initial state, before a GAPM_RESET is performed.

- `GAPM_STATE_RESET`

  After GAPM_RESET, before the device is configured.

- `GAPM_STATE_READY`

  After the device is configured with GAPM_SetDeviceConfig.

20.6.3 cfm_msg_instr_enum

Location: ble_gatt.h:142

Instructions to send or withhold confirmation message.

Members

- `CFM_MSG_INSTR_TO_SEND = 0x00`
- `CFM_MSG_INSTR_NOT_TO_SEND`

20.6.4 basc_app_msg_id

Location: ble_basc.h:70

BASC application task messages.

Members

- `BASC_BATT_LEVEL_REQ_TIMEOUT = TASK_FIRST_MSG(TASK_ID_BASC) + 50`

  BASC battery level request timeout id.

20.6.5 bass_app_msg_id

Location: ble_bass.h:64

BASS application task messages.

Members

- `BASS_BATT_LEVEL_NTF_TIMEOUT = TASK_FIRST_MSG(TASK_ID_BASS) + 50`

  BASS battery level notification timeout ID.

- `BASS_BATT_MONITORING_TIMEOUT`

  BASS battery level monitor timeout ID.

- `BASS_BATT_LEVEL_CHANGED`

  BASS battery level change ID.

20.7 Bluetooth Low Energy Stack Abstraction Macro Definition Documentation

20.7.1 GAPM_CFG_ADDR_PUBLIC

```
#define GAPM_CFG_ADDR_PUBLIC (0 << GAPM_PRIV_CFG_PRIV_ADDR_POS)
```

Location: ble_gap.h:36

GAPM privacy configuration identity address public.

20.7.2 GAPM_CFG_ADDR_PRIVATE

```
#define GAPM_CFG_ADDR_PRIVATE (1 << GAPM_PRIV_CFG_PRIV_ADDR_POS)
```

Location: ble_gap.h:39

GAPM privacy configuration identity address private.

20.7.3 GAPM_CFG_HOST_PRIVACY

```
#define GAPM_CFG_HOST_PRIVACY (0 << GAPM_PRIV_CFG_PRIV_EN_POS)
```

Location: ble_gap.h:42

GAPM privacy configuration set to host privacy.

20.7.4 GAPM_CFG_CONTROLLER_PRIVACY

```
#define GAPM_CFG_CONTROLLER_PRIVACY (1 << GAPM_PRIV_CFG_PRIV_EN_POS)
```

Location: ble_gap.h:45

GAPM privacy configuration set to controller privacy.

20.7.5 GAPM_DEFAULT_ROLE

```
#define GAPM_DEFAULT_ROLE GAP_ROLE_ALL
```

Location: ble_gap.h:52

GAPM default device configuration (used in GAPM_SET_DEV_CONFIG_CMD)

Default device role is both peripheral and central

20.7.6 GAPM_DEFAULT_RENEW_DUR

```
#define GAPM_DEFAULT_RENEW_DUR 60
```

Location: ble_gap.h:55

Default duration for regenerating device address when privacy is enabled (in seconds)

20.7.7 GAPM_DEFAULT_GAP_START_HDL

```
#define GAPM_DEFAULT_GAP_START_HDL 0
```

Location: ble_gap.h:57

Default start handle for GAP service (dynamically allocated)

20.7.8 GAPM_DEFAULT_GATT_START_HDL

```
#define GAPM_DEFAULT_GATT_START_HDL 0
```

Location: ble_gap.h:59

Default start handle for GATT service (dynamically allocated)

20.7.9 GAPM_DEFAULT_ATT_CFG

```
#define GAPM_DEFAULT_ATT_CFG 0
```

Location: ble_gap.h:60

Default attribute database configuration.

20.7.10 GAPM_DEFAULT_TX_OCT_MAX

```
#define GAPM_DEFAULT_TX_OCT_MAX 0xfb
```

Location: ble_gap.h:63

Default value for the controller's maximum transmitted number of payload octets to be used.

20.7.11 GAPM_DEFAULT_TX_TIME_MAX

```
#define GAPM_DEFAULT_TX_TIME_MAX (14 * 8 \
                                  + \
                                  GAPM_DEFAULT_TX_OCT_MAX \
                                  * 8)
```

Location: ble_gap.h:66

Suggested value for the controller's maximum packet transmission time to be used.

20.7.12 GAPM_DEFAULT_MTU_MAX

```
#define GAPM_DEFAULT_MTU_MAX 0x200
```

Location: ble_gap.h:71

Default maximum MTU acceptable for device (L2CAP)

20.7.13 GAPM_DEFAULT_MPS_MAX

```
#define GAPM_DEFAULT_MPS_MAX 0x200
```

Location: ble_gap.h:72

Default maximum MPS (L2CAP)

20.7.14 GAPM_DEFAULT_MAX_NB_LECB

```
#define GAPM_DEFAULT_MAX_NB_LECB 0
```

Location: ble_gap.h:75

Default maximum number of LE Credit based connection that can be established (L2CAP)

20.7.15 GAPM_DEFAULT_AUDIO_CFG

```
#define GAPM_DEFAULT_AUDIO_CFG 0
```

Location: ble_gap.h:76

Default LE Audio Mode Configuration.

20.7.16 GAP_ROLE_MASTER

```
#define GAP_ROLE_MASTER 0
```

Location: ble_gap.h:77

GAP master.

20.7.17 GAP_ROLE_SLAVE

```
#define GAP_ROLE_SLAVE 1
```

Location: ble_gap.h:78

GAP slave.

20.7.18 GAPM_DEFAULT_ADV_DATA

```
#define GAPM_DEFAULT_ADV_DATA { 7, GAP_AD_TYPE_COMPLETE_NAME, \
                                'O', 'N', ' ', 'B', 'L', 'E', \
                                3, GAP_AD_TYPE_MANU_SPECIFIC_DATA, \
                                0x62, 0x3 }
```

Location: ble_gap.h:80

Default advertisement data (length, data)

20.7.19 GAPM_DEFAULT_ADV_DATA_LEN

```
#define GAPM_DEFAULT_ADV_DATA_LEN 12
```

Location: ble_gap.h:84

Default length of advertisement data.

20.7.20 GAPM_DEFAULT_SCANRSP_DATA

```
#define GAPM_DEFAULT_SCANRSP_DATA GAPM_DEFAULT_ADV_DATA
```

Location: ble_gap.h:85

Default scan response data.

20.7.21 GAPM_DEFAULT_SCANRSP_DATA_LEN

```
#define GAPM_DEFAULT_SCANRSP_DATA_LEN GAPM_DEFAULT_ADV_DATA_LEN
```

Location: ble_gap.h:86

Default length of scan response data.

20.7.22 GAPM_DEFAULT_SCAN_INTERVAL

```
#define GAPM_DEFAULT_SCAN_INTERVAL 100
```

Location: ble_gap.h:89

GAPM default StartConnectionCmd configuration (GAPM_START_CONNECTION_CMD)

Default scan interval 62.5ms

20.7.23 GAPM_DEFAULT_SCAN_WINDOW

```
#define GAPM_DEFAULT_SCAN_WINDOW 50
```

Location: ble_gap.h:90

Scan window to 50% of the interval.

20.7.24 GAPM_DEFAULT_CON_INTV_MIN

```
#define GAPM_DEFAULT_CON_INTV_MIN 20
```

Location: ble_gap.h:93

Default connection interval and slave latency.

25 ms (units of 1.25ms)

20.7.25 GAPM_DEFAULT_CON_INTV_MAX

```
#define GAPM_DEFAULT_CON_INTV_MAX 20
```

Location: ble_gap.h:94

25 ms (units of 1.25ms)

20.7.26 GAPM_DEFAULT_CON_LATENCY

```
#define GAPM_DEFAULT_CON_LATENCY 0
```

Location: ble_gap.h:96

Slave latency: number of events that can be missed while maintaining a connection.

20.7.27 GAPM_DEFAULT_SUPERV_TO

```
#define GAPM_DEFAULT_SUPERV_TO 300
```

Location: ble_gap.h:99

Default supervisory timeout.

3 seconds

20.7.28 GAPM_DEFAULT_CE_LEN_MIN

```
#define GAPM_DEFAULT_CE_LEN_MIN 2 * GAPM_DEFAULT_CON_INTV_MIN
```

Location: ble_gap.h:101

Recommended minimum duration of connection events 40 ms.

20.7.29 GAPM_DEFAULT_CE_LEN_MAX

```
#define GAPM_DEFAULT_CE_LEN_MAX 2 * GAPM_DEFAULT_CON_INTV_MAX
```

Location: ble_gap.h:103

Recommended maximum duration of connection events 40 ms.

20.7.30 GAPM_DEFAULT_ADV_INTV_MIN

```
#define GAPM_DEFAULT_ADV_INTV_MIN 64
```

Location: ble_gap.h:106

Advertising minimum and maximum interval.

40ms (64*0.625ms)

20.7.31 GAPM_DEFAULT_ADV_INTV_MAX

```
#define GAPM_DEFAULT_ADV_INTV_MAX 64
```

Location: ble_gap.h:107

40ms (64*0.625ms)

20.7.32 GAPM_DEFAULT_ADV_CHMAP

```
#define GAPM_DEFAULT_ADV_CHMAP 0x07
```

Location: ble_gap.h:110

Advertising channel map - 37, 38, 39.

20.7.33 GATTC_DEFAULT_START_HDL

```
#define GATTC_DEFAULT_START_HDL 0x0001
```

Location: ble_gatt.h:35

Default GATTC handles.

GATTC default start handle

20.7.34 GATTC_DEFAULT_END_HDL

```
#define GATTC_DEFAULT_END_HDL 0xFFFF
```

Location: ble_gatt.h:36

GATTC default end handle.

20.7.35 CS_SERVICE_UUID_16

```
#define CS_SERVICE_UUID_16 { srvidx, { uuid, PERM(SVC_UUID_LEN, UUID_16), 0, 0 }, true,
0, NULL, NULL }
```

Location: ble_gatt.h:42

Macros to declare a (custom) service with 16, 32 and 128 bit UUID srvidx Service index uuid Service UUID.

Custom service with 16 bit UUID

20.7.36 CS_SERVICE_UUID_32

```
#define CS_SERVICE_UUID_32 { srvidx, { uuid, PERM(SVC_UUID_LEN, UUID_32), 0, 0 }, true,
0, NULL, NULL }
```

Location: ble_gatt.h:45

Custom service with 32 bit UUID.

20.7.37 CS_SERVICE_UUID_128

```
#define CS_SERVICE_UUID_128 { srvidx, { uuid, PERM(SVC_UUID_LEN, UUID_128), 0, 0 }, true,
0, NULL, NULL }
```

Location: ble_gatt.h:48

Custom service with 128 bit UUID.

20.7.38 CS_ATT_SERVICE_128

```
#define CS_ATT_SERVICE_128 { 0x00, 0x28, 0x00, 0x00, 0x00, 0x00, 0x00, \
                                     0x00, 0x00, 0x00, 0x00, 0x00, 0x00, \
                                     0x00, 0x00, 0x00 }
```

Location: ble_gatt.h:52

Standard declaration/description UUIDs in 16-byte format.

Custom service attribute

20.7.39 CS_ATT_CHARACTERISTIC_128

```
#define CS_ATT_CHARACTERISTIC_128 { 0x03, 0x28, 0x00, 0x00, 0x00, 0x00, 0x00, \
                                    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, \
                                    0x00, 0x00, 0x00 }
```

Location: ble_gatt.h:55

Custom service characteristic.

20.7.40 CS_ATT_CLIENT_CHAR_CFG_128

```
#define CS_ATT_CLIENT_CHAR_CFG_128 { 0x02, 0x29, 0x00, 0x00, 0x00, 0x00, 0x00, \
                                     0x00, 0x00, 0x00, 0x00, 0x00, 0x00, \
                                     0x00, 0x00, 0x00 }
```

Location: ble_gatt.h:59

Custom service attribute for client characteristic configuration.

20.7.41 CS_ATT_CHAR_USER_DESC_128

```
#define CS_ATT_CHAR_USER_DESC_128 { 0x01, 0x29, 0x00, 0x00, 0x00, 0x00, 0x00, \
                                    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, \
                                    0x00, 0x00, 0x00 }
```

Location: ble_gatt.h:63

Custom service attribute characteristic user description.

20.7.42 CS_CHAR_UUID_16

```
#define CS_CHAR_UUID_16 { attidx_char, { CS_ATT_CHARACTERISTIC_128, PERM(RD, ENABLE), 0,
0 }, false, 0, NULL, NULL }, \
    { attidx_val, { uuid, perm, length, PERM(RI, ENABLE) | PERM(UUID_LEN, UUID_16) },
false, length, data, callback }
```

Location: ble_gatt.h:83

Macros to define characteristics with 16, 32 and 128 bit UUID attidx_char: Characteristic attribute index attidx_val: Value attribute index uuid UUID perm Permissions (see gattm_att_desc) length Maximum length value (in bytes) data

Pointer to the data structure in the application callback Function to transfer the data between the application and the GATTM.

Custom service characteristic with 16 bit UUID

20.7.43 CS_CHAR_UUID_32

```
#define CS_CHAR_UUID_32 { attidx_char, { CS_ATT_CHARACTERISTIC_128, PERM(RD, ENABLE), 0,
0 }, false, 0, NULL, NULL }, \
    { attidx_val, { uuid, perm, length, PERM(RI, ENABLE) | PERM(UUID_LEN, UUID_32) },
false, length, data, callback }
```

Location: ble_gatt.h:95

Custom service characteristic with 32 bit UUID.

20.7.44 CS_CHAR_UUID_128

```
#define CS_CHAR_UUID_128 { attidx_char, { CS_ATT_CHARACTERISTIC_128, PERM(RD, ENABLE), 0,
0 }, false, 0, NULL, NULL }, \
    { attidx_val, { uuid, perm, length, PERM(RI, ENABLE) | PERM(UUID_LEN, UUID_128) },
false, length, data, callback }
```

Location: ble_gatt.h:106

Custom service characteristic with 128 bit UUID.

20.7.45 CS_CHAR_CCC

```
#define CS_CHAR_CCC { attidx, { CS_ATT_CLIENT_CHAR_CFG_128, PERM(RD, ENABLE) | PERM
(WRITE_REQ, ENABLE), 0, PERM(RI, ENABLE) }, false, 2, \
      data, \
      callback }
```

Location: ble_gatt.h:114

Macro to add to the characteristic a CCC attidx CCC attribute index data Pointer to the 2-byte CCC data value in the application callback Function to transfer the CCC data between the application and the GATTM.

20.7.46 CS_CHAR_USER_DESC

```
#define CS_CHAR_USER_DESC { attidx, { CS_ATT_CHAR_USER_DESC_128, PERM(RD, ENABLE),
length, PERM(RI, ENABLE) }, false, length, data, callback }
```

Location: ble_gatt.h:124

Macro to add to the characteristic a user description attidx Description attribute index length Description length (in bytes) data Pointer to the description string (constant) callback Function to transfer the description string to the GATTM.

20.7.47 MIN

```
#define MIN (((a) < (b)) ? (a) : (b))
```

Location: ble_gatt.h:128

Macro to Find Minimum.

20.7.48 HCI_VS_RF_CW_ENABLE_CMD_CODE

```
#define HCI_VS_RF_CW_ENABLE_CMD_CODE 0x01
```

Location: ble_protocol_support.h:33

Vendor specific to enable CW (RX or TX)

20.7.49 HCI_VS_RF_CW_DISABLE_CMD_CODE

```
#define HCI_VS_RF_CW_DISABLE_CMD_CODE 0x02
```

Location: ble_protocol_support.h:36

Vendor specific to enable CW (RX or TX)

20.7.50 HCI_VS_RF_OUTPUT_PWR_CMD_CODE

```
#define HCI_VS_RF_OUTPUT_PWR_CMD_CODE 0x03
```

Location: ble_protocol_support.h:39

Vendor specific to set desired output power.

20.8 Bluetooth Low Energy Stack Abstraction Function Documentation

20.8.1 GAP_Initialize

```
void GAP_Initialize()
```

Location: ble_gap.h:167

Initialize GAP environment.

Initialize GAP environment after each GAPM reset.

20.8.2 GAP_GetEnv

```
const GAP_Env_t * GAP_GetEnv()
```

Location: ble_gap.h:176

Get GAP environment.

Initialize GAP environment after each GAPM reset.

Returns:

Pointer to GAP environment structure

Returns:

Pointer to GAP environment structure

20.8.3 GAP_GetProfileAddedTaskId

```
uint16_t * GAP_GetProfileAddedTaskId()
```

Location: ble_gap.h:183

Get GAP environment added profiles task identifiers.

Returns:

Pointer to an array of added profiles task identifiers

20.8.4 GAP_IsAddrPrivateResolvable

```
bool GAP_IsAddrPrivateResolvable(const uint8_t * addr, uint8_t addrType)
```

Location: ble_gap.h:192

Check address if its private resolvable.

Returns:

True if address is private resolvable false otherwise

Parameters

Direction	Name	Description
in	*addr*	Pointer to address
in	*addrType*	Type of address

20.8.5 GAP_AddAdvData

```
bool GAP_AddAdvData(bool isExtendedAdv, uint8_t newDataLen, enum gap_ad_type newDataFlag,
const uint8_t * newData, uint8_t * resultAdvData, uint8_t * resultAdvDataLen)
```

Location: ble_gap.h:207

Add GAP advertise data.

Add BLE advertisement data and scan response data to buffer isExtendedAdv If extended advertising is required newDataLen Advertisement data length to add newDataFlag Type of advertisement data newData Pointer to new advertisement data to add resultAdvData Pointer to resultant advertisement data buffer resultAdvDataLen Length of resultant advertisement data

Returns:

True if new advertisement data is added false otherwise

Returns:

True if new advertisement data is added false otherwise

Parameters

Direction	Name	Description
in	*isExtendedAdv*	If extended advertising is required
in	*newDataLen*	Advertisement data length to add
in	*newDataFlag*	Type of advertisement data
in	*newData*	Pointer to new advertisement data to add
in	*resultAdvData*	Pointer to resultant advertisement data buffer
in	*resultAdvDataLen*	Length of resultant advertisement data

20.8.6 GAPM_ResetCmd

```
void GAPM_ResetCmd(enum gapm_operation operation)
```

Location: ble_gap.h:216

Perform GAPM reset with specified GAPM operation.

Parameters

Direction	Name	Description
in	*operation*	GAPM operation code

20.8.7 GAPM_SoftwareReset

```
void GAPM_SoftwareReset()
```

Location: ble_gap.h:223

Perform GAPM software reset.

Do software reset using GAPM_RESET operation.

20.8.8 GAPM_PlatformReset

```
void GAPM_PlatformReset()
```

Location: ble_gap.h:230

Perform GAPM platform reset.

Do platform reset using GAPM_PLF_RESET operation.

20.8.9 GAPM_SetDevConfigCmd

```
void GAPM_SetDevConfigCmd(const struct gapm_set_dev_config_cmd * deviceConfig)
```

Location: ble_gap.h:244

Prepare and send set the device configuration.

Set the device configuration such as device role, manage device address type, privacy configuration, GAP/GATT service start, configure data length extension, set specific write permissions in GAP database, enable or not audio features.
NOTE: : This command is allowed only when no link is established
deviceConfig Pointer to constant GAPM set device configuration command
NOTE: : This command is allowed only when no link is established

Parameters

Direction	Name	Description
in	*deviceConfig*	Pointer to constant GAPM set device configuration command

20.8.10 GAPM_GetDeviceConfig

```
const struct gapm_set_dev_config_cmd * GAPM_GetDeviceConfig()
```

Location: ble_gap.h:256

get Device configuration

Get the device configuration such as device role, manage device address type, privacy configuration, GAP/GATT service start, configure data length extension, set specific write permissions in GAP database, enable or not audio features.
Returns:
Pointer to constant constant GAPM set device configuration command

Returns:

Pointer to constant constant GAPM set device configuration command

20.8.11 GAPM_ProfileTaskAddCmd

```
void GAPM_ProfileTaskAddCmd(uint8_t sec_lvl, uint16_t prf_task_id, uint16_t app_task,
uint16_t start_hdl, uint8_t * param, uint32_t paramSize)
```

Location: ble_gap.h:271

Prepare and Send GAPM_PROFILE_TASK_ADD_CMD to the Bluetooth stack.

Prepare and send GAPM_PROFILE_TASK_ADD_CMD to the Bluetooth stack to allocate a task for a specific profile (service or client). sec_lvl Security level prf_task_id Profile task identifier of profile to add app_task Application task number start_hdl Service start handle (0 for dynamically allocated in attribute database) param Pointer to parameter value to initialize profile paramSize Size of parameter

Parameters

Direction	Name	Description
in	*sec_lvl*	Security level
in	*prf_task_id*	Profile task identifier of profile to add
in	*app_task*	Application task number
in	*start_hdl*	Service start handle (0 for dynamically allocated in attribute database)
in	*param*	Pointer to parameter value to initialize profile
in	*paramSize*	Size of parameter

20.8.12 GAPM_GetProfileAddedCount

```
uint16_t GAPM_GetProfileAddedCount()
```

Location: ble_gap.h:281

Get number of profiles added by the Bluetooth stack.

Get the GAP environment variable that keeps track of number of added GAPM profiles.
Returns:
Number of GAPM profile added successfully

Returns:

Number of GAPM profile added successfully

20.8.13 GAPM_LepsmRegisterCmd

```
void GAPM_LepsmRegisterCmd(uint16_t le_psm, uint16_t app_task, uint8_t sec_lvl)
```

Location: ble_gap.h:293

Prepare and send GAPM_LEPSM_REGISTER_CMD to the Bluetooth stack.

Prepare and send GAPM_LEPSM_REGISTER_CMD to register an LE protocol/Service multiplexer ID in the device allowing a peer device to create LE credit based connection on it. le_psm LE protocol/service multiplexer app_task Application task number sec_lvl Security level

Parameters

Direction	Name	Description
in	*le_psm*	LE protocol/service multiplexer
in	*app_task*	Application task number
in	*sec_lvl*	Security level

20.8.14 GAPM_GenRandAddrCmd

```
void GAPM_GenRandAddrCmd(enum random_addr_type rnd_type)
```

Location: ble_gap.h:303

Prepare and send GAPM_GEN_RAND_ADDR_CMD to the Bluetooth stack.

Prepare and send GAPM_LEPSM_REGISTER_CMD to generate a random device address without starting any air operation. rnd_type Random address type

Parameters

Direction	Name	Description
in	*rnd_type*	Random address type

20.8.15 GAPM_ResolvAddrCmd

```
void GAPM_ResolvAddrCmd(uint8_t conidx, const uint8_t * peerAddr)
```

Location: ble_gap.h:314

Prepare and send GAPM_RESOLV_ADDR_CMD to the Bluetooth stack.

Prepare and send GAPM_RESOLV_ADDR_CMD to resolve a random address using array of IRK exchanged and bonded with device during pairing operation. conidx Connection identifier peerAddr Resolvable random address to solve

Parameters

Direction	Name	Description
in	*conidx*	Connection identifier
in	*peerAddr*	Resolvable random address to solve

20.8.16 GAPM_ActivityCreateAdvCmd

```
bool GAPM_ActivityCreateAdvCmd(GAPM_ActivityStatus_t * actv_status, enum gapm_own_addr
own_addr_type, const struct gapm_adv_create_param * adv_param)
```

Location: ble_gap.h:327

Prepare and send GAPM_ACTIVITY_CREATE_CMD to create an advertising activity.

Prepare and send GAPM_ACTIVITY_CREATE_CMD with operation GAPM_CREATE_ADV_ACTIVITY to create an advertising activity. actv_status Pointer to GAPM activity status own_addr_type Bluetooth device`s own address type adv_param Pointer to constant advertising parameters structure

Returns:

False if no activity slot found, true otherwise

Returns:

False if no activity slot found, true otherwise

Parameters

Direction	Name	Description
in	*actv_status*	Pointer to GAPM activity status
in	*own_addr_type*	Bluetooth device`s own address type
in	*adv_param*	Pointer to constant advertising parameters structure

20.8.17 GAPM_ActivityCreateScanCmd

```
bool GAPM_ActivityCreateScanCmd(GAPM_ActivityStatus_t * actv_status, enum gapm_own_addr own_addr_type)
```

Location: ble_gap.h:341

Prepare and send GAPM_ACTIVITY_CREATE_CMD to create scan activity.

Prepare and send GAPM_ACTIVITY_CREATE_CMD with operation GAPM_CREATE_SCAN_ACTIVITY to create scan activity. actv_status Pointer to GAPM activity status own_addr_type Bluetooth device`s own address type

Returns:

False if no activity slot found, true otherwise

Returns:

False if no activity slot found, true otherwise

Parameters

Direction	Name	Description
in	*actv_status*	Pointer to GAPM activity status
in	*own_addr_type*	Bluetooth device`s own address type

20.8.18 GAPM_ActivityCreateInitCmd

```
bool GAPM_ActivityCreateInitCmd(GAPM_ActivityStatus_t * actv_status, enum gapm_own_addr own_addr_type)
```

Location: ble_gap.h:354

Prepare and send GAPM_ACTIVITY_CREATE_CMD to create an initiating activity.

Prepare and send GAPM_ACTIVITY_CREATE_CMD with operation GAPM_CREATE_INIT_ACTIVITY to create an initiating activity. actv_status Pointer to GAPM activity status own_addr_type Bluetooth device`s own address type

Returns:

False if no activity slot found, true otherwise

Returns:

False if no activity slot found, true otherwise

Parameters

Direction	Name	Description
in	*actv_status*	Pointer to GAPM activity status
in	*own_addr_type*	Bluetooth device`s own address type

20.8.19 GAPM_ActivityCreatePeriodSyncCmd

```
bool GAPM_ActivityCreatePeriodSyncCmd(GAPM_ActivityStatus_t * actv_status, enum gapm_own_addr own_addr_type)
```

Location: ble_gap.h:367

Prepare and send GAPM_ACTIVITY_CREATE_CMD to create a periodic sync activity.

Prepare and send GAPM_ACTIVITY_CREATE_CMD with operation GAPM_CREATE_PERIOD_SYNC_ACTIVITY to create a periodic synchronization activity. actv_status Pointer to GAPM activity status own_addr_type Bluetooth device`s own address type

Returns:

False if no activity slot found, true otherwise

Returns:

False if no activity slot found, true otherwise

Parameters

Direction	Name	Description
`in`	*actv_status*	Pointer to GAPM activity status
`in`	*own_addr_type*	Bluetooth device`s own address type

20.8.20 GAPM_AdvActivityStart

```
bool GAPM_AdvActivityStart(uint8_t actv_idx, uint16_t duration, uint8_t max_adv_evt)
```

Location: ble_gap.h:383

Prepare and send GAPM_ACTIVITY_START_CMD to start an advertising activity.

Prepare and send GAPM_ACTIVITY_START_CMD to request the host to start a previously created advertising activity. actv_idx Activity identifier duration Advertising duration (in unit of 10ms). 0 means that advertising continues until the host disables it. max_adv_evt Maximum number of extended advertising events the controller shall attempt to send prior to terminating the extended advertising

Returns:

False if no activity slot found, true otherwise

Returns:

False if no activity slot found, true otherwise

Parameters

Direction	Name	Description
in	*actv_idx*	Activity identifier
in	*duration*	Advertising duration (in unit of 10ms). 0 means that advertising continues until the host disables it.
in	*max_adv_evt*	Maximum number of extended advertising events the controller shall attempt to send prior to terminating the extended advertising

20.8.21 GAPM_InitActivityStart

```
bool GAPM_InitActivityStart(uint8_t actv_idx, struct gapm_init_param * initParam)
```

Location: ble_gap.h:395

Prepare and send GAPM_ACTIVITY_START_CMD to start an initiating activity.

Prepare and send GAPM_ACTIVITY_START_CMD to request the host to start previously created initiating activity.
actv_idx Activity identifier initParam Pointer to initiating parameters structure
Returns:
False if no activity slot found, true otherwise

Returns:

False if no activity slot found, true otherwise

Parameters

Direction	Name	Description
in	*actv_idx*	Activity identifier
in	*initParam*	Pointer to initiating parameters structure

20.8.22 GAPM_ScanActivityStart

```
bool GAPM_ScanActivityStart(uint8_t actv_idx, struct gapm_scan_param * scanParam)
```

Location: ble_gap.h:407

Prepare and send GAPM_ACTIVITY_START_CMD to start a scanning activity.

Prepare and send GAPM_ACTIVITY_START_CMD to request the host to start previously created scanning activity. actv_idx Activity identifier scanParam Pointer to scan parameters structure

Returns:

False if no activity slot found, true otherwise

Returns:

False if no activity slot found, true otherwise

Parameters

Direction	Name	Description
`in`	*actv_idx*	Activity identifier
`in`	*scanParam*	Pointer to scan parameters structure

20.8.23 GAPM_PerSyncActivityStart

```
bool GAPM_PerSyncActivityStart(uint8_t actv_idx, struct gapm_per_sync_param * per_sync_
param)
```

Location: ble_gap.h:419

Prepare and send GAPM_ACTIVITY_START_CMD to start a periodic sync activity.

Prepare and send GAPM_ACTIVITY_START_CMD to request the host to start previously created periodic synchronization activity. actv_idx Activity identifier per_sync_param Pointer to periodic sync parameters structure

Returns:

False if no activity slot found, true otherwise

Returns:

False if no activity slot found, true otherwise

Parameters

Direction	Name	Description
in	*actv_idx*	Activity identifier
in	*per_sync_param*	Pointer to periodic sync parameters structure

20.8.24 GAPM_ActivityStartCmd

```
bool GAPM_ActivityStartCmd(uint8_t actv_idx, union gapm_u_param * u_param)
```

Location: ble_gap.h:431

Prepare and send GAPM_ACTIVITY_START_CMD to start an activity.

Prepare and send GAPM_ACTIVITY_START_CMD to request the host to start previously created activity. actv_idx Activity identifier u_param Pointer to union to containing activity parameters

Returns:

False if no activity slot found, true otherwise

Returns:

False if no activity slot found, true otherwise

Parameters

Direction	Name	Description
in	*actv_idx*	Activity identifier
in	*u_param*	Pointer to union to containing activity parameters

20.8.25 GAPM_ActivityStop

```
bool GAPM_ActivityStop(uint8_t actv_idx)
```

Location: ble_gap.h:442

Prepare and send GAPM_ACTIVITY_STOP_CMD to stop a specified activity.

Prepare and send GAPM_ACTIVITY_STOP_CMD with operation GAPM_STOP_ACTIVITY request the host to stop a specified activity. actv_idx Activity identifier

Returns:

False if activity does not exist, true otherwise

Returns:

False if activity does not exist, true otherwise

Parameters

Direction	Name	Description
`in`	*actv_idx*	Activity identifier

20.8.26 GAPM_ActivityStopAll

```
bool GAPM_ActivityStopAll()
```

Location: ble_gap.h:452

Prepare and send GAPM_ACTIVITY_STOP_CMD to stop all existing activities.

Prepare and send GAPM_ACTIVITY_STOP_CMD with operation GAPM_STOP_ALL_ACTIVITIES request the host to stop all existing activities.

Returns:

False if activity does not exist, true otherwise

Returns:

False if activity does not exist, true otherwise

20.8.27 GAPM_DeleteActivity

```
bool GAPM_DeleteActivity(uint8_t actv_idx)
```

Location: ble_gap.h:463

Prepare and send GAPM_ACTIVITY_DELETE_CMD to delete a specified activity.

Prepare and send GAPM_ACTIVITY_DELETE_CMD with operation GAPM_DELETE_ACTIVITY to request the host to delete an existing specified activity. actv_idx Activity identifier (used only if operation is GAPM_DELETE_ACTIVITY)

Returns:

False if activity does not exist, true otherwise

Returns:

False if activity does not exist, true otherwise

Parameters

Direction	Name	Description
in	*actv_idx*	Activity identifier (used only if operation is GAPM_DELETE_ACTIVITY)

20.8.28 GAPM_DeleteAllActivities

```
bool GAPM_DeleteAllActivities()
```

Location: ble_gap.h:473

Prepare and send GAPM_ACTIVITY_DELETE_CMD to delete all the activities.

Prepare and send GAPM_ACTIVITY_DELETE_CMD with operation GAPM_DELETE_ALL_ACTIVITIES to request the host to delete all existing activities.

Returns:

False if activity does not exist, true otherwise

Returns:

False if activity does not exist, true otherwise

20.8.29 GAPM_ActivityStopCmd

```
bool GAPM_ActivityStopCmd(uint8_t operation, uint8_t actv_idx)
```

Location: ble_gap.h:485

Prepare and send GAPM_ACTIVITY_STOP_CMD to stop a specified activity.

Prepare and send GAPM_ACTIVITY_STOP_CMD with operation GAPM_STOP_ACTIVITY request the host to stop a specified activity. operation GAPM operation code [GAPM_STOP_ACTIVITY or GAPM_STOP_ALL_ACTIVITIES] actv_idx Activity identifier (used only if operation is GAPM_STOP_ACTIVITY)

Returns:

False if activity does not exist, true otherwise

Returns:

False if activity does not exist, true otherwise

Parameters

Direction	Name	Description
`in`	*operation*	GAPM operation code [GAPM_STOP_ACTIVITY or GAPM_STOP_ALL_ACTIVITIES]
`in`	*actv_idx*	Activity identifier (used only if operation is GAPM_STOP_ACTIVITY)

20.8.30 GAPM_DeleteActivityCmd

```
bool GAPM_DeleteActivityCmd(uint8_t operation, uint8_t actv_idx)
```

Location: ble_gap.h:497

Prepare and send GAPM_ACTIVITY_DELETE_CMD to delete activity/activities.

Prepare and send GAPM_ACTIVITY_DELETE_CMD to request the host to delete either an activity or all currently existing activities. operation GAPM operation code [GAPM_DELETE_ACTIVITY or GAPM_DELETE_ALL_ACTIVITIES] actv_idx Activity identifier (used only if operation is GAPM_DELETE_ACTIVITY)

Returns:

False if activity does not exist, true otherwise

Returns:

False if activity does not exist, true otherwise

Parameters

Direction	Name	Description
in	*operation*	GAPM operation code [GAPM_DELETE_ACTIVITY or GAPM_DELETE_ALL_ACTIVITIES]
in	*actv_idx*	Activity identifier (used only if operation is GAPM_DELETE_ACTIVITY)

20.8.31 GAPM_SetAdvDataCmd

```
bool GAPM_SetAdvDataCmd(uint8_t operation, uint8_t actv_idx, uint8_t length, uint8_t *
data)
```

Location: ble_gap.h:514

Prepare and send GAPM_SET_ADV_DATA_CMD to set an advertising data or scan response data or periodic advertising data.

Prepare and send GAPM_SET_ADV_DATA_CMD to either set an advertising data or the scan response data or the periodic advertising data for a given advertising activity identified by activity ID. operation GAPM operation code (GAPM_SET_ADV_DATA or GAPM_SET_SCAN_RSP_DATA or GAPM_SET_PERIOD_ADV_DATA) actv_idx Activity identifier length Length of data data Pointer to data

Returns:

False if activity slot not found, true otherwise

Returns:

False if activity slot not found, true otherwise

Parameters

Direction	Name	Description
in	*operation*	GAPM operation code (GAPM_SET_ADV_DATA or GAPM_SET_SCAN_RSP_DATA or GAPM_SET_PERIOD_ADV_DATA)
in	*actv_idx*	Activity identifier
in	*length*	Length of data
in	*data*	Pointer to data

20.8.32 GAPM_MsgHandler

```
void GAPM_MsgHandler(ke_msg_id_t const msg_id, void const * param, ke_task_id_t const dest_id, ke_task_id_t const src_id)
```

Location: ble_gap.h:527

Message handler for GAPM events.

This function receives GAPM events from he Bluetooth stack and perform actions accordingly. It initialize GAP and GATT at reset and also manages GAPM states and the GAP environment. msg_id Kernel message identifier param Pointer to constant parameter dest_id Constant destination kernel identifier src_id Constant source kernel identifier

Parameters

Direction	Name	Description
in	*msg_id*	Kernel message identifier
in	*param*	Pointer to constant parameter
in	*dest_id*	Constant destination kernel identifier
in	*src_id*	Constant source kernel identifier

20.8.33 GAPC_ParamUpdateCmd

```
void GAPC_ParamUpdateCmd(uint8_t conidx, uint16_t intv_min, uint16_t intv_max, uint16_t latency, uint16_t time_out, uint16_t ce_len_min, uint16_t ce_len_max)
```

Location: ble_gap.h:543

Prepare and send GAPC_PARAM_UPDATE_CMD to perform an update on parameters.

Prepare and send GAPC_PARAM_UPDATE_CMD to perform an update of connection parameters. conidx Connection identifier intv_min Minimum of connection interval (value = intv_max * 1.25 ms) intv_max Maximum of connection interval (value = intv_max * 1.25 ms) time_out Link supervision timeout (value = time_out * 10 ms) latency connection latency ce_len_min Minimum CE length (value = ce_len_min * 0.625 ms) ce_len_max Maximum CE length (value = ce_len_min * 0.625 ms)

Parameters

Direction	Name	Description
in	*conidx*	Connection identifier
in	*intv_min*	Minimum of connection interval (value = intv_max * 1.25 ms)
in	*intv_max*	Maximum of connection interval (value = intv_max * 1.25 ms)
in	*latency*	connection latency
in	*time_out*	Link supervision timeout (value = time_out * 10 ms)
in	*ce_len_min*	Minimum CE length (value = ce_len_min * 0.625 ms)
in	*ce_len_max*	Maximum CE length (value = ce_len_min * 0.625 ms)

20.8.34 GAPC_ParamUpdateCfm

```
void GAPC_ParamUpdateCfm(uint8_t conidx, bool accept, uint16_t ce_len_min, uint16_t ce_
len_max)
```

Location: ble_gap.h:557

Prepare and send GAPC_PARAM_UPDATE_CFM to accept or reject connection parameters.

Prepare and send GAPC_PARAM_UPDATE_CFM to accept or reject connection parameters proposed by the peer device. conidx Connection identifier accept Accept(0x01) or reject(0x00) slave connection parameters ce_len_min Minimum CE length (value = ce_len_min * 0.625 ms) ce_len_max Maximum CE length (value = ce_len_min * 0.625 ms)

Parameters

Direction	Name	Description
in	*conidx*	Connection identifier
in	*accept*	Accept(0x01) or reject(0x00) slave connection parameters
in	*ce_len_min*	Minimum CE length (value = ce_len_min * 0.625 ms)
in	*ce_len_max*	Maximum CE length (value = ce_len_min * 0.625 ms)

20.8.35 GAPC_ConnectionCfm

```
void GAPC_ConnectionCfm(uint8_t conidx, struct gapc_connection_cfm * param)
```

Location: ble_gap.h:569

Prepare and send GAPC_CONNECTION_CFM in a response to connection request.

Prepare and send GAPC_CONNECTION_CFM in a response to connection request from the peer to enable local attribute tasks and security manager for the connection. conidx Connection identifier param Pointer to GAPC connection confirmation structure

Parameters

Direction	Name	Description
`in`	*conidx*	Connection identifier
`in`	*param*	Pointer to GAPC connection confirmation structure

20.8.36 GAPC_DisconnectCmd

```
void GAPC_DisconnectCmd(uint8_t conidx, uint8_t reason)
```

Location: ble_gap.h:580

Prepare and send GAPC_DISCONNECT_CMD to disconnect link.

Prepare and send GAPC_DISCONNECT_CMD to request a disconnection of the link. conidx Connection identifier reason Reason for disconnection

Parameters

Direction	Name	Description
`in`	*conidx*	Connection identifier
`in`	*reason*	Reason for disconnection

20.8.37 GAPC_IsConnectionActive

```
bool GAPC_IsConnectionActive(uint8_t conidx)
```

Location: ble_gap.h:591

Check if the specified connection is active.

Check if specified connection ID is active or not. conidx Connection identifier
Returns:
True if given connection id have valid GAP connection handle, false otherwise

Returns:

True if given connection id have valid GAP connection handle, false otherwise

Parameters

Direction	Name	Description
`in`	*conidx*	Connection identifier

20.8.38 GAPC_DisconnectAll

```
void GAPC_DisconnectAll(uint8_t reason)
```

Location: ble_gap.h:601

Prepare and send GAPC_DISCONNECT_CMD to disconnect all the links.

Prepare and send GAPC_DISCONNECT_CMD to request disconnection for each active connections. reason Reason for disconnection

Parameters

Direction	Name	Description
`in`	*reason*	Reason for disconnection

20.8.39 GAPC_ConnectionCount

```
uint8_t GAPC_ConnectionCount()
```

Location: ble_gap.h:608

Get the number of active connection count.

Returns:

Number of active connection count

20.8.40 GAPC_MasterConnectionCount

```
uint8_t GAPC_MasterConnectionCount()
```

Location: ble_gap.h:615

Get the master connection count.

Returns:

Number of master connection count

20.8.41 GAPC_SlaveConnectionCount

```
uint8_t GAPC_SlaveConnectionCount()
```

Location: ble_gap.h:622

Get the slave connection count.

Returns:

Number of slave connection count

20.8.42 GAPC_GetConnectionInfo

```
const struct gapc_connection_req_ind * GAPC_GetConnectionInfo(uint8_t conidx)
```

Location: ble_gap.h:632

Get the connection information of a specified connection ID.

Get GAPC connection information of a specified connection ID. conidx Connection identifier
Returns:
Pointer to constant structure gapc_connection_req_ind

Returns:

Pointer to constant structure gapc_connection_req_ind

Parameters

Direction	Name	Description
`in`	*conidx*	Connection identifier

20.8.43 GAPC_GetDevInfoCfm_Name

```
void GAPC_GetDevInfoCfm_Name(uint8_t conidx, const char * devName)
```

Location: ble_gap.h:643

Prepare and send GAPC_GET_DEV_INFO_CFM to send the device name.

Prepare and send GAPC_GET_DEV_INFO_CFM to send the device name information to the peer device. conidx Connection identifier devName Pointer to device name

Parameters

Direction	Name	Description
`in`	*conidx*	Connection identifier
`in`	*devName*	Pointer to device name

20.8.44 GAPC_GetDevInfoCfm_Appearance

```
void GAPC_GetDevInfoCfm_Appearance(uint8_t conidx, uint16_t appearance)
```

Location: ble_gap.h:654

Prepare and send GAPC_GET_DEV_INFO_CFM to send the device appearance.

Prepare and send GAPC_GET_DEV_INFO_CFM to send the device appearance icon to the peer device. conidx Connection identifier appearance Appearance icon

Parameters

Direction	Name	Description
in	*conidx*	Connection identifier
in	*appearance*	Appearance icon

20.8.45 GAPC_GetDevInfoCfm_SlvPrefParams

```
void GAPC_GetDevInfoCfm_SlvPrefParams(uint8_t conidx, uint16_t con_intv_min, uint16_t
con_intv_max, uint16_t slave_latency, uint16_t conn_timeout)
```

Location: ble_gap.h:668

Prepare and send GAPC_GET_DEV_INFO_CFM to send device slave preferred parameters.

Prepare and send GAPC_GET_DEV_INFO_CFM to to send device slave preferred parameters to peer device. conidx Connection identifier con_intv_min Minimum connection interval (value = con_intv_min * 1.25 ms) con_intv_max Maximum connection interval (value = con_intv_max * 1.25 ms) slave_latency Slave latency conn_timeout Connection supervision timeout (value = conn_timeout * 10 ms)

Parameters

Direction	Name	Description
in	*conidx*	Connection identifier
in	*con_intv_min*	Minimum connection interval (value = con_intv_min * 1.25 ms)
in	*con_intv_max*	Maximum connection interval (value = con_intv_max * 1.25 ms)
in	*slave_latency*	Slave latency
in	*conn_timeout*	Connection supervision timeout (value = conn_timeout * 10 ms)

20.8.46 GAPC_GetDevInfoCfm

```
void GAPC_GetDevInfoCfm(uint8_t conidx, uint8_t req, const union gapc_dev_info_val * dat)
```

Location: ble_gap.h:682

Prepare and send GAPC_GET_DEV_INFO_CFM to send requested information.

Prepare and send GAPC_GET_DEV_INFO_CFM to send requested information to the peer device. conidx Connection identifier req Requested information dat Pointer to constant union GAPC device information value

Parameters

Direction	Name	Description
in	*conidx*	Connection identifier
in	*req*	Requested information
in	*dat*	Pointer to constant union GAPC device information value

20.8.47 GAPC_SetDevInfoCfm

```
bool GAPC_SetDevInfoCfm(uint8_t conidx, uint8_t req, bool accept)
```

Location: ble_gap.h:697

Prepare and send GAPC_SET_DEV_INFO_CFM to confirm if the requested device information was written.

Prepare and send GAPC_SET_DEV_INFO_CFM to provide confirmation to the Bluetooth stack if the requested device information was written. conidx Connection identifier req Requested information (GAPC_DEV_NAME or GAPC_DEV_APPEARANCE) accept True if the write request has been approved, false otherwise

Returns:

False if req is not GAPC_DEV_NAME or GAPC_DEV_APPEARANCE, true otherwise

Returns:

False if req is not GAPC_DEV_NAME or GAPC_DEV_APPEARANCE, true otherwise

Parameters

Direction	Name	Description
`in`	*conidx*	Connection identifier
`in`	*req*	Requested information (GAPC_DEV_NAME or GAPC_DEV_APPEARANCE)
`in`	*accept*	True if the write request has been approved, false otherwise

20.8.48 GAPC_BondCfm

```
void GAPC_BondCfm(uint8_t conidx, enum gapc_bond request, bool accept, const union gapc_bond_cfm_data * data)
```

Location: ble_gap.h:714

GAPC bond and encryption operations.

Prepare and send GAPC_BOND_CFM to send confirmation to a bond request Prepare and send GAPC_BOND_CFM to provide confirmation of the receipt of a GAPC_BOND_REQ_IND message. conidx Connection identifier request Bond request type accept Accept(0x01) or reject(0x00) request data Pointer to constant GAPC bond confirmation data union

Parameters

Direction	Name	Description
in	*conidx*	Connection identifier
in	*request*	Bond request type
in	*accept*	Accept(0x01) or reject(0x00) request
in	*data*	Pointer to constant GAPC bond confirmation data union

20.8.49 GAPC_EncryptCmd

```
void GAPC_EncryptCmd(uint8_t conidx, uint16_t ediv, const uint8_t * randnb, const uint8_t
* ltk, uint8_t key_size)
```

Location: ble_gap.h:729

Prepare and send GAPC_ENCRYPT_CMD to request encrypting the connection link.

Prepare and send GAPC_ENCRYPT_CMD as the master to request initiation of the link encryption procedure. conidx Connection identifier ediv Encryption diversifier randnb Pointer to constant random number ltk Pointer to constant long term key key_size Size of LTK key

Parameters

Direction	Name	Description
in	*conidx*	Connection identifier
in	*ediv*	Encryption diversifier
in	*randnb*	Pointer to constant random number
in	*ltk*	Pointer to constant long term key
in	*key_size*	Size of LTK key

20.8.50 GAPC_EncryptCfm

```
void GAPC_EncryptCfm(uint8_t conidx, bool found, const uint8_t * ltk, uint8_t key_size)
```

Location: ble_gap.h:743

Prepare and send GAPC_ENCRYPT_CFM to confirm an encryption request.

Prepare and send GAPC_ENCRYPT_CFM to provide confirmation of the receipt of a GAPC_ENCRYPY_REQ_IND message. conidx Connection identifier found Nonzero if LTK is found for peer device, zero otherwise ltk Pointer to constant long term key key_size Size of LTK key

Parameters

Direction	Name	Description
in	*conidx*	Connection identifier
in	*found*	Nonzero if LTK is found for peer device, zero otherwise
in	*ltk*	Pointer to constant long term key
in	*key_size*	Size of LTK key

20.8.51 GAPC_IsBonded

```
bool GAPC_IsBonded(uint8_t conidx)
```

Location: ble_gap.h:753

Check if the connection is bonded.

Check if the specified connection ID is active and if it has a valid bond state. conidx Connection identifier

Returns:

True if connection is active and it has valid bond state, false otherwise

Returns:

True if connection is active and it has valid bond state, false otherwise

Parameters

Direction	Name	Description
in	*conidx*	Connection identifier

20.8.52 GAPC_GetBondInfo

```
const BondInfo_t * GAPC_GetBondInfo(uint8_t conidx)
```

Location: ble_gap.h:764

Get bond information for the connection.

Get bond information for a specified connection ID. conidx Connection identifier
Returns:
Pointer to constant bond information structure if connection is active and its bonded, NULL otherwise

Returns:

Pointer to constant bond information structure if connection is active and its bonded, NULL otherwise

Parameters

Direction	Name	Description
in	*conidx*	Connection identifier

20.8.53 GAPC_BondCmd

```
void GAPC_BondCmd(uint8_t conidx, const struct gapc_pairing * pairing)
```

Location: ble_gap.h:775

Prepare and send GAPC_BOND_CMD to initiate a bond procedure.

Prepare and send GAPC_BOND_CMD to the master of the link in order to initiate the bond procedure. conidx Connection identifier pairing Pointer to constant structure pairing information

Parameters

Direction	Name	Description
in	*conidx*	Connection identifier
in	*pairing*	Pointer to constant structure pairing information

20.8.54 GAPC_AddRecordToBondList

```
uint16_t GAPC_AddRecordToBondList(uint8_t conidx)
```

Location: ble_gap.h:785

Add record to bond list.

Add device with a specified connection ID to bond list. conidx Connection identifier
Returns:
Non-zero if successful, zero otherwise

Returns:

Non-zero if successful, zero otherwise

Parameters

Direction	Name	Description
in	*conidx*	Connection identifier

20.8.55 GAPC_MsgHandler

```
void GAPC_MsgHandler(ke_msg_id_t const msg_id, void const * param, ke_task_id_t const
dest_id, ke_task_id_t const src_id)
```

Location: ble_gap.h:797

GAPC message handler.

Receives GAPC events from the Bluetooth stack and perform actions accordingly. msg_id Kernel message identifier param Pointer to constant parameter dest_id Constant destination kernel identifier src_id Constant source kernel identifier

Parameters

Direction	Name	Description
in	*msg_id*	Kernel message identifier
in	*param*	Pointer to constant parameter
in	*dest_id*	Constant destination kernel identifier
in	*src_id*	Constant source kernel identifier

20.8.56 GAPM_ListSetWlCmd

```
void GAPM_ListSetWlCmd(uint8_t operation, uint8_t nb, struct gap_bdaddr * devices_wl)
```

Location: ble_gap.h:810

Prepare and send GAPM_LIST_SET_CMD to set the white list content.

Prepare and send GAPM_LIST_SET_CMD to set the content of the white list. operation GAPM operation code to set the content of white list nb Number of entries to be added in the list devices_wl Pointer to structure containing list of addresses to be added in the white list

Parameters

Direction	Name	Description
in	*operation*	GAPM operation code to set the content of white list
in	*nb*	Number of entries to be added in the list
in	*devices_wl*	Pointer to structure containing list of addresses to be added in the white list

20.8.57 GAPM_ListSetRalCmd

```
void GAPM_ListSetRalCmd(uint8_t operation, uint8_t nb, struct gap_ral_dev_info * rl_
devinfo)
```

Location: ble_gap.h:823

Prepare and send GAPM_LIST_SET_CMD to set the resolving list content.

Prepare and send GAPM_LIST_SET_CMD to set the content of the resolving list. operation GAPM operation code to set the content of white list nb Number of entries to be added in the list rl_devinfo Pointer to structure containing list of addresses to be added in the resolving list

Parameters

Direction	Name	Description
in	*operation*	GAPM operation code to set the content of white list
in	*nb*	Number of entries to be added in the list
in	*rl_devinfo*	Pointer to structure containing list of addresses to be added in the resolving list

20.8.58 GAPM_IsIRKValid

```
bool GAPM_IsIRKValid(const BondInfo_t * bond_info)
```

Location: ble_gap.h:834

Check if IRK is valid.

Function checks if a valid IRK has been previously exchanged or not. bond_info Pointer to constant bond information

Returns:

True if irk_exchanged in the specified bond_info is nonzero, false otherwise

Returns:

True if irk_exchanged in the specified bond_info is nonzero, false otherwise

Parameters

Direction	Name	Description
in	*bond_info*	Pointer to constant bond information

20.8.59 WhiteList_ResolvableList_Update

```
bool WhiteList_ResolvableList_Update()
```

Location: ble_gap.h:842

Update the white list and the resolve List.

Copies the addresses of the devices in the bond list and updates them in the Bluetooth stack's white list and the resolvable list (if applicable).

20.8.60 GAPM_GetWhitelistNumDev

```
uint16_t GAPM_GetWhitelistNumDev()
```

Location: ble_gap.h:851

Get number of devices in whitelist.

Get the variable that keeps track of number of devices added to whitelist.

Returns:

Number of devices in whitelist

Returns:

Number of devices in whitelist

20.8.61 GAPM_SetWhitelistNumDev

```
void GAPM_SetWhitelistNumDev()
```

Location: ble_gap.h:860

Set the number of devices in whitelist.

Set the GAPM variable that keeps track of number of devices added to whitelist to value.

Returns:

None

Returns:

None

20.8.62 GAPC_SetPhyCmd

```
void GAPC_SetPhyCmd(uint8_t conidx, uint8_t rx_rate, uint8_t tx_rate, uint8_t
preferredRate)
```

Location: ble_gap.h:877

GAPC PHY management operations.

Prepare and send GAPC_SET_PHY_CMD to set the preferred PHY Prepare and send GAPC_SET_PHY_CMD to set the preferred PHY for current active link. conidx Connection identifier rx_rate Preferred LE PHYs for reception tx_rate Preferred LE PHYs for transmission preferredRate Preferred coding scheme for LE coded

Parameters

Direction	Name	Description
in	*conidx*	Connection identifier
in	*rx_rate*	Preferred LE PHYs for reception
in	*tx_rate*	Preferred LE PHYs for transmission
in	*preferredRate*	Preferred coding scheme for LE coded

20.8.63 GAPC_CteTxCfgCmd

```
void GAPC_CteTxCfgCmd(uint8_t conidx, uint8_t cte_type, uint8_t ant_pattern_len, uint8_t
* ant_id)
```

Location: ble_gap.h:895

GAPC constant Tone extension operations.

Prepare and send GAPC_CTE_TX_CFG_CMD to configure CTE TX Prepare and send GAPC_CTE_TX_CFG_CMD to configure constant tone extension transmission. conidx Connection identifier cte_type CTE types ant_pattern_len Length of switching pattern ant_id Pointer to antenna IDs

Parameters

Direction	Name	Description
in	*conidx*	Connection identifier
in	*cte_type*	CTE types
in	*ant_pattern_len*	Length of switching pattern
in	*ant_id*	Pointer to antenna IDs

20.8.64 GAPC_CteRxCfgCmd

```
void GAPC_CteRxCfgCmd(uint8_t conidx, uint8_t slot_dur, uint8_t ant_pattern_len, uint8_t
* ant_id, bool sample_en)
```

Location: ble_gap.h:910

Prepare and send GAPC_CTE_RX_CFG_CMD to configure CTE RX.

Prepare and send GAPC_CTE_RX_CFG_CMD to configure constant tone extension reception. conidx Connection identifier slot_dur Slot durations (1: 1us, 2: 2us) ant_pattern_len Length of switching pattern ant_id Pointer to antenna IDs sample_en Enable or disable sampling

Parameters

Direction	Name	Description
in	*conidx*	Connection identifier
in	*slot_dur*	Slot durations (1: 1us, 2: 2us)
in	*ant_pattern_len*	Length of switching pattern
in	*ant_id*	Pointer to antenna IDs
in	*sample_en*	Enable or disable sampling

20.8.65 GAPC_CteReqCtrlCmd

```
void GAPC_CteReqCtrlCmd(uint8_t conidx, uint8_t cte_type, uint8_t cte_len, uint8_t cte_
interval, bool enable)
```

Location: ble_gap.h:925

Prepare and send GAPC_CTE_REQ_CTRL_CMD to control CTE requests.

Prepare and send GAPC_CTE_REQ_CTRL_CMD to control constant tone extension requests. conidx Connection identifier cte_type CTE types cte_len Request CTE length (in 8us unit) cte_interval CTE request interval (in no of connection events) enable Enable or disable TX/RX CTE

Parameters

Direction	Name	Description
in	*conidx*	Connection identifier
in	*cte_type*	CTE types
in	*cte_len*	Request CTE length (in 8us unit)
in	*cte_interval*	CTE request interval (in no of connection events)
in	*enable*	Enable or disable TX/RX CTE

20.8.66 GAPC_CteRspCtrlCmd

```
void GAPC_CteRspCtrlCmd(uint8_t conidx, bool enable)
```

Location: ble_gap.h:937

Prepare and send GAPC_CTE_RSP_CTRL_CMD to control CTE reception.

Prepare and send GAPC_CTE_RSP_CTRL_CMD to control constant tone extension reception. conidx Connection identifier enable Enable or disable TX/RX CTE

Parameters

Direction	Name	Description
in	*conidx*	Connection identifier
in	*enable*	Enable or disable TX/RX CTE

20.8.67 GAPC_GetInfoCmd

```
void GAPC_GetInfoCmd(uint8_t conidx, uint8_t operation)
```

Location: ble_gap.h:952

GAPC local and peer device information operations.

Prepare and send GAPC_GET_INFO_CMD to get the peer information Prepare and send GAPC_GET_INFO_CMD to get information about the peer device or about the current active link. conidx Connection identifier operation GAPC requested operation

Parameters

Direction	Name	Description
`in`	*conidx*	Connection identifier
`in`	*operation*	GAPC requested operation

20.8.68 GAPM_PerAdvCteTxCmd

```
bool GAPM_PerAdvCteTxCmd(uint8_t actv_idx, bool enable)
```

Location: ble_gap.h:962

Prepare and send GAPM_PER_ADV_CTE_TX_CTL_CMD to control CTE transmission in a periodic advertising activity.

Returns:

False if activity slot is equal to max number of activity, true otherwise

Parameters

Direction	Name	Description
`in`	*actv_idx*	Activity identifier
`in`	*enable*	Enable or disable CTE transmission

20.8.69 GAPM_PerAdvReportCtrlCmd

```
bool GAPM_PerAdvReportCtrlCmd(uint8_t actv_idx, bool enable)
```

Location: ble_gap.h:972

Prepare and send GAPM_PER_ADV_REPORT_CTRL_CMD to control reception of periodic advertising report in a periodic advertising sync activity.

Returns:

False if activity slot is equal to max number of activity, true otherwise

Parameters

Direction	Name	Description
`in`	*actv_idx*	Activity identifier
`in`	*enable*	Enable or disable reception of periodic advertising report

20.8.70 GAPM_PerSyncIQSamplingCtrlCmd

```
bool GAPM_PerSyncIQSamplingCtrlCmd(uint8_t actv_idx, uint8_t slot_dur, uint8_t max_
sample_cte, uint8_t ant_pattern_len, uint8_t * ant_id, bool enable)
```

Location: ble_gap.h:987

Prepare and send GAPM_PER_SYNC_IQ_SAMPLING_CTRL_CMD to control capturing IQ samples from the constant tone extension of periodic advertising packets.

Returns:

False if activity slot is equal to max number of activity, true otherwise

Parameters

Direction	Name	Description
in	*actv_idx*	Activity identifier
in	*slot_dur*	Slot durations (1: 1us | 2: 2us)
in	*max_sample_cte*	Maximum sampled CTEs
in	*ant_pattern_len*	Length of switching pattern
in	*ant_id*	Pointer to antenna IDs
in	*enable*	Enable IQ sampling

20.8.71 GATT_Initialize

```
void GATT_Initialize()
```

Location: ble_gatt.h:198

GATT initialization.

Initialize the GATT environment.

20.8.72 GATT_GetEnv

```
const GATT_Env_t * GATT_GetEnv()
```

Location: ble_gatt.h:207

Get GATT environment.

Function gets a reference to the internal GATT environment structure.

Returns:

A constant pointer to GATT_Env_t

Returns:

A constant pointer to GATT_Env_t

20.8.73 GATT_SetEnvData

```
void GATT_SetEnvData(uint16_t * disc_svc_count, cust_svc_desc * custom_service_db, uint8_t max_cust_svc_num)
```

Location: ble_gatt.h:219

Set GATT environment data.

Set the GATT environment discovery service counter (array), custom service database and maximum number of custom services. disc_svc_count Pointer to discovered services count custom_service_db Pointer to custom service database max_cust_svc_num Maximum number of custom services

Parameters

Direction	Name	Description
in	*disc_svc_count*	Pointer to discovered services count
in	*custom_service_db*	Pointer to custom service database
in	*max_cust_svc_num*	Maximum number of custom services

20.8.74 GATT_GetMaxCustomServiceNumber

```
uint16_t GATT_GetMaxCustomServiceNumber()
```

Location: ble_gatt.h:229

Get the maximum number of custom services in GATT.

Function gets the maximum number of custom services in GATT environment.

Returns:

max_cust_svc_num maximum number of custom services

Returns:

max_cust_svc_num maximum number of custom services

20.8.75 GATTM_GetServiceAddedCount

```
uint16_t GATTM_GetServiceAddedCount()
```

Location: ble_gatt.h:238

Get GATT added services count.

Function gets number of GATT services that have been added.
Returns:
the number of GATT services that have been added

Returns:

the number of GATT services that have been added

20.8.76 GATTM_ResetServiceAttributeDatabaseID

```
void GATTM_ResetServiceAttributeDatabaseID()
```

Location: ble_gatt.h:247

Resets GATT services attribute database ID.

Function resets GATT services attribute database ID. This should be called before re-adding the attribute databases via GATTM_AddAttributeDatabase() without complete system reset. GATTM_AddAttributeDatabase() without complete system reset.

20.8.77 GATTM_AddAttributeDatabase

```
void GATTM_AddAttributeDatabase(const struct att_db_desc * att_db, uint16_t att_db_len)
```

Location: ble_gatt.h:259

Add service and attributes to the Bluetooth stack.

Prepare and send GATTM_ADD_SVC_REQ to add services/characteristics into the Bluetooth stack database. att_db Pointer to constant custom service attribute database att_db_len Number of elements in the att_db array
NOTE: Triggers a GATTM_ADD_SVC_RSP for every service added.

NOTE: Triggers a GATTM_ADD_SVC_RSP for every service added.

Parameters

Direction	Name	Description
in	*att_db*	Pointer to constant custom service attribute database
in	*att_db_len*	Number of elements in the att_db array

20.8.78 GATTM_GetHandle

```
uint16_t GATTM_GetHandle(uint8_t cs_svc_number, uint16_t attidx)
```

Location: ble_gatt.h:274

Get the handle for an attribute.

Returns the stack database handle for a certain attribute index. cs_svc_number Custom service number related to attribute attidx Attribute index

Returns:

Handle value, in case of success or 0, in case the attidx or the start handle is invalid

NOTE: The user has used GATT_AddAttributeDatabase() to construct the database and the stack has already finished adding services to the database.

Returns:

Handle value, in case of success or 0, in case the attidx or the start handle is invalid

NOTE: The user has used GATT_AddAttributeDatabase() to construct the database and the stack has already finished adding services to the database.

Parameters

Direction	Name	Description
in	*cs_svc_number*	Custom service number related to attribute
in	*attidx*	Attribute index

20.8.79 GATTM_MsgHandler

```
void GATTM_MsgHandler(ke_msg_id_t const msg_id, void const * param, ke_task_id_t const dest_id, ke_task_id_t const src_id)
```

Location: ble_gatt.h:286

Handle GATTM messages.

Message handler for the GATT manager msg_id Kernel message identifier param Pointer to constant message parameter (unused) dest_id Destination task identifier src_id Source task identifier

Parameters

Direction	Name	Description
in	*msg_id*	Kernel message identifier
in	*param*	Pointer to constant message parameter (unused)
in	*dest_id*	Destination task identifier
in	*src_id*	Source task identifier

20.8.80 GATTC_Initialize

```
void GATTC_Initialize()
```

Location: ble_gatt.h:294

GATTC initialization.

Initialize GATTC

20.8.81 GATTC_DiscByUUIDSvc

```
void GATTC_DiscByUUIDSvc(uint8_t conidx, uint8_t uuid, uint8_t uuid_len, uint16_t start_hdl, uint16_t end_hdl)
```

Location: ble_gatt.h:308

GATTC services discovery by UUID.

Start a discovery by UUID(GATTC_DISC_BY_UUID_SVC) in a specified handle range. conidx Connection index uuid[] Service UUID uuid_len UUID length (ATT_UUID_16_LEN or ATT_UUID_32_LEN or ATT_UUID_128_LEN) start_hdl Start handle of a discovery range end_hdl End handle of a discovery range

Parameters

Direction	Name	Description
`in`	*conidx*	Connection index
	uuid	
`in`	*uuid_len*	UUID length (ATT_UUID_16_LEN or ATT_UUID_32_LEN or ATT_UUID_128_LEN)
`in`	*start_hdl*	Start handle of a discovery range
`in`	*end_hdl*	End handle of a discovery range

20.8.82 GATTC_DiscAllSvc

```
void GATTC_DiscAllSvc(uint8_t conidx, uint16_t start_hdl, uint16_t end_hdl)
```

Location: ble_gatt.h:320

GATTC all services discovery.

Start a discovery for all services in a specified handle range. conidx Connection index start_hdl Start handle of discovery range end_hdl End handle of discovery range

Parameters

Direction	Name	Description
`in`	*conidx*	Connection index
`in`	*start_hdl*	Start handle of discovery range
`in`	*end_hdl*	End handle of discovery range

20.8.83 GATTC_DiscAllChar

```
void GATTC_DiscAllChar(uint8_t conidx, uint16_t start_hdl, uint16_t end_hdl)
```

Location: ble_gatt.h:332

GATTC all characteristics discovery.

Start a characteristic discovery for all characteristics in a specified handle range. conidx Connection index start_hdl Start handle of discovery range end_hdl End handle of discovery range

Parameters

Direction	Name	Description
in	*conidx*	Connection index
in	*start_hdl*	Start handle of discovery range
in	*end_hdl*	End handle of discovery range

20.8.84 GATTC_SendEvtCmd

```
void GATTC_SendEvtCmd(uint8_t conidx, uint8_t operation, uint16_t seq_num, uint16_t
handle, uint16_t length, uint8_t * value)
```

Location: ble_gatt.h:346

GATTC send event.

Send GATTC characteristic notification or indication event. conidx Connection index operation GATTC_NOTIFY or GATTC_INDICATE seq_num Operation sequence number handle Characteristic handle length Attribute length value Pointer to attribute data value

Parameters

Direction	Name	Description
in	*conidx*	Connection index
in	*operation*	GATTC_NOTIFY or GATTC_INDICATE
in	*seq_num*	Operation sequence number
in	*handle*	Characteristic handle
in	*length*	Attribute length
in	*value*	Pointer to attribute data value

20.8.85 GATTC_SendEvtCfm

```
void GATTC_SendEvtCfm(uint8_t conidx, uint16_t handle)
```

Location: ble_gatt.h:357

GATTC send event confirmation.

Send a GATTC indication event confirmation. conidx Connection index handle Characteristic handle

Parameters

Direction	Name	Description
in	*conidx*	Connection index
in	*handle*	Characteristic handle

20.8.86 GATTC_MtuExchange

```
void GATTC_MtuExchange(uint8_t conidx)
```

Location: ble_gatt.h:366

Prepare and send GATTC_EXC_MTU_CMD to start MTU exchange procedure.

Prepare and send GATTC_EXC_MTU_CMD to start MTU exchange procedure. conidx Connection index

Parameters

Direction	Name	Description
in	*conidx*	Connection index

20.8.87 GATTC_ReadReqInd

```
void GATTC_ReadReqInd(ke_msg_id_t const msg_id, struct gattc_read_req_ind const * param,
ke_task_id_t const dest_id, ke_task_id_t const src_id)
```

Location: ble_gatt.h:380

GATTC handle read request indication.

Handle a received read request indication from GATT controller. msg_id Kernel message identifier param Pointer to constant message parameters (in format of structure gattc_read_req_ind) dest_id Destination task identifier src_id Source task identifier

NOTE: Indicate if the message was consumed, compare with KE_MSG_CONSUMED

NOTE: Indicate if the message was consumed, compare with KE_MSG_CONSUMED

Parameters

Direction	Name	Description
in	*msg_id*	Kernel message identifier
in	*param*	Pointer to constant message parameters (in format of structure gattc_read_req_ind)
in	*dest_id*	Destination task identifier
in	*src_id*	Source task identifier

20.8.88 GATTC_WriteReqInd

```
void GATTC_WriteReqInd(ke_msg_id_t const msg_id, struct gattc_write_req_ind const *
param, ke_task_id_t const dest_id, ke_task_id_t const src_id)
```

Location: ble_gatt.h:396

GATTC handle write request indication.

Handle a received write request indication from GATT controller. msg_id Kernel message identifier param Pointer to constant message parameters (in format of structure gattc_write_req_ind) dest_id Destination task identifier src_id Source task identifier

NOTE: Indicate if the message was consumed, compare with KE_MSG_CONSUMED

NOTE: Indicate if the message was consumed, compare with KE_MSG_CONSUMED

Parameters

Direction	Name	Description
`in`	*msg_id*	Kernel message identifier
`in`	*param*	Pointer to constant message parameters (in format of structure gattc_write_req_ind)
`in`	*dest_id*	Destination task identifier
`in`	*src_id*	Source task identifier

20.8.89 GATTC_AttInfoReqInd

```
void GATTC_AttInfoReqInd(ke_msg_id_t const msg_id, struct gattc_read_req_ind const *
param, ke_task_id_t const dest_id, ke_task_id_t const src_id)
```

Location: ble_gatt.h:415

GATTC request attribute information.

Request an attribute info indication to be passed to the upper layers of the Bluetooth stack. Could also be triggered during prepare write to check if attribute modification is authorized by profile/application or not and to get current attribute length. msg_id Kernel message identifier param Pointer to constant message parameters (in format of structure gattc_read_req_ind) dest_id Destination task identifier src_id Source task identifier

NOTE: Indicate if the message was consumed, compare with KE_MSG_CONSUMED

NOTE: Indicate if the message was consumed, compare with KE_MSG_CONSUMED

Parameters

Direction	Name	Description
in	*msg_id*	Kernel message identifier
in	*param*	Pointer to constant message parameters (in format of structure gattc_read_req_ind)
in	*dest_id*	Destination task identifier
in	*src_id*	Source task identifier

20.8.90 GATTC_MsgHandler

```
void GATTC_MsgHandler(ke_msg_id_t const msg_id, void const * param, ke_task_id_t const dest_id, ke_task_id_t const src_id)
```

Location: ble_gatt.h:430

Handle GATTM messages.

Message handler for GATT manager messages received from kernel msg_id Kernel message identifier param Pointer to constant message parameter (unused) dest_id Destination task identifier src_id Source task identifier

NOTE: Indicate if the message was consumed, compare with KE_MSG_CONSUMED

NOTE: Indicate if the message was consumed, compare with KE_MSG_CONSUMED

Parameters

Direction	Name	Description
in	*msg_id*	Kernel message identifier
in	*param*	Pointer to constant message parameter (unused)
in	*dest_id*	Destination task identifier
in	*src_id*	Source task identifier

20.8.91 Device_BLE_Public_Address_Read

```
void Device_BLE_Public_Address_Read(uint32_t ble_addr_location)
```

Location: ble_protocol_support.h:77

Read the Bluetooth public address to ble_public_addr.

Read the Bluetooth public address from a given location and save it to ble_public_addr (array). ble_addr_location Location to read value from

Parameters

Direction	Name	Description
`in`	*ble_addr_location*	Location to read value from

20.8.92 Device_BLE_Param_Get

```
uint8_t Device_BLE_Param_Get(uint8_t param_id, uint8_t * lengthPtr, uint8_t * buf)
```

Location: ble_protocol_support.h:90

Read BLE device parameters.

Read bluetooth low energy parameters, security keys, and channel assessment parameters that are provided by the application or NVR3. param_id Parameter identifier lengthPtr Pointer to the length of parameter buf Pointer to returned value

Returns:

Indicate if the requested parameter is provided by application

Returns:

Indicate if the requested parameter is provided by application

Parameters

Direction	Name	Description
`in`	*param_id*	Parameter identifier
`in`	*lengthPtr*	Pointer to the length of parameter
`in`	*buf*	Pointer to returned value

20.8.93 rand_func

```
int rand_func()
```

Location: ble_protocol_support.h:97

Generate a pseudo-random number.

Returns:

A pseudo-random number

20.8.94 set_app_rand_func

```
void set_app_rand_func(int(*)(void) func)
```

Location: ble_protocol_support.h:105

Sets random number generator function to be used by the BLE stack when calling rand_func()

Parameters

Direction	Name	Description
in	*func*	Function containing random number generator

20.8.95 srand_func

```
void srand_func(uint32_t seed)
```

Location: ble_protocol_support.h:112

Seeds pseudo-random number generator used by function rand()

Parameters

Direction	Name	Description
in	*seed*	Seed value

20.8.96 set_app_rand_seed_val

```
void set_app_rand_seed_val(uint32_t seed)
```

Location: ble_protocol_support.h:119

Sets seed value to be used by the BLE stack when calling srand_func()

Parameters

Direction	Name	Description
in	*seed*	Seed value

20.8.97 default_rf_rssi_func

```
int8_t default_rf_rssi_func(uint8_t rssi_reg)
```

Location: ble_protocol_support.h:129

Default RF to RSSI conversion function to be used by the BLE stack when calling Device_RF_RSSI_Convert().

This is in the event that APP_BLE_RSSI_CONVERSION_DEFINED is set to 1, but the user has not called set_app_rf_rssi_func(). rssi_reg RF data value set_app_rf_rssi_func().

Parameters

Direction	Name	Description
in	*rssi_reg*	RF data value

20.8.98 set_app_rf_rssi_func

```
void set_app_rf_rssi_func(int8_t(*)(uint8_t) func)
```

Location: ble_protocol_support.h:137

Sets RF to RSSI conversion function to be used by the BLE stack when calling Device_RF_RSSI_Convert()

Parameters

Direction	Name	Description
in	*func*	Function containing RF to RSSI calculations

20.8.99 MsgHandler_GetTaskAppDesc

```
const struct ke_task_desc * MsgHandler_GetTaskAppDesc()
```

Location: msg_handler.h:42

Provide an application task descriptor with default handler set to the MsgHandler_Notify function.

Returns:

A TASK_APP descriptor to be used in the ke_create_task function

20.8.100 MsgHandler_Add

```
bool MsgHandler_Add(ke_msg_id_t const msg_id, MsgHandlerCallback_t callback)
```

Location: msg_handler.h:58

Add/subscribe a message handler callback function.

If the msg_id argument is a message identifier (such as GAPM_CMP_EVT), the callback function is called only when the specific event is triggered
If the msg_id argument is a task identifier (such as TASK_ID_GAPM), the callback function is for for any event triggered from that task
msg_id A message identifier (such as GAPM_CMP_EVT) or a task identifier (such as TASK_ID_GAPM) callback A pointer to the callback function that should be called when an event matching msg_id happens
Returns:

True if it was able to add/register the handler, false otherwise

If the msg_id argument is a task identifier (such as TASK_ID_GAPM), the callback function is for for any event triggered from that task

Returns:

True if it was able to add/register the handler, false otherwise

Parameters

Direction	Name	Description
in	*msg_id*	A message identifier (such as GAPM_CMP_EVT) or a task identifier (such as TASK_ID_GAPM)
in	*callback*	A pointer to the callback function that should be called when an event matching msg_id happens

20.8.101 MsgHandler_Notify

```
int MsgHandler_Notify(ke_msg_id_t const msg_id, void const * param, ke_task_id_t const dest_id, ke_task_id_t const src_id)
```

Location: msg_handler.h:80

Notify the callback functions associated with the msg_id.

This function searches the list of handlers added using MsgHandler_Add and notify each one with matching msg_id. It also makes sure to call the BLE abstraction message handlers prior to any application message handler msg_id A message identifier (such as GAPM_CMP_EVT) or a task identifier (such as TASK_ID_GAPM) param Message parameter dest_id Destination task identifier src_id Source task identifier

Returns:

KE_MSG_CONSUMED back to the kernel

NOTE: This function was designed to be used as the default handler of the kernel and shall NOT be called directly by the application. To notify an event, the application should instead enqueue a message in the kernel, in order to avoid chaining the context of function calls and causing a stack overflow

It also makes sure to call the BLE abstraction message handlers prior to any application message handler

Returns:

KE_MSG_CONSUMED back to the kernel

NOTE: This function was designed to be used as the default handler of the kernel and shall NOT be called directly by the application. To notify an event, the application should instead enqueue a message in the kernel, in order to avoid chaining the context of function calls and causing a stack overflow

Parameters

Direction	Name	Description
in	*msg_id*	A message identifier (such as GAPM_CMP_EVT) or a task identifier (such as TASK_ID_GAPM)
in	*param*	Message parameter
in	*dest_id*	Destination task identifier
in	*src_id*	Source task identifier

20.8.102 BASC_Initialize

```
void BASC_Initialize()
```

Location: ble_basc.h:81

Initialize BASC environment.

Initialize battery service client environment and configure message handlers.

20.8.103 BASC_EnableReq

```
void BASC_EnableReq(uint8_t conidx)
```

Location: ble_basc.h:92

Enable the client role of the BAS.

Prepare and send BASC_ENABLE_REQ to enable the client role of the battery service. conidx Connection index

NOTE: conidx is valid (between 0 and GAP_MAX_CONNECTIONS-1)

NOTE: conidx is valid (between 0 and GAP_MAX_CONNECTIONS-1)

Parameters

Direction	Name	Description
in	*conidx*	Connection index

20.8.104 BASC_ReadInfoReq

```
void BASC_ReadInfoReq(uint8_t conidx, uint8_t bas_nb, uint8_t info)
```

Location: ble_basc.h:104

Prepare and send BASC_READ_INFO_REQ to read value from peer.

Prepare and send BASC_READ_INFO_REQ to read the value of characteristic or descriptor in the peer device database. conidx Connection index bas_nb Battery instance number info Peer battery info type

Parameters

Direction	Name	Description
in	*conidx*	Connection index
in	*bas_nb*	Battery instance number
in	*info*	Peer battery info type

20.8.105 BASC_BattLevelNtfCfgReq

```
void BASC_BattLevelNtfCfgReq(uint8_t conidx, uint8_t bas_nb, uint8_t ntf_cfg)
```

Location: ble_basc.h:116

Send a request to change notification configuration.

Prepare and send BASC_BATT_LEVEL_NTF_CFG_REQ to request to change battery level notification configuration in the peer device. conidx Connection index bas_nb Battery instance number ntf_cfg The value of notification configuration

Parameters

Direction	Name	Description
`in`	*conidx*	Connection index
`in`	*bas_nb*	Battery instance number
`in`	*ntf_cfg*	The value of notification configuration

20.8.106 BASC_RequestBattLevelOnTimeout

```
void BASC_RequestBattLevelOnTimeout(uint32_t timeout_ms)
```

Location: ble_basc.h:126

Send a request for periodic battery level update.

Configure a kernel timer to periodically send battery level requests. timeout_ms Delay between requests in units of 10ms. If timeout == 0, disable periodic requests.

Parameters

Direction	Name	Description
`in`	*timeout_ms*	Delay between requests in units of 10ms. If timeout == 0, disable periodic requests.

20.8.107 BASC_GetLastBatteryLevel

```
uint8_t BASC_GetLastBatteryLevel(uint8_t conidx, uint8_t bas_nb)
```

Location: ble_basc.h:137

Read the most recent battery level.

Read the most recent value of battery level from the BASC environment. conidx Connection index bas_nb Battery instance number

Returns:

Value of battery level

Returns:

Value of battery level

Parameters

Direction	Name	Description
in	*conidx*	Connection index
in	*bas_nb*	Battery instance number

20.8.108 BASC_GetEnv

```
const BASC_Env_t * BASC_GetEnv()
```

Location: ble_basc.h:146

Get the BASC environment.

Return a reference to the internal BASC environment structure.

Returns:

Constant pointer to BASC_Env_t

Returns:

Constant pointer to BASC_Env_t

20.8.109 BASC_MsgHandler

```
void BASC_MsgHandler(ke_msg_id_t const msg_id, void const * param, ke_task_id_t const dest_id, ke_task_id_t const src_id)
```

Location: ble_basc.h:158

Handle the BASC events.

Handle all the events related to battery service client. msg_id Kernel message identifier param Pointer to constant message parameter dest_id Destination task identifier src_id Source task identifier

Parameters

Direction	Name	Description
in	*msg_id*	Kernel message identifier
in	*param*	Pointer to constant message parameter
in	*dest_id*	Destination task identifier
in	*src_id*	Source task identifier

20.8.110 BASS_Initialize

```
void BASS_Initialize(uint8_t bas_nb, uint8_t(*)(uint8_t bas_nb) readBattLevelCallback)
```

Location: ble_bass.h:82

Initialize BASS environment.

Initialize a battery service server environment and configure message handlers. bas_nb Number of battery instances [1,2]. readBattLevelCallback Pointer to user application callback function that returns the battery value.

Parameters

Direction	Name	Description
in	*bas_nb*	Number of battery instances [1,2].
in	*readBattLevelCallback*	Pointer to user application callback function that returns the battery value.

20.8.111 BASS_NotifyOnTimeout

```
void BASS_NotifyOnTimeout(uint32_t timeout)
```

Location: ble_bass.h:93

BASS configure periodic notification.

Configure a battery service to send periodic notifications. timeout Timeout period in units of 10ms. disabled if timeout == 0

Parameters

Direction	Name	Description
`in`	*timeout*	Timeout period in units of 10ms. disabled if timeout == 0

20.8.112 BASS_NotifyOnBattLevelChange

```
void BASS_NotifyOnBattLevelChange(uint32_t timeout)
```

Location: ble_bass.h:107

BASS notify peers on battery level change.

Configure battery service to periodically monitor the battery level according to the timeout. The battery level is queried read through the callback function passed as argument to BASS_Initialize(). Connected peers are only notified if the battery level changes. It triggers a BASS_BATT_LEVEL_CHANGED event, which the user application may subscribe to. timeout Timeout period in units of 10ms. disabled if timeout == 0 BASS_Initialize(). Connected peers are only notified if the battery level changes. It triggers a BASS_BATT_LEVEL_CHANGED event, which the user application may subscribe to.

Parameters

Direction	Name	Description
`in`	*timeout*	Timeout period in units of 10ms. disabled if timeout == 0

20.8.113 BASS_ProfileTaskAddCmd

```
void BASS_ProfileTaskAddCmd()
```

Location: ble_bass.h:114

Add a battery profile in kernel.

Send request to add a battery profile in kernel and database.

20.8.114 BASS_EnableReq

```
void BASS_EnableReq(uint8_t conidx)
```

Location: ble_bass.h:124

Enable the server role in BAS.

Prepare and send BASS_ENABLE_REQ to enable the server role of the battery service. conidx Connection index

Parameters

Direction	Name	Description
in	*conidx*	Connection index

20.8.115 BASS_BattLevelUpdReq

```
void BASS_BattLevelUpdReq(uint8_t batt_lvl, uint8_t bas_nb)
```

Location: ble_bass.h:137

Prepare and send BASS_BATT_LEVEL_UPD_REQ for battery level notification.

Prepare and send BASS_BATT_LEVEL_UPD_REQ to update the battery level characteristic value for the BAS instance. This value will be stored in the database so that it can be read by the peer device. batt_lvl Battery level [0, 100] bas_nb Battery server instance number

Parameters

Direction	Name	Description
in	*batt_lvl*	Battery level [0, 100]
in	*bas_nb*	Battery server instance number

20.8.116 BASS_GetEnv

```
const BASS_Env_t * BASS_GetEnv()
```

Location: ble_bass.h:146

Get the BASS environment.

Return a reference to the internal BASS environment structure.

Returns:

Constant pointer to BASS_Env_t

Returns:

Constant pointer to BASS_Env_t

20.8.117 BASS_IsAdded

```
bool BASS_IsAdded()
```

Location: ble_bass.h:156

Check if the BASS is added successfully.

Returns a boolean indication if the BAS server has been successfully added to the services database.

Returns:

True if the service has been successfully added, false otherwise

Returns:

True if the service has been successfully added, false otherwise

20.8.118 BASS_MsgHandler

```
void BASS_MsgHandler(ke_msg_id_t const msg_id, void const * param, ke_task_id_t const
dest_id, ke_task_id_t const src_id)
```

Location: ble_bass.h:168

Handle BASS messages.

Handle all events related to the battery service server msg_id Kernel message identifier param Pointer to constant message parameters dest_id Destination task identifier src_id Source task identifier

Parameters

Direction	Name	Description
in	*msg_id*	Kernel message identifier
in	*param*	Pointer to constant message parameters
in	*dest_id*	Destination task identifier
in	*src_id*	Source task identifier

20.8.119 DISS_Initialize

```
int DISS_Initialize(uint16_t features, const struct DISS_DeviceInfo_t * deviceInfo)
```

Location: ble_diss.h:81

Initialize the device information server service environment and configure message handlers.

Returns:

0 if the initialization was successful, < 0 otherwise

Parameters

Direction	Name	Description
in	*features*	The enabled features for the device information service. Values define as enum diss_features in diss_task.h.
in	*deviceInfo*	Constant pointer to DISS_DeviceInfo_t

20.8.120 DISS_ProfileTaskAddCmd

```
void DISS_ProfileTaskAddCmd()
```

Location: ble_diss.h:87

Send a request to add the device information service in kernel and database.

20.8.121 DISS_EnvGet

```
const DISS_Env_t * DISS_EnvGet()
```

Location: ble_diss.h:94

Return a reference to the internal DISS environment structure.

Returns:

A constant pointer to DISS_Env_t

20.8.122 DISS_IsAdded

```
bool DISS_IsAdded()
```

Location: ble_diss.h:102

Return a boolean indication if the DISS server has been successfully added to the services database.

Returns:

True if the service is added successful, false otherwise

20.8.123 DISS_MsgHandler

```
void DISS_MsgHandler(ke_msg_id_t const msg_id, void const * param, ke_task_id_t const
dest_id, ke_task_id_t const src_id)
```

Location: ble_diss.h:112

Handle all the events related to the device information service server.

Parameters

Direction	Name	Description
in	*msg_id*	Kernel message identifier
in	*param*	Message parameter
in	*dest_id*	Destination task identifier
in	*src_id*	Source task identifier

20.8.124 DISS_DeviceInfoValueReqInd

```
void DISS_DeviceInfoValueReqInd(ke_msg_id_t const msgid, void const * param, ke_task_id_t const dest_id, ke_task_id_t const src_id)
```

Location: ble_diss.h:123

Handle the indication when the peer device requests a value.

Parameters

Direction	Name	Description
in	*msgid*	Kernel message identifier
in	*param*	Message parameters in format of struct diss_value_req_ind
in	*dest_id*	Destination task identifier
in	*src_id*	Source task identifier

CHAPTER 21

BLE_ABSTRACTION

21.1 Summary

Data Structures

- BondInfo_t : Bond Information.

Macros

- BOND_INFO_BASE : Start address and size of the memory region where bond list is stored:
- BOND_INFO_SIZE : Each Bond takes up 72 Bytes of the FLASH.
- STATIC_ASSERT
- BONDLIST_MAX_SIZE
- BOND_INFO_FLASH_SECTORS_COUNT
- BOND_INFO_STATE_INVALID : Invalid bond info state.
- BOND_INFO_STATE_EMPTY : Empty bond info state.
- BOND_INFO_STATE_VALID : Macro for valid bond info state.

Functions

- STATIC_ASSERT
- BondList_Size : Bondlist functions.
- BondList_GetIRKs : Get the IRKs from bond lists.
- BondList_FindByIRK : Search for the bond information matching specified IRK in flash.
- BondList_FindByAddr : Search for the bonding information for a peer device in flash matching specified address and address type.
- BondList_FlashDefrag : Squeeze bond information together to make space for a new entry.
- BondList_Add : Add bond information to the bond list.
- BondList_Remove : Remove a bond list entry matching the specified index.
- BondList_RemoveAll : Erase bond list sectors containing bond list information in flash.

21.2 BLE_ABSTRACTION Data Structures Type Documentation

21.2.1 BondInfo_t

Location: bondlist.h:39

Bond Information.

Data Fields

Type	*Name*	Description
uint16_t	*state*	State of bond.
uint8_t	*pairing_lvl*	Pairing level.
uint8_t	*csrk_exchanged*	Non-zero if CSRK has been exchanged.
uint8_t	*ltk[16]*	Long term key.
uint16_t	*ediv*	Encryption diversifier.
uint8_t	*reserved1[2]*	Reserved.
uint8_t	*addr[6]*	Peer address.
uint8_t	*addr_type*	Address type.
uint8_t	*irk_exchanged*	Non-zero if IRK has been exchanged.
uint8_t	*csrk[16]*	Connection resolving signature key.
uint8_t	*irk[16]*	Identity resolving key.
uint8_t	*rand[8]*	Random number.

21.3 BLE_ABSTRACTION Macro Definition Documentation

21.3.1 BOND_INFO_BASE

```
#define BOND_INFO_BASE FLASH_BOND_INFO_BASE
```

Location: bondlist.h:67

Start address and size of the memory region where bond list is stored:

- BOND_INFO_BASE: 3KB from base of data flash array (the first 3KB of data flash array is reserved to ROM use)
- BOND_INFO_FLASH_SECTORS_COUNT: Defaulted to 8 sectors (2KB) Notes: if needed, user application can re-define BOND_INFO_BASE and BOND_INFO_FLASH_SECTORS_COUNT. Any increase to the number of sectors will also need to be updated in the linker script. Start of address for bond info

21.3.2 BOND_INFO_SIZE

```
#define BOND_INFO_SIZE (72)
```

Location: bondlist.h:74

Each Bond takes up 72 Bytes of the FLASH.

21.3.3 STATIC_ASSERT

```
#define STATIC_ASSERT typedef char static_assertion_failed_##MSG[(COND) ? 1 : -1]
```

Location: bondlist.h:76

21.3.4 BONDLIST_MAX_SIZE

```
#define BONDLIST_MAX_SIZE (28)
```

Location: bondlist.h:85

21.3.5 BOND_INFO_FLASH_SECTORS_COUNT

```
#define BOND_INFO_FLASH_SECTORS_COUNT ((BONDLIST_MAX_SIZE * BOND_INFO_SIZE) >> 8)
```

Location: bondlist.h:93

21.3.6 BOND_INFO_STATE_INVALID

```
#define BOND_INFO_STATE_INVALID 0x00
```

Location: bondlist.h:98

Invalid bond info state.

21.3.7 BOND_INFO_STATE_EMPTY

```
#define BOND_INFO_STATE_EMPTY 0xFFFF
```

Location: bondlist.h:101

Empty bond info state.

21.3.8 BOND_INFO_STATE_VALID

```
#define BOND_INFO_STATE_VALID ((state != BOND_INFO_STATE_INVALID) && \
                                       (state != BOND_INFO_STATE_EMPTY) &&   \
                                       (state <= BONDLIST_MAX_SIZE))
```

Location: bondlist.h:104

Macro for valid bond info state.

21.4 BLE_ABSTRACTION FUNCTION DOCUMENTATION

21.4.1 STATIC_ASSERT

```
STATIC_ASSERT()
```

Location: bondlist.h:81

21.4.2 BondList_Size

```
uint8_t BondList_Size()
```

Location: bondlist.h:117

Bondlist functions.

Get the number of entries in the bond list stored to flash

Returns:

Number of entries in the the bond list with a valid bond state

Returns:

Number of entries in the the bond list with a valid bond state

21.4.3 BondList_GetIRKs

```
uint8_t BondList_GetIRKs(struct gap_sec_key * irks)
```

Location: bondlist.h:127

Get the IRKs from bond lists.

Get the IRKs from bond lists having valid bond state irks Pointer to gap_sec_key structure

Returns:

Number of IRKs found

Returns:

Number of IRKs found

Parameters

Direction	Name	Description
in	*irks*	Pointer to gap_sec_key structure

21.4.4 BondList_FindByIRK

```
const BondInfo_t * BondList_FindByIRK(const uint8_t * irk)
```

Location: bondlist.h:136

Search for the bond information matching specified IRK in flash.

Returns:

If found an entry in flash, return its address as a pointer to a const BondInfo_t element NULL otherwise

Parameters

Direction	Name	Description
in	*irk*	Matching IRK to search for

21.4.5 BondList_FindByAddr

```
const BondInfo_t * BondList_FindByAddr(const uint8_t * addr, uint8_t addrType)
```

Location: bondlist.h:147

Search for the bonding information for a peer device in flash matching specified address and address type.

Returns:

If found an entry in flash, return its address as a pointer to a const BondInfo_t element NULL otherwise

Parameters

Direction	Name	Description
in	*addr*	Address of the peer device to search
in	*addrType*	Address type of the peer device to search

21.4.6 BondList_FlashDefrag

```
bool BondList_FlashDefrag()
```

Location: bondlist.h:154

Squeeze bond information together to make space for a new entry.

Returns:

True if successful, false otherwise

21.4.7 BondList_Add

```
uint16_t BondList_Add(BondInfo_t * bond_info)
```

Location: bondlist.h:163

Add bond information to the bond list.

Returns:

Non-zero if bond information is added to bond list successfully, false otherwise

Parameters

Direction	Name	Description
`in`	*bond_info*	Pointer to bond information

21.4.8 BondList_Remove

```
bool BondList_Remove(uint16_t bondStateIndex)
```

Location: bondlist.h:171

Remove a bond list entry matching the specified index.

Returns:

True if successful, false otherwise

Parameters

Direction	Name	Description
`in`	*bondStateIndex*	Index of bond list to remove

21.4.9 BondList_RemoveAll

```
bool BondList_RemoveAll()
```

Location: bondlist.h:179

Erase bond list sectors containing bond list information in flash.

Returns:

If found an error erasing any sector, return false true otherwise

Copyright 2023 Semiconductor Components Industries, LLC ("onsemi"). All rights reserved. Unless agreed to differently in a separate onsemi license agreement, onsemi is providing this "Technology" (e.g. reference design kit, development product, prototype, sample, any other non-production product, software, design-IP, evaluation board, etc.) "AS IS" and does not assume any liability arising from its use; nor does onsemi convey any license to its or any third party's intellectual property rights. This Technology is provided only to assist users in evaluation of the Technology and the recipient assumes all liability and risk associated with its use, including, but not limited to, compliance with all regulatory standards. onsemi reserves the right to make changes without further notice to any of the Technology.

The Technology is not a finished product and is as such not available for sale to consumers. Unless agreed otherwise in a separate agreement, the Technology is only intended for research, development, demonstration and evaluation purposes and should only be used in laboratory or development areas by persons with technical training and familiarity with the risks associated with handling electrical/mechanical components, systems and subsystems. The user assumes full responsibility/liability for proper and safe handling. Any other use, resale or redistribution for any other purpose is strictly prohibited.

The Technology is not designed, intended, or authorized for use in life support systems, or any FDA Class 3 medical devices or medical devices with a similar or equivalent classification in a foreign jurisdiction, or any devices intended for implantation in the human body. Should you purchase or use the Technology for any such unintended or unauthorized application, you shall indemnify and hold onsemi and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that onsemi was negligent regarding the design or manufacture of the board.

The Technology does not fall within the scope of the European Union directives regarding electromagnetic compatibility, restricted substances (RoHS), recycling (WEEE), FCC, CE or UL, and may not meet the technical requirements of these or other related directives.

THE TECHNOLOGY IS NOT WARRANTED AND PROVIDED ON AN "AS IS" BASIS ONLY. ANY WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT, ARE HEREBY EXPRESSLY DISCLAIMED.

TO THE FULLEST EXTENT PERMITTED BY APPLICABLE LAW, IN NO EVENT SHALL ONSEMI BE LIABLE TO CUSTOMER OR ANY THIRD PARTY. IN NO EVENT SHALL ONSEMI BE LIABLE FOR SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES OF ANY NATURE WHATSOEVER (INCLUDING, BUT NOT LIMITED TO, LOSS OR DISGORGEMENT OF PROFITS, LOSS OF USE AND LOSS OF GOODWILL), REGARDLESS OF WHETHER ONSEMI HAS BEEN GIVEN NOTICE OF ANY SUCH ALLEGED DAMAGES, AND REGARDLESS OF WHETHER SUCH ALLEGED DAMAGES ARE SOUGHT UNDER CONTRACT, TORT OR OTHER THEORIES OF LAW.

Do not use this Technology unless you have carefully read and agree to these limited terms and conditions. By using this Technology, you expressly agree to the limited terms and conditions. All source code is onsemi proprietary and confidential information.

PUBLICATION ORDERING INFORMATION

LITERATURE FULFILLMENT:

Literature Distribution Center for onsemi

19521 E. 32nd Pkwy, Aurora, Colorado 80011 USA

Phone: 303-675-2175 or 800-344-3860 Toll Free USA/Canada

Fax: 303-675-2176 or 800-344-3867 Toll Free USA/Canada

Email: orderlit@onsemi.com

N. American Technical Support:

800-282-9855 Toll Free USA/Canada

Europe, Middle East and Africa Technical Support: Phone: 421 33 790 2910

onsemi Website: www.onsemi.com

Order Literature: http://www.onsemi.com/orderlit

For additional information, please contact your local Sales Representative

M-20871-005