

Kinetis SDK v.2.0 API Reference Manual

NXP Semiconductors

Document Number: MCUXSDKAPIRM

Rev. 0

Dec 2019

Contents

Chapter [Introduction](#)

Chapter [Driver errors status](#)

Chapter [Trademarks](#)

Chapter [Architectural Overview](#)

Chapter [ADC: 12-bit SAR Analog-to-Digital Converter Driver](#)

5.1	Overview	11
5.2	Typical use case	11
5.2.1	Polling Configuration	11
5.2.2	Interrupt Configuration	11
5.3	Data Structure Documentation	15
5.3.1	struct adc_config_t	15
5.3.2	struct adc_conv_seq_config_t	16
5.3.3	struct adc_result_info_t	17
5.4	Macro Definition Documentation	17
5.4.1	FSL_ADC_DRIVER_VERSION	17
5.5	Enumeration Type Documentation	17
5.5.1	_adc_status_flags	17
5.5.2	_adc_interrupt_enable	19
5.5.3	adc_clock_mode_t	19
5.5.4	adc_resolution_t	19
5.5.5	adc_trigger_polarity_t	19
5.5.6	adc_priority_t	20
5.5.7	adc_seq_interrupt_mode_t	20
5.5.8	adc_threshold_compare_status_t	20
5.5.9	adc_threshold_crossing_status_t	20
5.5.10	adc_threshold_interrupt_mode_t	20
5.5.11	adc_inforeresult_t	21
5.5.12	adc_tempsensor_common_mode_t	21
5.5.13	adc_second_control_t	21

Contents

Section Number	Title	Page Number
5.6	Function Documentation	21
5.6.1	ADC_Init	21
5.6.2	ADC_Deinit	22
5.6.3	ADC_GetDefaultConfig	22
5.6.4	ADC_EnableTemperatureSensor	22
5.6.5	ADC_EnableConvSeqA	23
5.6.6	ADC_SetConvSeqAConfig	23
5.6.7	ADC_DoSoftwareTriggerConvSeqA	23
5.6.8	ADC_EnableConvSeqABurstMode	23
5.6.9	ADC_SetConvSeqAHighPriority	24
5.6.10	ADC_EnableConvSeqB	24
5.6.11	ADC_SetConvSeqBConfig	24
5.6.12	ADC_DoSoftwareTriggerConvSeqB	24
5.6.13	ADC_EnableConvSeqBBurstMode	25
5.6.14	ADC_SetConvSeqBHighPriority	25
5.6.15	ADC_GetConvSeqAGlobalConversionResult	25
5.6.16	ADC_GetConvSeqBGlobalConversionResult	26
5.6.17	ADC_GetChannelConversionResult	26
5.6.18	ADC_SetThresholdPair0	26
5.6.19	ADC_SetThresholdPair1	27
5.6.20	ADC_SetChannelWithThresholdPair0	27
5.6.21	ADC_SetChannelWithThresholdPair1	27
5.6.22	ADC_EnableInterrupts	27
5.6.23	ADC_DisableInterrupts	28
5.6.24	ADC_EnableShresholdCompareInterrupt	28
5.6.25	ADC_EnableThresholdCompareInterrupt	28
5.6.26	ADC_GetStatusFlags	28
5.6.27	ADC_ClearStatusFlags	29
Chapter	AES: AES encryption decryption driver	
6.1	Overview	31
6.2	AES Driver Initialization and Configuration	31
6.3	Comments about API usage in RTOS	31
6.4	AES Driver Examples	31
6.5	Macro Definition Documentation	32
6.5.1	FSL_AES_DRIVER_VERSION	32
6.6	Function Documentation	33
6.6.1	AES_SetKey	33
6.6.2	AES_EncryptEcb	33

Contents

Section Number	Title	Page Number
6.6.3	AES_DecryptEcb	33
6.6.4	AES_EncryptCbc	34
6.6.5	AES_DecryptCbc	34
6.6.6	AES_EncryptCfb	35
6.6.7	AES_DecryptCfb	35
6.6.8	AES_EncryptOfb	36
6.6.9	AES_DecryptOfb	36
6.6.10	AES_CryptCtr	37
6.6.11	AES_EncryptTagGcm	37
6.6.12	AES_DecryptTagGcm	38
Chapter	Clock: Clock driver	
7.1	Overview	41
7.2	Function groups	41
7.2.1	SYSCON Clock frequency functions	41
7.2.2	SYSCON clock Selection Muxes	41
7.2.3	SYSCON clock dividers	41
7.3	Data Structure Documentation	49
7.3.1	struct ClockCapacitanceCompensation_t	49
7.4	Macro Definition Documentation	49
7.4.1	FSL_CLOCK_DRIVER_VERSION	49
7.4.2	FLEXCOMM_CLOCKS	49
7.4.3	CTIMER_CLOCKS	50
7.4.4	GINT_CLOCKS	50
7.4.5	WWDT_CLOCKS	50
7.4.6	DMIC_CLOCKS	50
7.4.7	ADC_CLOCKS	51
7.4.8	SPIFI_CLOCKS	51
7.4.9	GPIO_CLOCKS	51
7.4.10	DMA_CLOCKS	51
7.5	Enumeration Type Documentation	51
7.5.1	CHIP_SYSCON_MAINCLKSRC_T	51
7.5.2	CHIP_SYSCON_FRGCLKSRC_T	52
7.5.3	clock_name_t	52
7.5.4	clock_sel_ofst_t	53
7.5.5	clock_attach_id_t	54
7.5.6	clock_div_name_t	55
7.5.7	main_clock_src_t	56
7.5.8	clkout_clock_src_t	56
7.5.9	wdt_clock_src_t	56

Contents

Section Number	Title	Page Number
7.5.10	frg_clock_src_t	56
7.5.11	apb_clock_src_t	57
7.5.12	fmeas_clock_src_t	57
7.5.13	spifi_clock_src_t	57
7.5.14	adc_clock_src_t	58
7.5.15	pwm_clock_source_t	58
7.5.16	Fro_ClkSel_t	58
7.6	Function Documentation	58
7.6.1	CLOCK_GetFreq	58
7.6.2	CLOCK_AttachClk	58
7.6.3	CLOCK_SetClkDiv	59
7.6.4	CLOCK_EnableClock	59
7.6.5	CLOCK_DisableClock	60
7.6.6	CLOCK_IsClockEnable	60
7.6.7	CLOCK_GetApbCLKFreq	60
7.6.8	CLOCK_GetSpifiClkFreq	61
7.6.9	CLOCK_uDelay	61
7.6.10	CLOCK_XtalBasicTrim	61
7.6.11	CLOCK_Xtal32M_Trim	62
7.6.12	CLOCK_Xtal32k_Trim	63
7.6.13	CLOCK_SetXtal32M_LDO	63
7.6.14	CLOCK_Xtal32M_WaitUntilStable	64
Chapter	Common Driver	
8.1	Overview	65
8.2	Macro Definition Documentation	70
8.2.1	FSL_RESET_DRIVER_VERSION	70
8.2.2	ADC_RSTS	70
8.2.3	MAKE_STATUS	71
8.2.4	MAKE_VERSION	71
8.2.5	FSL_COMMON_DRIVER_VERSION	71
8.2.6	DEBUG_CONSOLE_DEVICE_TYPE_NONE	71
8.2.7	DEBUG_CONSOLE_DEVICE_TYPE_UART	71
8.2.8	DEBUG_CONSOLE_DEVICE_TYPE_LPUART	71
8.2.9	DEBUG_CONSOLE_DEVICE_TYPE_LPSCI	71
8.2.10	DEBUG_CONSOLE_DEVICE_TYPE_USBCDC	71
8.2.11	DEBUG_CONSOLE_DEVICE_TYPE_FLEXCOMM	71
8.2.12	DEBUG_CONSOLE_DEVICE_TYPE_IUART	71
8.2.13	DEBUG_CONSOLE_DEVICE_TYPE_VUSART	71
8.2.14	DEBUG_CONSOLE_DEVICE_TYPE_MINI_USART	71
8.2.15	DEBUG_CONSOLE_DEVICE_TYPE_SWO	71
8.2.16	ARRAY_SIZE	71

Contents

Section Number	Title	Page Number
8.3	Typedef Documentation	71
8.3.1	status_t	71
8.4	Enumeration Type Documentation	71
8.4.1	SYSCON_RSTn_t	71
8.4.2	_status_groups	73
8.4.3	anonymous enum	75
8.5	Function Documentation	75
8.5.1	RESET_SetPeripheralReset	75
8.5.2	RESET_ClearPeripheralReset	75
8.5.3	RESET_PeripheralReset	75
8.5.4	RESET_SystemReset	76
8.5.5	EnableIRQ	76
8.5.6	DisableIRQ	76
8.5.7	DisableGlobalIRQ	77
8.5.8	EnableGlobalIRQ	77
8.5.9	EnableDeepSleepIRQ	77
8.5.10	DisableDeepSleepIRQ	78
8.5.11	SDK_Malloc	78
8.5.12	SDK_Free	78
8.5.13	SDK_DelayAtLeastUs	79
Chapter	CTIMER: Standard counter/timers	
9.1	Overview	81
9.2	Function groups	81
9.2.1	Initialization and deinitialization	81
9.2.2	PWM Operations	81
9.2.3	Match Operation	81
9.2.4	Input capture operations	81
9.3	Typical use case	82
9.3.1	Match example	82
9.3.2	PWM output example	82
9.4	Data Structure Documentation	85
9.4.1	struct ctimer_match_config_t	85
9.4.2	struct ctimer_config_t	85
9.5	Enumeration Type Documentation	85
9.5.1	ctimer_capture_channel_t	85
9.5.2	ctimer_capture_edge_t	86
9.5.3	ctimer_match_t	86
9.5.4	ctimer_match_output_control_t	86

Contents

Section Number	Title	Page Number
9.5.5	ctimer_interrupt_enable_t	86
9.5.6	ctimer_status_flags_t	87
9.5.7	ctimer_callback_type_t	87
9.6	Function Documentation	87
9.6.1	CTIMER_Init	87
9.6.2	CTIMER_Deinit	87
9.6.3	CTIMER_GetDefaultConfig	88
9.6.4	CTIMER_SetupPwmPeriod	88
9.6.5	CTIMER_SetupPwm	89
9.6.6	CTIMER_UpdatePwmPulsePeriod	89
9.6.7	CTIMER_UpdatePwmDutycycle	90
9.6.8	CTIMER_SetupMatch	90
9.6.9	CTIMER_SetupCapture	90
9.6.10	CTIMER_GetTimerCountValue	91
9.6.11	CTIMER_RegisterCallBack	92
9.6.12	CTIMER_EnableInterrupts	92
9.6.13	CTIMER_DisableInterrupts	92
9.6.14	CTIMER_GetEnabledInterrupts	93
9.6.15	CTIMER_GetStatusFlags	94
9.6.16	CTIMER_ClearStatusFlags	94
9.6.17	CTIMER_StartTimer	94
9.6.18	CTIMER_StopTimer	94
9.6.19	CTIMER_Reset	95
Chapter	Debug Console	
10.1	Overview	97
10.2	Function groups	97
10.2.1	Initialization	97
10.2.2	Advanced Feature	98
10.3	Typical use case	101
10.4	Function groups	103
10.4.1	Initialization	103
10.4.2	Advanced Feature	103
10.5	Typical use case	107
10.6	Macro Definition Documentation	110
10.6.1	DEBUGCONSOLE_REDIRECT_TO_TOOLCHAIN	110
10.6.2	DEBUGCONSOLE_REDIRECT_TO_SDK	110
10.6.3	DEBUGCONSOLE_DISABLE	110
10.6.4	SDK_DEBUGCONSOLE	110

Contents

Section Number	Title	Page Number
10.6.5	SDK_DEBUGCONSOLE_UART	110
10.6.6	PRINTF	110
10.7	Function Documentation	110
10.7.1	DbgConsole_Init	110
10.7.2	DbgConsole_Deinit	111
10.7.3	DbgConsole_Printf	111
10.7.4	DbgConsole_Putchar	111
10.7.5	DbgConsole_Scanf	112
10.7.6	DbgConsole_Getchar	112
10.7.7	DbgConsole_Flush	113
10.7.8	StrFormatPrintf	113
10.7.9	StrFormatScanf	113
10.8	Semihosting	115
10.8.1	Guide Semihosting for IAR	115
10.8.2	Guide Semihosting for Keil μ Vision	115
10.8.3	Guide Semihosting for MCUXpresso IDE	116
10.8.4	Guide Semihosting for ARMGCC	116
10.8.5	Guide Semihosting for IAR	118
10.8.6	Guide Semihosting for Keil μ Vision	119
10.8.7	Guide Semihosting for ARMGCC	119
10.9	SWO	121
10.9.1	Guide SWO for SDK	121
10.9.2	Guide SWO for Keil μ Vision	122
10.9.3	Guide SWO for MCUXpresso IDE	123
10.9.4	Guide SWO for ARMGCC	123
Chapter	DMA: Direct Memory Access Controller Driver	
11.1	Overview	125
11.2	Typical use case	125
11.2.1	DMA Operation	125
11.3	Data Structure Documentation	130
11.3.1	struct dma_descriptor_t	130
11.3.2	struct dma_xfercfg_t	130
11.3.3	struct dma_channel_trigger_t	131
11.3.4	struct dma_channel_config_t	131
11.3.5	struct dma_transfer_config_t	132
11.3.6	struct dma_handle_t	132
11.4	Macro Definition Documentation	132
11.4.1	FSL_DMA_DRIVER_VERSION	132

Contents

Section Number	Title	Page Number
11.4.2	DMA_ALLOCATE_HEAD_DESCRIPTORs	133
11.4.3	DMA_ALLOCATE_HEAD_DESCRIPTORs_AT_NONCACHEABLE	134
11.4.4	DMA_ALLOCATE_LINK_DESCRIPTORs	134
11.4.5	DMA_ALLOCATE_LINK_DESCRIPTORs_AT_NONCACHEABLE	134
11.4.6	DMA_DESCRIPTOR_END_ADDRESS	134
11.4.7	DMA_CHANNEL_XFER	135
11.5	Typedef Documentation	135
11.5.1	dma_callback	135
11.6	Enumeration Type Documentation	136
11.6.1	_dma_transfer_status	136
11.6.2	_dma_addr_interleave_size	136
11.6.3	_dma_transfer_width	136
11.6.4	dma_priority_t	136
11.6.5	dma_irq_t	137
11.6.6	dma_trigger_type_t	137
11.6.7	_dma_burst_size	137
11.6.8	dma_trigger_burst_t	137
11.6.9	dma_burst_wrap_t	138
11.6.10	dma_transfer_type_t	138
11.7	Function Documentation	138
11.7.1	DMA_Init	138
11.7.2	DMA_Deinit	138
11.7.3	DMA_InstallDescriptorMemory	139
11.7.4	DMA_ChannelIsActive	139
11.7.5	DMA_ChannelIsBusy	139
11.7.6	DMA_EnableChannelInterrupts	140
11.7.7	DMA_DisableChannelInterrupts	141
11.7.8	DMA_EnableChannel	141
11.7.9	DMA_DisableChannel	141
11.7.10	DMA_EnableChannelPeriphRq	141
11.7.11	DMA_DisableChannelPeriphRq	142
11.7.12	DMA_ConfigureChannelTrigger	142
11.7.13	DMA_SetChannelConfig	142
11.7.14	DMA_GetRemainingBytes	143
11.7.15	DMA_SetChannelPriority	144
11.7.16	DMA_GetChannelPriority	144
11.7.17	DMA_SetChannelConfigValid	144
11.7.18	DMA_DoChannelSoftwareTrigger	145
11.7.19	DMA_LoadChannelTransferConfig	145
11.7.20	DMA_CreateDescriptor	145
11.7.21	DMA_SetupDescriptor	146
11.7.22	DMA_SetupChannelDescriptor	147

Contents

Section Number	Title	Page Number
11.7.23	DMA_LoadChannelDescriptor	147
11.7.24	DMA_AbortTransfer	148
11.7.25	DMA_CreateHandle	148
11.7.26	DMA_SetCallback	148
11.7.27	DMA_PrepareTransfer	149
11.7.28	DMA_PrepareChannelTransfer	149
11.7.29	DMA_SubmitTransfer	150
11.7.30	DMA_SubmitChannelTransferParameter	150
11.7.31	DMA_SubmitChannelDescriptor	151
11.7.32	DMA_SubmitChannelTransfer	152
11.7.33	DMA_StartTransfer	153
11.7.34	DMA_IRQHandle	153
Chapter	DMIC: Digital Microphone	
12.1	Overview	155
12.2	Function groups	155
12.2.1	Initialization and deinitialization	155
12.2.2	Configuration	155
12.2.3	DMIC Data and status	155
12.2.4	DMIC Interrupt Functions	155
12.2.5	DMIC HWVAD Functions	156
12.2.6	DMIC HWVAD Interrupt Functions	156
12.3	Typical use case	156
12.3.1	DMIC DMA Configuration	156
12.3.2	DMIC use case	156
12.4	DMIC Driver	157
12.4.1	Overview	157
12.4.2	Data Structure Documentation	160
12.4.3	Macro Definition Documentation	161
12.4.4	Typedef Documentation	161
12.4.5	Enumeration Type Documentation	161
12.4.6	Function Documentation	163
12.5	DMIC DMA Driver	175
12.5.1	Overview	175
12.5.2	Data Structure Documentation	176
12.5.3	Macro Definition Documentation	177
12.5.4	Typedef Documentation	177
12.5.5	Function Documentation	177

Contents

Section Number	Title	Page Number
Chapter	FLASH: Flash driver	
13.1	Overview	181
13.2	Data Structure Documentation	183
13.2.1	struct IMG_HEADER_T	183
13.2.2	struct BOOT_BLOCK_T	184
13.2.3	struct flash_config_t	184
13.3	Macro Definition Documentation	185
13.3.1	FLASH_FAIL	185
13.3.2	FLASH_FAIL	185
13.4	Enumeration Type Documentation	185
13.4.1	flash_status_t	185
13.4.2	flash_read_mode_t	185
13.5	Function Documentation	185
13.5.1	FLASH_Init	185
13.5.2	FLASH_Powerdown	186
13.5.3	FLASH_Wait	186
13.5.4	FLASH_Erase	186
13.5.5	FLASH_ErasePages	187
13.5.6	FLASH_BlankCheck	187
13.5.7	FLASH_MarginCheck	188
13.5.8	FLASH_Program	188
13.5.9	FLASH_Checksum	189
13.5.10	FLASH_Read	189
13.5.11	FLASH_SetReadMode	190
13.5.12	FLASH_CalculateChecksum	190
13.5.13	FLASH_ConfigPageVerifyPageChecksum	191
13.5.14	FLASH_ConfigPageVerifyGpoChecksum	191
13.5.15	FLASH_ConfigPageUpdate	192
13.5.16	FLASH_GetStatus	193
13.6	Variable Documentation	193
13.6.1	img_len	193
Chapter	FLEXCOMM: FLEXCOMM Driver	
14.1	Overview	195
14.2	FLEXCOMM Driver	196
14.2.1	Overview	196
14.2.2	Macro Definition Documentation	197
14.2.3	Typedef Documentation	197

Contents

Section Number	Title	Page Number
14.2.4	Enumeration Type Documentation	197
14.2.5	Function Documentation	197
14.2.6	Variable Documentation	197
Chapter	I2C: Inter-Integrated Circuit Driver	
15.1	Overview	199
15.2	Typical use case	199
15.2.1	Master Operation in functional method	199
15.2.2	Master Operation in interrupt transactional method	200
15.2.3	Master Operation in DMA transactional method	201
15.2.4	Slave Operation in functional method	201
15.2.5	Slave Operation in interrupt transactional method	202
15.3	I2C Driver	204
15.3.1	Overview	204
15.3.2	Macro Definition Documentation	205
15.3.3	Enumeration Type Documentation	205
15.4	I2C Master Driver	206
15.4.1	Overview	206
15.4.2	Data Structure Documentation	208
15.4.3	Typedef Documentation	211
15.4.4	Enumeration Type Documentation	212
15.4.5	Function Documentation	213
15.5	I2C Slave Driver	223
15.5.1	Overview	223
15.5.2	Data Structure Documentation	225
15.5.3	Typedef Documentation	228
15.5.4	Enumeration Type Documentation	228
15.5.5	Function Documentation	230
15.6	I2C DMA Driver	238
15.6.1	Overview	238
15.6.2	Data Structure Documentation	239
15.6.3	Macro Definition Documentation	240
15.6.4	Typedef Documentation	240
15.6.5	Function Documentation	240
15.7	I2C FreeRTOS Driver	243
15.7.1	Overview	243
15.7.2	Data Structure Documentation	243
15.7.3	Macro Definition Documentation	244
15.7.4	Function Documentation	244

Contents

Section Number	Title	Page Number
Chapter	SPI: Serial Peripheral Interface Driver	
16.1	Overview	247
16.2	Typical use case	247
16.2.1	SPI master transfer using an interrupt method	247
16.2.2	SPI Send/receive using a DMA method	248
16.3	SPI Driver	250
16.3.1	Overview	250
16.3.2	Data Structure Documentation	255
16.3.3	Macro Definition Documentation	258
16.3.4	Enumeration Type Documentation	258
16.3.5	Function Documentation	261
16.3.6	Variable Documentation	272
16.4	SPI DMA Driver	273
16.4.1	Overview	273
16.4.2	Data Structure Documentation	274
16.4.3	Macro Definition Documentation	275
16.4.4	Typedef Documentation	275
16.4.5	Function Documentation	275
16.5	SPI FreeRTOS driver	280
16.5.1	Overview	280
16.5.2	Data Structure Documentation	280
16.5.3	Macro Definition Documentation	281
16.5.4	Function Documentation	281
Chapter	USART: Universal Asynchronous Receiver/Transmitter Driver	
17.1	Overview	283
17.2	Typical use case	284
17.2.1	USART Send/receive using a polling method	284
17.2.2	USART Send/receive using an interrupt method	284
17.2.3	USART Receive using the ringbuffer feature	285
17.2.4	USART Send/Receive using the DMA method	286
17.3	USART Driver	288
17.3.1	Overview	288
17.3.2	Data Structure Documentation	292
17.3.3	Macro Definition Documentation	294
17.3.4	Typedef Documentation	294
17.3.5	Enumeration Type Documentation	294
17.3.6	Function Documentation	297

Contents

Section Number	Title	Page Number
17.4	USART DMA Driver	310
17.4.1	Overview	310
17.4.2	Data Structure Documentation	311
17.4.3	Macro Definition Documentation	312
17.4.4	Typedef Documentation	312
17.4.5	Function Documentation	312
17.5	USART FreeRTOS Driver	316
17.5.1	Overview	316
17.5.2	Data Structure Documentation	316
17.5.3	Macro Definition Documentation	317
17.5.4	Function Documentation	317
Chapter	FMEAS: Frequency Measure Driver	
18.1	Overview	321
18.2	Frequency Measure Driver operation	321
18.3	Typical use case	321
18.4	Macro Definition Documentation	322
18.4.1	FSL_FMEAS_DRIVER_VERSION	322
18.5	Function Documentation	322
18.5.1	FMEAS_StartMeasure	322
18.5.2	FMEAS_StartMeasureWithScale	322
18.5.3	FMEAS_IsMeasureComplete	322
18.5.4	FMEAS_GetFrequency	323
18.5.5	FMEAS_GetCountWithScale	324
Chapter	GINT: Group GPIO Input Interrupt Driver	
19.1	Overview	325
19.2	Group GPIO Input Interrupt Driver operation	325
19.3	Typical use case	325
19.4	Macro Definition Documentation	326
19.4.1	FSL_GINT_DRIVER_VERSION	326
19.5	Typedef Documentation	326
19.5.1	gint_cb_t	326
19.6	Enumeration Type Documentation	326

Contents

Section Number	Title	Page Number
19.6.1	gint_comb_t	326
19.6.2	gint_trig_t	326
19.7	Function Documentation	327
19.7.1	GINT_Init	327
19.7.2	GINT_SetCtrl	328
19.7.3	GINT_GetCtrl	328
19.7.4	GINT_ConfigPins	329
19.7.5	GINT_GetConfigPins	329
19.7.6	GINT_EnableCallback	330
19.7.7	GINT_DisableCallback	330
19.7.8	GINT_ClrStatus	330
19.7.9	GINT_GetStatus	331
19.7.10	GINT_Deinit	331
Chapter	GPIO: General Purpose I/O	
20.1	Overview	333
20.2	Function groups	333
20.2.1	Initialization and deinitialization	333
20.2.2	Pin manipulation	333
20.2.3	Port manipulation	333
20.2.4	Port masking	333
20.3	Typical use case	333
20.4	Data Structure Documentation	335
20.4.1	struct gpio_pin_config_t	335
20.5	Macro Definition Documentation	335
20.5.1	FSL_GPIO_DRIVER_VERSION	335
20.6	Enumeration Type Documentation	335
20.6.1	gpio_pin_direction_t	335
20.7	Function Documentation	335
20.7.1	GPIO_PortInit	335
20.7.2	GPIO_PinInit	335
20.7.3	GPIO_PinWrite	336
20.7.4	GPIO_PinRead	336
20.7.5	GPIO_PortSet	337
20.7.6	GPIO_PortClear	337
20.7.7	GPIO_PortToggle	337

Contents

Section Number	Title	Page Number
Chapter	INPUTMUX: Input Multiplexing Driver	
21.1	Overview	339
21.2	Input Multiplexing Driver operation	339
21.3	Typical use case	339
21.4	Macro Definition Documentation	343
21.4.1	PINTSEL_PMUX_ID	343
21.4.2	FSL_INPUTMUX_DRIVER_VERSION	343
21.5	Enumeration Type Documentation	343
21.5.1	inputmux_connection_t	343
21.6	Function Documentation	345
21.6.1	INPUTMUX_Init	345
21.6.2	INPUTMUX_AttachSignal	345
21.6.3	INPUTMUX_Deinit	345
Chapter	IOCON: I/O pin configuration	
22.1	Overview	347
22.2	Function groups	347
22.2.1	Pin mux set	347
22.2.2	Pin mux set	347
22.3	Typical use case	347
22.4	Data Structure Documentation	349
22.4.1	struct iocon_group_t	349
22.5	Macro Definition Documentation	349
22.5.1	LPC_IOCON_DRIVER_VERSION	349
22.5.2	IOCON_FUNC0	349
22.6	Function Documentation	350
22.6.1	IOCON_PinMuxSet	350
22.6.2	IOCON_SetPinMuxing	351
22.6.3	IOCON_PullSet	351
22.6.4	IOCON_FuncSet	352
Chapter	IRM: Infra-Red Modulator driver	
23.1	Overview	353

Contents

Section Number	Title	Page Number
23.2	Data Structure Documentation	355
23.2.1	struct cic_irb_config_t	355
23.2.2	struct cic_irb_instance_data_t	355
23.3	Enumeration Type Documentation	355
23.3.1	cic_irb_protocols_t	355
23.3.2	cic_irb_status_t	356
23.3.3	cic_irb_carrier_frequency_t	356
23.3.4	cic_irb_sirc_version_t	356
23.3.5	cic_irb_rcmm_mode_t	356
23.3.6	cic_irb_rcmm_signal_free_time_t	356
23.4	Function Documentation	356
23.4.1	CIC_IRB_GetDefaultConfig	356
23.4.2	CIC_IRB_Init	356
23.4.3	CIC_IRB_Enable	357
23.4.4	CIC_IRB_SendRC5Packet	357
23.4.5	CIC_IRB_SendRC6Packet	358
23.4.6	CIC_IRB_SendSIRCPacket	359
23.4.7	CIC_IRB_SendRCMMPacket	359
23.4.8	CIC_IRB_IsBusy	360
23.4.9	CIC_IRB_Disable	360
23.4.10	CIC_IRB_DeInit	360
23.4.11	CIC_IRB_GetInstanceData	361
23.4.12	CIC_IRB_RC5Initialise	361
23.4.13	CIC_IRB_RC6Initialise	361
23.4.14	CIC_IRB_SIRCInitialise	361
23.4.15	CIC_IRB_RCMMInitialise	362
23.4.16	CIC_IRB_GetInstance	362
23.4.17	CIC_IRB_LoadAndSendFifo	362
23.4.18	CIC_IRB_RCxAppendEnvelopes	363
Chapter	PINT: Pin Interrupt and Pattern Match Driver	
24.1	Overview	365
24.2	Pin Interrupt and Pattern match Driver operation	365
24.2.1	Pin Interrupt use case	365
24.2.2	Pattern match use case	365
24.3	Typedef Documentation	368
24.3.1	pint_cb_t	368
24.4	Enumeration Type Documentation	368
24.4.1	pint_pin_enable_t	368

Contents

Section Number	Title	Page Number
24.4.2	pint_pin_int_t	368
24.4.3	pint_pmatch_input_src_t	368
24.4.4	pint_pmatch_bslice_t	369
24.4.5	pint_pmatch_bslice_cfg_t	369
24.5	Function Documentation	369
24.5.1	PINT_Init	369
24.5.2	PINT_PinInterruptConfig	369
24.5.3	PINT_PinInterruptGetConfig	370
24.5.4	PINT_PinInterruptClrStatus	370
24.5.5	PINT_PinInterruptGetStatus	371
24.5.6	PINT_PinInterruptClrStatusAll	371
24.5.7	PINT_PinInterruptGetStatusAll	371
24.5.8	PINT_PinInterruptClrFallFlag	372
24.5.9	PINT_PinInterruptGetFallFlag	372
24.5.10	PINT_PinInterruptClrFallFlagAll	372
24.5.11	PINT_PinInterruptGetFallFlagAll	373
24.5.12	PINT_PinInterruptClrRiseFlag	373
24.5.13	PINT_PinInterruptGetRiseFlag	374
24.5.14	PINT_PinInterruptClrRiseFlagAll	375
24.5.15	PINT_PinInterruptGetRiseFlagAll	375
24.5.16	PINT_PatternMatchConfig	376
24.5.17	PINT_PatternMatchGetConfig	377
24.5.18	PINT_PatternMatchGetStatus	377
24.5.19	PINT_PatternMatchGetStatusAll	378
24.5.20	PINT_PatternMatchResetDetectLogic	379
24.5.21	PINT_PatternMatchEnable	379
24.5.22	PINT_PatternMatchDisable	379
24.5.23	PINT_PatternMatchEnableRXEV	380
24.5.24	PINT_PatternMatchDisableRXEV	380
24.5.25	PINT_EnableCallback	380
24.5.26	PINT_DisableCallback	381
24.5.27	PINT_Deinit	381
24.5.28	PINT_EnableCallbackByIndex	381
24.5.29	PINT_DisableCallbackByIndex	382
Chapter	Power: Power driver	
25.1	Overview	383
25.2	Function description	383
25.2.1	Power enable and disable	383
25.3	Data Structure Documentation	389
25.3.1	struct pm_bod_cfg_t	389

Contents

Section Number	Title	Page Number
25.3.2	struct pm_power_config_t	389
25.4	Macro Definition Documentation	389
25.4.1	POWER_BOD_LVL_1_75V	389
25.4.2	POWER_BOD_HYST_25MV	390
25.4.3	POWER_ENTER_SLEEP	390
25.5	Enumeration Type Documentation	390
25.5.1	pd_bit_t	390
25.5.2	pm_power_mode_t	390
25.5.3	reset_cause_t	390
25.5.4	pm_ldo_volt_t	391
25.6	Function Documentation	391
25.6.1	POWER_Init	391
25.6.2	POWER_SetTrimDefaultActiveVoltage	391
25.6.3	POWER_BodSetUp	391
25.6.4	POWER_BodActivate	392
25.6.5	POWER_EnablePD	392
25.6.6	POWER_DisablePD	392
25.6.7	POWER_GetIoWakeStatus	392
25.6.8	POWER_EnterSleep	393
25.6.9	POWER_EnterPowerMode	393
25.6.10	POWER_GetResetCause	393
25.6.11	POWER_GetLibVersion	394
25.6.12	POWER_BodVbatGetDefaultConfig	395
25.6.13	POWER_BodVbatConfig	395
25.6.14	POWER_ApplyLdoActiveVoltage	395
Chapter	PWM: Pulse Width Modulator	
26.1	Overview	397
26.2	Typical use case	397
26.2.1	PWM output	397
26.3	Data Structure Documentation	400
26.3.1	struct pwm_config_t	400
26.3.2	struct pwm_setup_t	400
26.4	Macro Definition Documentation	401
26.4.1	FSL_PWM_DRIVER_VERSION	401
26.5	Enumeration Type Documentation	401
26.5.1	pwm_channels_t	401
26.5.2	pwm_polarity_control_t	401

Contents

Section Number	Title	Page Number
26.5.3	pwm_dis_output_level_t	402
26.5.4	pwm_interrupt_enable_t	402
26.5.5	pwm_interrupt_status_t	402
26.6	Function Documentation	402
26.6.1	PWM_GetDefaultConfig	402
26.6.2	PWM_Init	402
26.6.3	PWM_Deinit	403
26.6.4	PWM_SetupPwm	403
26.6.5	PWM_EnableInterrupts	404
26.6.6	PWM_DisableInterrupts	404
26.6.7	PWM_GetEnabledInterrupts	404
26.6.8	PWM_GetStatusFlags	405
26.6.9	PWM_ClearStatusFlags	405
26.6.10	PWM_StartTimer	405
26.6.11	PWM_StopTimer	406
26.6.12	PWM_ReadPeriodValue	406
26.6.13	PWM_ReadCompareValue	406
Chapter	RNG: Random Number generator	
27.1	Overview	409
27.2	Enumeration Type Documentation	409
27.2.1	trng_mode_t	409
27.3	Function Documentation	409
27.3.1	TRNG_GetDefaultConfig	409
27.3.2	TRNG_Init	410
27.3.3	TRNG_Deinit	410
27.3.4	TRNG_GetRandomData	410
Chapter	RTC: Real Time Clock	
28.1	Overview	413
28.2	Function groups	413
28.2.1	Initialization and deinitialization	413
28.2.2	Set & Get Datetime	413
28.2.3	Set & Get Alarm	413
28.2.4	Start & Stop timer	413
28.2.5	Status	414
28.2.6	Interrupt	414
28.2.7	High resolution timer	414

Contents

Section Number	Title	Page Number
28.3	Typical use case	414
28.3.1	RTC tick example	414
28.4	Data Structure Documentation	416
28.4.1	struct rtc_datetime_t	416
28.5	Enumeration Type Documentation	416
28.5.1	rtc_interrupt_enable_t	416
28.5.2	rtc_status_flags_t	416
28.6	Function Documentation	417
28.6.1	RTC_Init	417
28.6.2	RTC_Deinit	417
28.6.3	RTC_SetDatetime	417
28.6.4	RTC_GetDatetime	417
28.6.5	RTC_SetAlarm	418
28.6.6	RTC_GetAlarm	418
28.6.7	RTC_SetWakeupCount	418
28.6.8	RTC_GetWakeupCount	419
28.6.9	RTC_EnableInterrupts	419
28.6.10	RTC_DisableInterrupts	419
28.6.11	RTC_GetEnabledInterrupts	419
28.6.12	RTC_GetStatusFlags	420
28.6.13	RTC_ClearStatusFlags	420
28.6.14	RTC_StartTimer	420
28.6.15	RTC_StopTimer	421
28.6.16	RTC_Reset	421
Chapter	SPIFI: SPIFI flash interface driver	
29.1	Overview	423
29.2	Data Structure Documentation	426
29.2.1	struct spifi_command_t	426
29.2.2	struct spifi_config_t	426
29.2.3	struct spifi_transfer_t	427
29.2.4	struct _spifi_dma_handle	427
29.3	Macro Definition Documentation	428
29.3.1	FSL_SPIFI_DRIVER_VERSION	428
29.3.2	FSL_SPIFI_DMA_DRIVER_VERSION	428
29.4	Enumeration Type Documentation	428
29.4.1	_status_t	428
29.4.2	spifi_interrupt_enable_t	428
29.4.3	spifi_spi_mode_t	428

Contents

Section Number	Title	Page Number
29.4.4	spifi_dual_mode_t	428
29.4.5	spifi_data_direction_t	429
29.4.6	spifi_command_format_t	429
29.4.7	spifi_command_type_t	429
29.4.8	_spifi_status_flags	429
29.5	Function Documentation	430
29.5.1	SPIFI_GetInstance	430
29.5.2	SPIFI_Init	431
29.5.3	SPIFI_GetDefaultConfig	431
29.5.4	SPIFI_Deinit	431
29.5.5	SPIFI_SetCommand	431
29.5.6	SPIFI_SetCommandAddress	432
29.5.7	SPIFI_SetIntermediateData	432
29.5.8	SPIFI_SetCacheLimit	432
29.5.9	SPIFI_ResetCommand	433
29.5.10	SPIFI_SetMemoryCommand	433
29.5.11	SPIFI_EnableInterrupt	433
29.5.12	SPIFI_DisableInterrupt	433
29.5.13	SPIFI_GetStatusFlag	434
29.5.14	SPIFI_EnableDMA	434
29.5.15	SPIFI_GetDataRegisterAddress	434
29.5.16	SPIFI_WriteData	435
29.5.17	SPIFI_WriteBuffer	435
29.5.18	SPIFI_WriteDataByte	435
29.5.19	SPIFI_WriteDataHalfword	435
29.5.20	SPIFI_ReadData	436
29.5.21	SPIFI_ReadDataByte	436
29.5.22	SPIFI_ReadDataHalfword	436
29.5.23	SPIFI_TransferTxCreateHandleDMA	437
29.5.24	SPIFI_TransferRxCreateHandleDMA	438
29.5.25	SPIFI_TransferSendDMA	438
29.5.26	SPIFI_TransferReceiveDMA	438
29.5.27	SPIFI_TransferAbortSendDMA	439
29.5.28	SPIFI_TransferAbortReceiveDMA	439
29.5.29	SPIFI_TransferGetSendCountDMA	439
29.5.30	SPIFI_TransferGetReceiveCountDMA	440
29.6	SPIFI Driver	441
29.6.1	Typical use case	441
29.7	SPIFI DMA Driver	442
29.7.1	Typical use case	442

Contents

Section Number	Title	Page Number
Chapter	WWDT: Windowed Watchdog Timer Driver	
30.1	Overview	443
30.2	Function groups	443
30.2.1	Initialization and deinitialization	443
30.2.2	Status	443
30.2.3	Interrupt	443
30.2.4	Watch dog Refresh	443
30.3	Typical use case	443
30.4	Data Structure Documentation	445
30.4.1	struct wwdt_config_t	445
30.5	Macro Definition Documentation	445
30.5.1	FSL_WWDT_DRIVER_VERSION	445
30.6	Enumeration Type Documentation	445
30.6.1	_wwdt_status_flags_t	445
30.7	Function Documentation	446
30.7.1	WWDT_GetDefaultConfig	446
30.7.2	WWDT_Init	446
30.7.3	WWDT_Deinit	446
30.7.4	WWDT_Enable	447
30.7.5	WWDT_Disable	447
30.7.6	WWDT_GetStatusFlags	447
30.7.7	WWDT_ClearStatusFlags	448
30.7.8	WWDT_SetWarningValue	448
30.7.9	WWDT_SetTimeoutValue	448
30.7.10	WWDT_SetWindowValue	449
30.7.11	WWDT_Refresh	449
Chapter	CMP: Comparator driver	
31.1	Overview	451
31.2	Data Structure Documentation	452
31.2.1	struct cmp_config_t	452
31.3	Macro Definition Documentation	453
31.3.1	FSL_CMP_DRIVER_VERSION	453
31.4	Enumeration Type Documentation	453
31.4.1	_cmp_status	453

Contents

Section Number	Title	Page Number
31.4.2	cmp_interrupt_mask_t	453
31.4.3	cmp_mode_t	453
31.4.4	cmp_input_t	453
31.5	Function Documentation	453
31.5.1	CMP_Init	453
31.5.2	CMP_Deinit	454
31.5.3	CMP_SwapExtInput	454
31.5.4	CMP_EnableLowPowerMode	454
31.5.5	CMP_EnableInnerInput	454
31.5.6	CMP_GetOutput	454
31.5.7	CMP_SetInterruptConfig	454
31.5.8	CMP_GetStatus	455
31.5.9	CMP_ClearStatus	455
31.5.10	CMP_GetInterruptStatus	455
Chapter	FLASHIAP: Flash In Application Programming Driver	
32.1	Overview	457
32.2	GFlash In Application Programming operation	457
32.3	Typical use case	457
32.4	Macro Definition Documentation	459
32.4.1	FSL_FLASHIAP_DRIVER_VERSION	459
32.5	Enumeration Type Documentation	459
32.5.1	_flashiap_status	459
32.5.2	_flashiap_commands	460
32.6	Function Documentation	460
32.6.1	iap_entry	460
32.6.2	FLASHIAP_PrepereSectorForWrite	460
32.6.3	FLASHIAP_CopyRamToFlash	461
32.6.4	FLASHIAP_EraseSector	462
32.6.5	FLASHIAP_ErasePage	463
32.6.6	FLASHIAP_BlankCheckSector	464
32.6.7	FLASHIAP_Compare	464
Chapter	SHA: SHA encryption decryption driver	
33.1	Overview	467
33.2	SHA Driver Initialization and Configuration	467

Contents

Section Number	Title	Page Number
33.3	Comments about API usage in RTOS	467
33.4	SHA Driver Example	467
33.5	Data Structure Documentation	468
33.5.1	struct sha_ctx_t	468
33.6	Macro Definition Documentation	468
33.6.1	FSL_SHA_DRIVER_VERSION	468
33.6.2	SHA_CTX_SIZE	468
33.7	Enumeration Type Documentation	468
33.7.1	sha_algo_t	468
Chapter	Serial Manager	
34.1	Overview	469
34.2	Data Structure Documentation	471
34.2.1	struct serial_manager_config_t	471
34.2.2	struct serial_manager_callback_message_t	471
34.3	Enumeration Type Documentation	471
34.3.1	serial_port_type_t	471
34.3.2	serial_manager_status_t	472
34.4	Function Documentation	472
34.4.1	SerialManager_Init	472
34.4.2	SerialManager_Deinit	473
34.4.3	SerialManager_OpenWriteHandle	474
34.4.4	SerialManager_CloseWriteHandle	475
34.4.5	SerialManager_OpenReadHandle	476
34.4.6	SerialManager_CloseReadHandle	476
34.4.7	SerialManager_WriteBlocking	477
34.4.8	SerialManager_ReadBlocking	478
34.4.9	SerialManager_EnterLowpower	478
34.4.10	SerialManager_ExitLowpower	479
34.5	Serial Port Uart	480
34.5.1	Overview	480
34.5.2	Data Structure Documentation	480
34.5.3	Enumeration Type Documentation	481
34.6	Serial Port USB	482
34.6.1	Overview	482
34.6.2	Data Structure Documentation	483

Contents

Section Number	Title	Page Number
34.6.3	Enumeration Type Documentation	483
34.6.4	USB Device Configuration	484
34.7	Serial Port SWO	486
34.7.1	Overview	486
34.7.2	Data Structure Documentation	486
34.7.3	Enumeration Type Documentation	486
34.8	Serial Port Virtual USB	487
34.8.1	Overview	487
34.8.2	Data Structure Documentation	488
34.8.3	Enumeration Type Documentation	488
Chapter	NTAG: integrated NTAG	
35.1	Overview	491
35.2	Data Structure Documentation	492
35.2.1	struct ntag_config_t	492
35.3	Macro Definition Documentation	492
35.3.1	RFT1503	492
35.4	Typedef Documentation	492
35.4.1	ntag_field_detect_callback_t	492
35.5	Enumeration Type Documentation	492
35.5.1	ntag_state_t	492
35.5.2	ntag_field_detect_t	492
35.6	Function Documentation	492
35.6.1	NTAG_GetDefaultConfig	492
35.6.2	NTAG_Init	493
35.6.3	NTAG_PollFieldDetect	493
35.6.4	NTAG_SetState	493
Chapter	Wtimer	
36.1	Overview	495
36.2	Function Documentation	496
36.2.1	WTIMER_Init	496
36.2.2	WTIMER_DeInit	496
36.2.3	WTIMER_EnableInterrupts	496
36.2.4	WTIMER_GetStatusFlags	496
36.2.5	WTIMER_ClearStatusFlags	496

Contents

Section Number	Title	Page Number
36.2.6	WTIMER_StartTimer	497
36.2.7	WTIMER_StopTimer	497
36.2.8	WTIMER_CalibrateTimer	497
36.2.9	WTIMER_ReadTimer	497
36.2.10	WTIMER_ReadTimerSafe	498
Chapter	ROM_API	
37.1	Overview	501
37.2	Data Structure Documentation	516
37.2.1	struct IMAGE_DATA_T	516
37.2.2	struct ISP_MEM_FUNC_T	516
37.2.3	struct ISP_MEM_INFO_T	516
37.2.4	struct ISP_ENC_STATE_T	517
37.2.5	struct ISP_STATE_T	517
37.2.6	struct LPC_LOWPOWER_T	518
37.2.7	struct LPC_LOWPOWER_LDOVOLTAGE_T	519
37.2.8	struct MPU_Settings_t	519
37.2.9	struct image_directory_entry_t	519
37.2.10	struct psector_header_t	520
37.2.11	struct IMAGE_CERT_T	520
37.3	Macro Definition Documentation	520
37.3.1	FSL_JN_ROMAPI_DRIVER_VERSION	520
37.3.2	ISP_FLAG_HAS_CRC32	520
37.3.3	LOWPOWER_CFG_FLASHPWDNMODE_FLASHPWND	521
37.3.4	LOWPOWER_PMUPWDN_DCDC	521
37.3.5	LOWPOWER_DIGPWDN_FLASH	521
37.3.6	LOWPOWER_WAKEUPSRCINT0_SYSTEM_IRQ	521
37.3.7	LOWPOWER_SLEEPPOSTPONE_FORCED	521
37.3.8	LOWPOWER_TIMERCFG_TIMER_WAKEUPTIMER0	521
37.3.9	LOWPOWER_TIMERCFG_2ND_TIMER_WAKEUPTIMER0	521
37.3.10	PSECTOR_PAGE_WORDS	521
37.4	Typedef Documentation	522
37.4.1	IMAGE_VERIFY_T	522
37.5	Enumeration Type Documentation	522
37.5.1	AES_MODE_T	522
37.5.2	AES_KEY_SIZE_T	522
37.5.3	teFlashProgCommand	522
37.5.4	ISP_STATUS_T	523
37.5.5	ISP_MEMORY_TYPE_E	524
37.5.6	MpuRegion_t	524

Contents

Section Number	Title	Page Number
37.5.7	psector_partition_id_t	525
37.5.8	psector_page_state_t	525
37.5.9	psector_write_status_t	525
37.5.10	AuthMode_t	526
37.6	Function Documentation	526
37.6.1	aesInit	526
37.6.2	aesWriteByte	526
37.6.3	aesWrite	526
37.6.4	aesRead	527
37.6.5	aesWriteBlock	527
37.6.6	aesReadBlock	527
37.6.7	aesMode	528
37.6.8	aesAbort	529
37.6.9	aesLoadCounter	529
37.6.10	aesLoadKeyFromSW	529
37.6.11	aesLoadIV	530
37.6.12	aesProcess	530
37.6.13	aesWriteYInputGf128	530
37.6.14	aesReadGf128Hash	531
37.6.15	aesReadGcmTag	531
37.6.16	aesGetDriverVersion	532
37.6.17	aesIsSupported	532
37.6.18	BOOT_RemapAddress	532
37.6.19	boot_Verify_eScoreImageList	532
37.6.20	BOOT_FindImage	533
37.6.21	BOOT_GetStartPowerMode	533
37.6.22	BOOT_SetResumeStackPointer	533
37.6.23	ROM_GetFlash	534
37.6.24	ROM_GetSRAM0	534
37.6.25	ROM_GetSRAM1	534
37.6.26	ISP_Entry	535
37.6.27	Chip_LOWPOWER_SetUpLowPowerModeWakeUpTimer	535
37.6.28	Chip_LOWPOWER_SetSystemFrequency	535
37.6.29	Chip_LOWPOWER_SetMemoryLowPowerMode	536
37.6.30	Chip_LOWPOWER_GetSystemVoltages	536
37.6.31	Chip_LOWPOWER_SetSystemVoltages	536
37.6.32	Chip_LOWPOWER_SetLowPowerMode	537
37.6.33	Chip_LOWPOWER_ChipSoftwareReset	537
37.6.34	Chip_LOWPOWER_ArmSoftwareReset	537
37.6.35	MPU_pSectorGrantAccessRights	538
37.6.36	MPU_pSectorWithdrawAccessRights	538
37.6.37	MPU_GetCurrentSettings	538
37.6.38	MPU_AllocateRegionDesc	539
37.6.39	pmc_reset_get_cause	539

Contents

Section Number	Title	Page Number
37.6.40	pmc_reset_clear_cause	540
37.6.41	psector_WriteUpdatePage	541
37.6.42	psector_EraseUpdate	541
37.6.43	psector_ReadData	541
37.6.44	psector_CalculateChecksum	542
37.6.45	psector_Read_CustomerId	542
37.6.46	psector_Read_RomPatchInfo	542
37.6.47	psector_Read_ImgAuthLevel	543
37.6.48	psector_Read_AppSearchGranularity	543
37.6.49	psector_Read_QspiAppSearchGranularity	543
37.6.50	psector_Read_DeviceId	544
37.6.51	psector_Read_UnlockKey	544
37.6.52	psector_Read_ISP_protocol_key	544
37.6.53	psector_ReadIeee802_15_4_MacId1	545
37.6.54	psector_ReadIeee802_15_4_MacId2	545
37.6.55	psector_Read_MinDeviceId	545
37.6.56	psector_Read_MaxDeviceId	546
37.6.57	psector_Read_MinVersion	546
37.6.58	psector_SetEscoreImageData	546
37.6.59	psector_ReadEscoreImageData	547
37.6.60	psector_Read_ImagePubKey	547
37.6.61	secure_VerifySignature	547
37.6.62	secure_VerifyBlock	548
37.6.63	secure_VerifyCertificate	548
37.6.64	secure_VerifyImage	549
37.7	Variable Documentation	549
37.7.1	access	549
37.7.2	mode	549
37.7.3	MinVersion	550
37.7.4	MinVersion	550
37.7.5	rom_patch_region_addr	550
37.7.6	rom_patch_region_addr	550
37.7.7	application_flash_sz	550
37.7.8	application_flash_sz	550
37.7.9	ram1_bank_sz	550
37.7.10	ram1_bank_sz	551
37.7.11	app_search_granularity	551
37.7.12	app_search_granularity	551
37.7.13	unlock_key	551
37.7.14	unlock_key	551
37.8	Sha_algorithm_level_api	552
37.8.1	Overview	552
37.8.2	Function Documentation	552

Contents

Section Number	Title	Page Number
Chapter	GenericList	
38.1	Overview	555
38.2	Data Structure Documentation	556
38.2.1	struct list_label_t	556
38.2.2	struct list_element_t	556
38.3	Enumeration Type Documentation	556
38.3.1	list_status_t	556
38.4	Function Documentation	557
38.4.1	LIST_Init	557
38.4.2	LIST_GetList	557
38.4.3	LIST_AddHead	557
38.4.4	LIST_AddTail	557
38.4.5	LIST_RemoveHead	558
38.4.6	LIST_GetHead	558
38.4.7	LIST_GetNext	558
38.4.8	LIST_GetPrev	559
38.4.9	LIST_RemoveElement	559
38.4.10	LIST_AddPrevElement	559
38.4.11	LIST_GetSize	560
38.4.12	LIST_GetAvailableSize	560
Chapter	UART_Adapter	
39.1	Overview	561
39.2	Data Structure Documentation	562
39.2.1	struct hal_uart_config_t	562
39.2.2	struct hal_uart_transfer_t	562
39.3	Macro Definition Documentation	563
39.3.1	HAL_UART_TRANSFER_MODE	563
39.4	Typedef Documentation	563
39.4.1	hal_uart_transfer_callback_t	563
39.5	Enumeration Type Documentation	563
39.5.1	hal_uart_status_t	563
39.5.2	hal_uart_parity_mode_t	563
39.5.3	hal_uart_stop_bit_count_t	564
39.6	Function Documentation	564
39.6.1	HAL_UartInit	564

Section Number	<h1>Contents</h1> Title	Page Number
39.6.2	HAL_UartDeinit	564
39.6.3	HAL_UartReceiveBlocking	565
39.6.4	HAL_UartSendBlocking	565

Chapter 1

Introduction

The MCUXpresso Software Development Kit (MCUXpresso SDK) is a collection of software enablement for NXP Microcontrollers that includes peripheral drivers, multicore support and integrated RTOS support for FreeRTOS™. In addition to the base enablement, the MCUXpresso SDK is augmented with demo applications, driver example projects, and API documentation to help users quickly leverage the support provided by MCUXpresso SDK. The [MCUXpresso SDK Web Builder](#) is available to provide access to all MCUXpresso SDK packages. See the *MCUXpresso Software Development Kit (SDK) Release Notes* (document MCUXSDKRN) in the Supported Devices section at [MCUXpresso-SDK: Software Development Kit for MCUXpresso](#) for details.

The MCUXpresso SDK is built with the following runtime software components:

- Arm® and DSP standard libraries, and CMSIS-compliant device header files which provide direct access to the peripheral registers.
- Peripheral drivers that provide stateless, high-performance, ease-of-use APIs. Communication drivers provide higher-level transactional APIs for a higher-performance option.
- RTOS wrapper driver built on top of MCUXpresso SDK peripheral drivers and leverage native RTOS services to better comply to the RTOS cases.
- Real time operation systems (RTOS) for FreeRTOS OS.
- Stacks and middleware in source or object formats including:
 - CMSIS-DSP, a suite of common signal processing functions.
 - The MCUXpresso SDK comes complete with software examples demonstrating the usage of the peripheral drivers, RTOS wrapper drivers, middleware, and RTOSes.

All demo applications and driver examples are provided with projects for the following toolchains:

- IAR Embedded Workbench
- GNU Arm Embedded Toolchain

The peripheral drivers and RTOS driver wrappers can be used across multiple devices within the product family without modification. The configuration items for each driver are encapsulated into C language data structures. Device-specific configuration information is provided as part of the MCUXpresso SDK and need not be modified by the user. If necessary, the user is able to modify the peripheral driver and RTOS wrapper driver configuration during runtime. The driver examples demonstrate how to configure the drivers by passing the proper configuration data to the APIs. The folder structure is organized to reduce the total number of includes required to compile a project.

The rest of this document describes the API references in detail for the peripheral drivers and RTOS wrapper drivers. For the latest version of this and other MCUXpresso SDK documents, see the mcuxpresso.nxp.com/apidoc/.

Deliverable	Location
Demo Applications	<install_dir>/boards/<board_name>/demo_ - apps
Driver Examples	<install_dir>/boards/<board_name>/driver_ - examples
Documentation	<install_dir>/docs
Middleware	<install_dir>/middleware
Drivers	<install_dir>/<device_name>/drivers/
CMSIS Standard Arm Cortex-M Headers, math and DSP Libraries	<install_dir>/CMSIS
Device Startup and Linker	<install_dir>/<device_name>/<toolchain>/
MCUXpresso SDK Utilities	<install_dir>/devices/<device_name>/utilities
RTOS Kernel Code	<install_dir>/rtos

Table 2: MCUXpresso SDK Folder Structure

Chapter 2

Driver errors status

- `kStatus_DMA_Busy` = 5000
- `kStatus_DMIC_Busy` = 5800
- `kStatus_DMIC_Idle` = 5801
- `kStatus_DMIC_OverRunError` = 5802
- `kStatus_DMIC_UnderRunError` = 5803
- `kStatus_I2C_Busy` = 2600
- `kStatus_I2C_Idle` = 2601
- `kStatus_I2C_Nak` = 2602
- `kStatus_I2C_InvalidParameter` = 2603
- `kStatus_I2C_BitError` = 2604
- `kStatus_I2C_ArbitrationLost` = 2605
- `kStatus_I2C_NoTransferInProgress` = 2606
- `kStatus_I2C_DmaRequestFail` = 2607
- `#kStatus_I2C_StartStopError` = 2608
- `#kStatus_I2C_UnexpectedState` = 2609
- `kStatus_I2C_Timeout` = 2610
- `kStatus_SPI_Busy` = 1400
- `kStatus_SPI_Idle` = 1401
- `kStatus_SPI_Error` = 1402
- `kStatus_USART_TxBusy` = 5700
- `kStatus_USART_RxBusy` = 5701
- `kStatus_USART_TxIdle` = 5702
- `kStatus_USART_RxIdle` = 5703
- `kStatus_USART_TxError` = 5707
- `kStatus_USART_RxError` = 5709
- `kStatus_USART_RxRingBufferOverrun` = 5708
- `kStatus_USART_NoiseError` = 5710
- `kStatus_USART_FramingError` = 5711
- `kStatus_USART_ParityError` = 5712
- `kStatus_USART_BaudrateNotSupport` = 5713
- `kStatus_SPIFI_Busy` = 5900
- `kStatus_SPIFI_Idle` = 5901
- `kStatus_SPIFI_Error` = 5902
- `kStatus_FLASHIAP_Success` = 0
- `kStatus_FLASHIAP_InvalidCommand` = 2501
- `kStatus_FLASHIAP_SrcAddrError` = 2502
- `kStatus_FLASHIAP_DstAddrError` = 2503
- `kStatus_FLASHIAP_SrcAddrNotMapped` = 2504

- `kStatus_FLASHIAP_DstAddrNotMapped` = 2505
- `kStatus_FLASHIAP_CountError` = 2506
- `kStatus_FLASHIAP_InvalidSector` = 2507
- `kStatus_FLASHIAP_SectorNotblank` = 2508
- `kStatus_FLASHIAP_NotPrepared` = 2509
- `kStatus_FLASHIAP_CompareError` = 2510
- `kStatus_FLASHIAP_Busy` = 2511
- `kStatus_FLASHIAP_ParamError` = 2512
- `kStatus_FLASHIAP_AddrError` = 2513
- `kStatus_FLASHIAP_AddrNotMapped` = 2514
- `kStatus_FLASHIAP_NoPower` = 2514
- `kStatus_FLASHIAP_NoClock` = 2527

Chapter 3

Trademarks

Information in this document is provided solely to enable system and software implementers to use NXP products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits based on the information in this document. NXP reserves the right to make changes without further notice to any products herein.

How to Reach Us:

Home Page: nxp.com

Web Support: nxp.com/support

NXP makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does NXP assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. “Typical” parameters that may be provided in NXP data sheets and/or specifications can and do vary in different applications, and actual performance may vary over time. All operating parameters, including “typicals,” must be validated for each customer application by customer’s technical experts. NXP does not convey any license under its patent rights nor the rights of others. NXP sells products pursuant to standard terms and conditions of sale, which can be found at the following address: <http://www.nxp.com/SalesTermsandConditions>.

NXP, the NXP logo, NXP SECURE CONNECTIONS FOR A SMARTER WORLD, COOLFLUX, EMBRACE, GREENCHIP, HITAG, I2C BUS, ICODE, JCOP, LIFE VIBES, MIFARE, MIFARE CLASSIC, MIFARE DESFire, MIFARE PLUS, MIFARE FLEX, MANTIS, MIFARE ULTRALIGHT, MIFARE4MOBILE, MIGLO, NTAG, ROADLINK, SMARTLX, SMARTMX, STARPLUG, TOPFET, TRENCHMOS, UCODE, Freescale, the Freescale logo, AltiVec, C-5, CodeTEST, CodeWarrior, ColdFire, ColdFire+, C-Ware, the Energy Efficient Solutions logo, Kinetis, Layerscape, MagniV, mobileGT, PEG, PowerQUICC, Processor Expert, QorIQ, QorIQ Qonverge, Ready Play, SafeAssure, the SafeAssure logo, StarCore, Symphony, VortiQa, Vybrid, Airfast, BeeKit, BeeStack, CoreNet, Flexis, MXC, Platform in a Package, QUICC Engine, SMARTMOS, Tower, TurboLink, and UMEMS are trademarks of NXP B.V. All other product or service names are the property of their respective owners. AMBA, Arm, Arm7, Arm7TDMI, Arm9, Arm11, Artisan, big.LITTLE, Cordio, CoreLink, CoreSight, Cortex, DesignStart, DynamIQ, Jazelle, Keil, Mali, Mbed, Mbed Enabled, NEON, POP, RealView, SecurCore, Socrates, Thumb, TrustZone, ULINK, ULINK2, ULINK-ME, ULINK-PLUS, ULINKpro, Vision, Versatile are trademarks or registered trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere. The related technology may be protected by any or all of patents, copyrights, designs and trade secrets. All rights reserved. Oracle and Java are registered trademarks of Oracle and/or its affiliates. The Power Architecture and Power.org word marks and the Power and Power.org logos and related marks are trademarks and service marks licensed by Power.org.

© 2019 NXP B.V.



Chapter 4 Architectural Overview

This chapter provides the architectural overview for the MCUXpresso Software Development Kit (MCUXpresso SDK). It describes each layer within the architecture and its associated components.

Overview

The MCUXpresso SDK architecture consists of five key components listed below.

1. The Arm Cortex Microcontroller Software Interface Standard (CMSIS) CORE compliance device-specific header files, SOC Header, and CMSIS math/DSP libraries.
2. Peripheral Drivers
3. Real-time Operating Systems (RTOS)
4. Stacks and Middleware that integrate with the MCUXpresso SDK
5. Demo Applications based on the MCUXpresso SDK

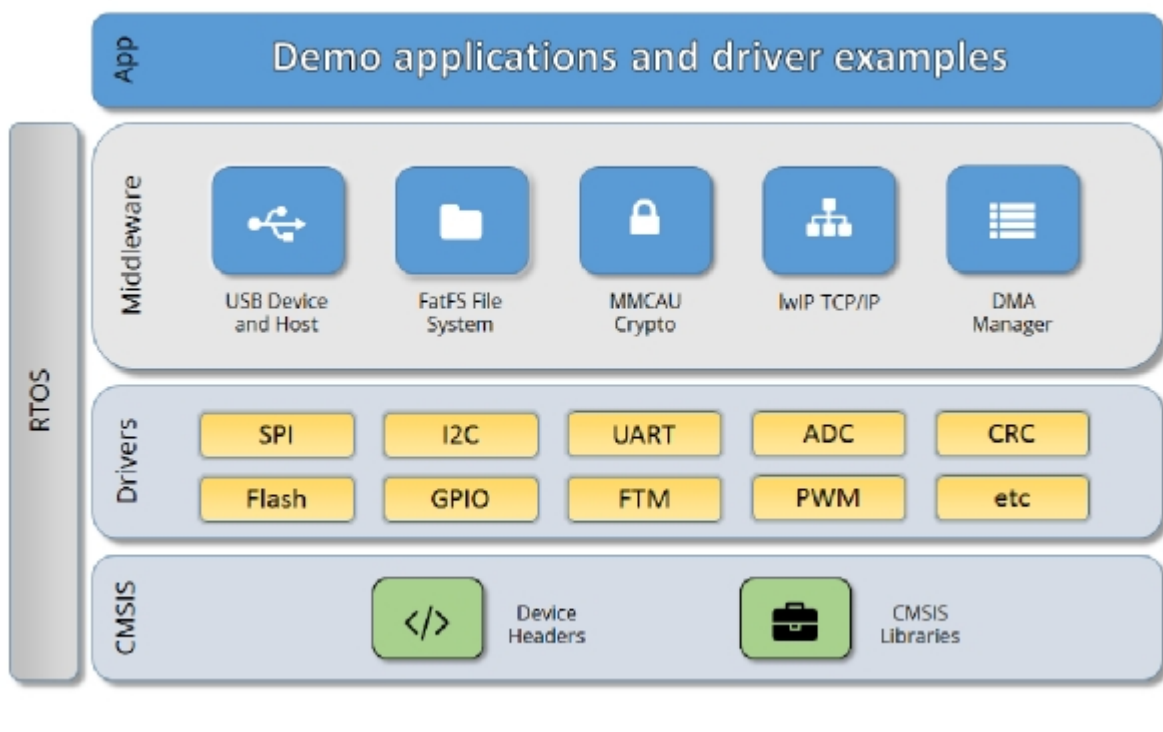


Figure 1: MCUXpresso SDK Block Diagram

MCU header files

Each supported MCU device in the MCUXpresso SDK has an overall System-on Chip (SoC) memory-

mapped header file. This header file contains the memory map and register base address for each peripheral and the IRQ vector table with associated vector numbers. The overall SoC header file provides access to the peripheral registers through pointers and predefined bit masks. In addition to the overall SoC memory-mapped header file, the MCUXpresso SDK includes a feature header file for each device. The feature header file allows NXP to deliver a single software driver for a given peripheral. The feature file ensures that the driver is properly compiled for the target SOC.

CMSIS Support

Along with the SoC header files and peripheral extension header files, the MCUXpresso SDK also includes common CMSIS header files for the Arm Cortex-M core and the math and DSP libraries from the latest CMSIS release. The CMSIS DSP library source code is also included for reference.

MCUXpresso SDK Peripheral Drivers

The MCUXpresso SDK peripheral drivers mainly consist of low-level functional APIs for the MCU product family on-chip peripherals and also of high-level transactional APIs for some bus drivers/DM-A driver/eDMA driver to quickly enable the peripherals and perform transfers.

All MCUXpresso SDK peripheral drivers only depend on the CMSIS headers, device feature files, `fsl_common.h`, and `fsl_clock.h` files so that users can easily pull selected drivers and their dependencies into projects. With the exception of the clock/power-relevant peripherals, each peripheral has its own driver. Peripheral drivers handle the peripheral clock gating/ungating inside the drivers during initialization and deinitialization respectively.

Low-level functional APIs provide common peripheral functionality, abstracting the hardware peripheral register accesses into a set of stateless basic functional operations. These APIs primarily focus on the control, configuration, and function of basic peripheral operations. The APIs hide the register access details and various MCU peripheral instantiation differences so that the application can be abstracted from the low-level hardware details. The API prototypes are intentionally similar to help ensure easy portability across supported MCUXpresso SDK devices.

Transactional APIs provide a quick method for customers to utilize higher-level functionality of the peripherals. The transactional APIs utilize interrupts and perform asynchronous operations without user intervention. Transactional APIs operate on high-level logic that requires data storage for internal operation context handling. However, the Peripheral Drivers do not allocate this memory space. Rather, the user passes in the memory to the driver for internal driver operation. Transactional APIs ensure the NVIC is enabled properly inside the drivers. The transactional APIs do not meet all customer needs, but provide a baseline for development of custom user APIs.

Note that the transactional drivers never disable an NVIC after use. This is due to the shared nature of interrupt vectors on devices. It is up to the user to ensure that NVIC interrupts are properly disabled after usage is complete.

Interrupt handling for transactional APIs

A double weak mechanism is introduced for drivers with transactional API. The double weak indicates two levels of weak vector entries. See the examples below:

```
PUBWEAK SPI0_IRQHandler
PUBWEAK SPI0_DriverIRQHandler
SPI0_IRQHandler
```



```
LDR    R0, =SPI0_DriverIRQHandler
BX     R0
```

The first level of the weak implementation are the functions defined in the vector table. In the devices/(<DEVICE_NAME>)/(<TOOLCHAIN>)/startup_<DEVICE_NAME>.s/.S file, the implementation of the first layer weak function calls the second layer of weak function. The implementation of the second layer weak function (ex. SPI0_DriverIRQHandler) jumps to itself (B). The MCUXpresso SDK drivers with transactional APIs provide the reimplement of the second layer function inside of the peripheral driver. If the MCUXpresso SDK drivers with transactional APIs are linked into the image, the SPI0_DriverIRQHandler is replaced with the function implemented in the MCUXpresso SDK SPI driver.

The reason for implementing the double weak functions is to provide a better user experience when using the transactional APIs. For drivers with a transactional function, call the transactional APIs and the drivers complete the interrupt-driven flow. Users are not required to redefine the vector entries out of the box. At the same time, if users are not satisfied by the second layer weak function implemented in the MCUXpresso SDK drivers, users can redefine the first layer weak function and implement their own interrupt handler functions to suit their implementation.

The limitation of the double weak mechanism is that it cannot be used for peripherals that share the same vector entry. For this use case, redefine the first layer weak function to enable the desired peripheral interrupt functionality. For example, if the MCU's UART0 and UART1 share the same vector entry, redefine the UART0_UART1_IRQHandler according to the use case requirements.

Feature Header Files

The peripheral drivers are designed to be reusable regardless of the peripheral functional differences from one MCU device to another. An overall Peripheral Feature Header File is provided for the MCUXpresso SDK-supported MCU device to define the features or configuration differences for each sub-family device.

Application

See the *Getting Started with MCUXpresso SDK* document (MCUXSDKGSUG).



Chapter 5

ADC: 12-bit SAR Analog-to-Digital Converter Driver

5.1 Overview

The MCUXpresso SDK provides a peripheral driver for the 12-bit Successive Approximation (SAR) Analog-to-Digital Converter (ADC) module of MCUXpresso SDK devices.

5.2 Typical use case

5.2.1 Polling Configuration

Refer to the driver examples codes located at `<SDK_ROOT>/boards/<BOARD>/driver_examples/fsl_adc`

5.2.2 Interrupt Configuration

Refer to the driver examples codes located at `<SDK_ROOT>/boards/<BOARD>/driver_examples/fsl_adc`

Files

- file [fsl_adc.h](#)

Data Structures

- struct [adc_config_t](#)
Define structure for configuring the block. [More...](#)
- struct [adc_conv_seq_config_t](#)
Define structure for configuring conversion sequence. [More...](#)
- struct [adc_result_info_t](#)
Define structure of keeping conversion result information. [More...](#)

Enumerations

- enum `_adc_status_flags` {
 - `kADC_ThresholdCompareFlagOnChn0` = 1U << 0U,
 - `kADC_ThresholdCompareFlagOnChn1` = 1U << 1U,
 - `kADC_ThresholdCompareFlagOnChn2` = 1U << 2U,
 - `kADC_ThresholdCompareFlagOnChn3` = 1U << 3U,
 - `kADC_ThresholdCompareFlagOnChn4` = 1U << 4U,
 - `kADC_ThresholdCompareFlagOnChn5` = 1U << 5U,
 - `kADC_ThresholdCompareFlagOnChn6` = 1U << 6U,
 - `kADC_ThresholdCompareFlagOnChn7` = 1U << 7U,
 - `kADC_ThresholdCompareFlagOnChn8` = 1U << 8U,
 - `kADC_ThresholdCompareFlagOnChn9` = 1U << 9U,
 - `kADC_ThresholdCompareFlagOnChn10` = 1U << 10U,
 - `kADC_ThresholdCompareFlagOnChn11` = 1U << 11U,
 - `kADC_OverrunFlagForChn0`,
 - `kADC_OverrunFlagForChn1`,
 - `kADC_OverrunFlagForChn2`,
 - `kADC_OverrunFlagForChn3`,
 - `kADC_OverrunFlagForChn4`,
 - `kADC_OverrunFlagForChn5`,
 - `kADC_OverrunFlagForChn6`,
 - `kADC_OverrunFlagForChn7`,
 - `kADC_OverrunFlagForChn8`,
 - `kADC_OverrunFlagForChn9`,
 - `kADC_OverrunFlagForChn10`,
 - `kADC_OverrunFlagForChn11`,
 - `kADC_GlobalOverrunFlagForSeqA` = 1U << 24U,
 - `kADC_GlobalOverrunFlagForSeqB` = 1U << 25U,
 - `kADC_ConvSeqAInterruptFlag` = 1U << 28U,
 - `kADC_ConvSeqBInterruptFlag` = 1U << 29U,
 - `kADC_ThresholdCompareInterruptFlag` = 1U << 30U,
 - `kADC_OverrunInterruptFlag` = (int)(1U << 31U) }

Flags.
- enum `_adc_interrupt_enable` {
 - `kADC_ConvSeqAInterruptEnable` = ADC_INTEN_SEQA_INTEN_MASK,
 - `kADC_ConvSeqBInterruptEnable` = ADC_INTEN_SEQB_INTEN_MASK,
 - `kADC_OverrunInterruptEnable` = ADC_INTEN_OVR_INTEN_MASK }

Interrupts.
- enum `adc_clock_mode_t` {
 - `kADC_ClockSynchronousMode`,
 - `kADC_ClockAsynchronousMode` = 1U }

Define selection of clock mode.
- enum `adc_resolution_t` {

- ```

kADC_Resolution6bit = 3U,
kADC_Resolution8bit = 2U,
kADC_Resolution10bit = 1U,
kADC_Resolution12bit = 0U }

```
- Define selection of resolution.*
- enum `adc_trigger_polarity_t` {  
`kADC_TriggerPolarityNegativeEdge` = 0U,  
`kADC_TriggerPolarityPositiveEdge` = 1U }
- Define selection of polarity of selected input trigger for conversion sequence.*
- enum `adc_priority_t` {  
`kADC_PriorityLow` = 0U,  
`kADC_PriorityHigh` = 1U }
- Define selection of conversion sequence's priority.*
- enum `adc_seq_interrupt_mode_t` {  
`kADC_InterruptForEachConversion` = 0U,  
`kADC_InterruptForEachSequence` = 1U }
- Define selection of conversion sequence's interrupt.*
- enum `adc_threshold_compare_status_t` {  
`kADC_ThresholdCompareInRange` = 0U,  
`kADC_ThresholdCompareBelowRange` = 1U,  
`kADC_ThresholdCompareAboveRange` = 2U }
- Define status of threshold compare result.*
- enum `adc_threshold_crossing_status_t` {  
`kADC_ThresholdCrossingNoDetected` = 0U,  
`kADC_ThresholdCrossingDownward` = 2U,  
`kADC_ThresholdCrossingUpward` = 3U }
- Define status of threshold crossing detection result.*
- enum `adc_threshold_interrupt_mode_t` {  
`kADC_ThresholdInterruptDisabled` = 0U,  
`kADC_ThresholdInterruptOnOutside` = 1U,  
`kADC_ThresholdInterruptOnCrossing` = 2U }
- Define interrupt mode for threshold compare event.*
- enum `adc_inforestult_t` {  
`kADC_Resolution12bitInfoResultShift` = 0U,  
`kADC_Resolution10bitInfoResultShift` = 2U,  
`kADC_Resolution8bitInfoResultShift` = 4U,  
`kADC_Resolution6bitInfoResultShift` = 6U }
- Define the info result mode of different resolution.*
- enum `adc_tempsensor_common_mode_t` {  
`kADC_HighNegativeOffsetAdded` = 0x0U,  
`kADC_IntermediateNegativeOffsetAdded`,  
`kADC_NoOffsetAdded` = 0x8U,  
`kADC_LowPositiveOffsetAdded` = 0xcU }
- Define common modes for Temperature sensor.*
- enum `adc_second_control_t` {

## Typical use case

```
kADC_Impedance621Ohm = 0x1U << 9U,
kADC_Impedance55kOhm,
kADC_Impedance87kOhm = 0x1fU << 9U,
kADC_NormalFunctionalMode = 0x0U << 14U,
kADC_MultiplexeTestMode = 0x1U << 14U,
kADC_ADCInUnityGainMode = 0x2U << 14U }
```

*Define source impedance modes for GPADC control.*

## Driver version

- #define `FSL_ADC_DRIVER_VERSION` (`MAKE_VERSION(2, 3, 2)`)  
*ADC driver version 2.3.1.*

## Initialization and Deinitialization

- void `ADC_Init` (`ADC_Type *base`, const `adc_config_t *config`)  
*Initialize the ADC module.*
- void `ADC_Deinit` (`ADC_Type *base`)  
*Deinitialize the ADC module.*
- void `ADC_GetDefaultConfig` (`adc_config_t *config`)  
*Gets an available pre-defined settings for initial configuration.*
- void `ADC_EnableTemperatureSensor` (`ADC_Type *base`, bool enable)  
*Enable the internal temperature sensor measurement.*

## Control conversion sequence A.

- static void `ADC_EnableConvSeqA` (`ADC_Type *base`, bool enable)  
*Enable the conversion sequence A.*
- void `ADC_SetConvSeqAConfig` (`ADC_Type *base`, const `adc_conv_seq_config_t *config`)  
*Configure the conversion sequence A.*
- static void `ADC_DoSoftwareTriggerConvSeqA` (`ADC_Type *base`)  
*Do trigger the sequence's conversion by software.*
- static void `ADC_EnableConvSeqABurstMode` (`ADC_Type *base`, bool enable)  
*Enable the burst conversion of sequence A.*
- static void `ADC_SetConvSeqAHighPriority` (`ADC_Type *base`)  
*Set the high priority for conversion sequence A.*

## Control conversion sequence B.

- static void `ADC_EnableConvSeqB` (`ADC_Type *base`, bool enable)  
*Enable the conversion sequence B.*
- void `ADC_SetConvSeqBConfig` (`ADC_Type *base`, const `adc_conv_seq_config_t *config`)  
*Configure the conversion sequence B.*
- static void `ADC_DoSoftwareTriggerConvSeqB` (`ADC_Type *base`)  
*Do trigger the sequence's conversion by software.*
- static void `ADC_EnableConvSeqBBurstMode` (`ADC_Type *base`, bool enable)  
*Enable the burst conversion of sequence B.*
- static void `ADC_SetConvSeqBHighPriority` (`ADC_Type *base`)  
*Set the high priority for conversion sequence B.*

**Data result.**

- bool [ADC\\_GetConvSeqAGlobalConversionResult](#) (ADC\_Type \*base, [adc\\_result\\_info\\_t](#) \*info)  
*Get the global ADC conversion information of sequence A.*
- bool [ADC\\_GetConvSeqBGlobalConversionResult](#) (ADC\_Type \*base, [adc\\_result\\_info\\_t](#) \*info)  
*Get the global ADC conversion information of sequence B.*
- bool [ADC\\_GetChannelConversionResult](#) (ADC\_Type \*base, uint32\_t channel, [adc\\_result\\_info\\_t](#) \*info)  
*Get the channel's ADC conversion completed under each conversion sequence.*

**Threshold function.**

- static void [ADC\\_SetThresholdPair0](#) (ADC\_Type \*base, uint32\_t lowValue, uint32\_t highValue)  
*Set the threshold pair 0 with low and high value.*
- static void [ADC\\_SetThresholdPair1](#) (ADC\_Type \*base, uint32\_t lowValue, uint32\_t highValue)  
*Set the threshold pair 1 with low and high value.*
- static void [ADC\\_SetChannelWithThresholdPair0](#) (ADC\_Type \*base, uint32\_t channelMask)  
*Set given channels to apply the threshold pair 0.*
- static void [ADC\\_SetChannelWithThresholdPair1](#) (ADC\_Type \*base, uint32\_t channelMask)  
*Set given channels to apply the threshold pair 1.*

**Interrupts.**

- static void [ADC\\_EnableInterrupts](#) (ADC\_Type \*base, uint32\_t mask)  
*Enable interrupts for conversion sequences.*
- static void [ADC\\_DisableInterrupts](#) (ADC\_Type \*base, uint32\_t mask)  
*Disable interrupts for conversion sequence.*
- static void [ADC\\_EnableShresholdCompareInterrupt](#) (ADC\_Type \*base, uint32\_t channel, [adc\\_threshold\\_interrupt\\_mode\\_t](#) mode)  
*Enable the interrupt of threshold compare event for each channel.*
- static void [ADC\\_EnableThresholdCompareInterrupt](#) (ADC\_Type \*base, uint32\_t channel, [adc\\_threshold\\_interrupt\\_mode\\_t](#) mode)  
*Enable the interrupt of threshold compare event for each channel.*

**Status.**

- static uint32\_t [ADC\\_GetStatusFlags](#) (ADC\_Type \*base)  
*Get status flags of ADC module.*
- static void [ADC\\_ClearStatusFlags](#) (ADC\_Type \*base, uint32\_t mask)  
*Clear status flags of ADC module.*

**5.3 Data Structure Documentation****5.3.1 struct [adc\\_config\\_t](#)****Data Fields**

- [adc\\_clock\\_mode\\_t](#) clockMode  
*Select the clock mode for ADC converter.*

## Data Structure Documentation

- uint32\_t [clockDividerNumber](#)  
*This field is only available when using kADC\_ClockSynchronousMode for "clockMode" field.*
- [adc\\_resolution\\_t](#) [resolution](#)  
*Select the conversion bits.*
- uint32\_t [sampleTimeNumber](#)  
*By default, with value as "0U", the sample period would be 2.5 ADC clocks.*

### 5.3.1.0.0.1 Field Documentation

#### 5.3.1.0.0.1.1 [adc\\_clock\\_mode\\_t](#) [adc\\_config\\_t::clockMode](#)

#### 5.3.1.0.0.1.2 [uint32\\_t](#) [adc\\_config\\_t::clockDividerNumber](#)

The divider would be plused by 1 based on the value in this field. The available range is in 8 bits.

#### 5.3.1.0.0.1.3 [adc\\_resolution\\_t](#) [adc\\_config\\_t::resolution](#)

#### 5.3.1.0.0.1.4 [uint32\\_t](#) [adc\\_config\\_t::sampleTimeNumber](#)

Then, to plus the "sampleTimeNumber" value here. The available value range is in 3 bits.

## 5.3.2 struct [adc\\_conv\\_seq\\_config\\_t](#)

### Data Fields

- [uint32\\_t](#) [channelMask](#)  
*Selects which one or more of the ADC channels will be sampled and conversion sequence is launched.*
- [uint32\\_t](#) [triggerMask](#)  
*Selects which one or more of the available hardware trigger sources will conversion sequence to be initiated.*
- [adc\\_trigger\\_polarity\\_t](#) [triggerPolarity](#)  
*Select the trigger to launch conversion sequence.*
- bool [enableSyncBypass](#)  
*To enable this feature allows the hardware trigger input to bypass synchronous flip-flop stages and therefore shorten the time between the trigger input signal and the start of a conversion.*
- bool [enableSingleStep](#)  
*When enabling this feature, a trigger will launch a single conversion on channel in the sequence instead of the default response of launching an entire sequence of conversions.*
- [adc\\_seq\\_interrupt\\_mode\\_t](#) [interruptMode](#)  
*Select the interrupt/DMA trigger mode.*

### 5.3.2.0.0.2 Field Documentation

#### 5.3.2.0.0.2.1 [uint32\\_t](#) [adc\\_conv\\_seq\\_config\\_t::channelMask](#)

The masked channels would be involved in current conversion sequence, beginning with the lowest-order. The available range is in 12-bit.



**5.3.2.0.0.2.2 uint32\_t adc\_conv\_seq\_config\_t::triggerMask**

The available range is 6-bit.

**5.3.2.0.0.2.3 adc\_trigger\_polarity\_t adc\_conv\_seq\_config\_t::triggerPolarity****5.3.2.0.0.2.4 bool adc\_conv\_seq\_config\_t::enableSyncBypass****5.3.2.0.0.2.5 bool adc\_conv\_seq\_config\_t::enableSingleStep****5.3.2.0.0.2.6 adc\_seq\_interrupt\_mode\_t adc\_conv\_seq\_config\_t::interruptMode****5.3.3 struct adc\_result\_info\_t****Data Fields**

- uint32\_t [result](#)  
*Keep the conversion data value.*
- [adc\\_threshold\\_compare\\_status\\_t](#) [thresholdCompareStatus](#)  
*Keep the threshold compare status.*
- [adc\\_threshold\\_crossing\\_status\\_t](#) [thresholdCorssingStatus](#)  
*Keep the threshold crossing status.*
- uint32\_t [channelNumber](#)  
*Keep the channel number for this conversion.*
- bool [overrunFlag](#)  
*Keep the status whether the conversion is overrun or not.*

**5.3.3.0.0.3 Field Documentation****5.3.3.0.0.3.1 uint32\_t adc\_result\_info\_t::result****5.3.3.0.0.3.2 adc\_threshold\_compare\_status\_t adc\_result\_info\_t::thresholdCompareStatus****5.3.3.0.0.3.3 adc\_threshold\_crossing\_status\_t adc\_result\_info\_t::thresholdCorssingStatus****5.3.3.0.0.3.4 uint32\_t adc\_result\_info\_t::channelNumber****5.3.3.0.0.3.5 bool adc\_result\_info\_t::overrunFlag****5.4 Macro Definition Documentation****5.4.1 #define FSL\_ADC\_DRIVER\_VERSION (MAKE\_VERSION(2, 3, 2))****5.5 Enumeration Type Documentation****5.5.1 enum \_adc\_status\_flags**

Enumerator

***kADC\_ThresholdCompareFlagOnChn0*** Threshold comparison event on Channel 0.

## Enumeration Type Documentation

|                                                  |                                                                             |
|--------------------------------------------------|-----------------------------------------------------------------------------|
| <b><i>kADC_ThresholdCompareFlagOnChn1</i></b>    | Threshold comparison event on Channel 1.                                    |
| <b><i>kADC_ThresholdCompareFlagOnChn2</i></b>    | Threshold comparison event on Channel 2.                                    |
| <b><i>kADC_ThresholdCompareFlagOnChn3</i></b>    | Threshold comparison event on Channel 3.                                    |
| <b><i>kADC_ThresholdCompareFlagOnChn4</i></b>    | Threshold comparison event on Channel 4.                                    |
| <b><i>kADC_ThresholdCompareFlagOnChn5</i></b>    | Threshold comparison event on Channel 5.                                    |
| <b><i>kADC_ThresholdCompareFlagOnChn6</i></b>    | Threshold comparison event on Channel 6.                                    |
| <b><i>kADC_ThresholdCompareFlagOnChn7</i></b>    | Threshold comparison event on Channel 7.                                    |
| <b><i>kADC_ThresholdCompareFlagOnChn8</i></b>    | Threshold comparison event on Channel 8.                                    |
| <b><i>kADC_ThresholdCompareFlagOnChn9</i></b>    | Threshold comparison event on Channel 9.                                    |
| <b><i>kADC_ThresholdCompareFlagOnChn10</i></b>   | Threshold comparison event on Channel 10.                                   |
| <b><i>kADC_ThresholdCompareFlagOnChn11</i></b>   | Threshold comparison event on Channel 11.                                   |
| <b><i>kADC_OverrunFlagForChn0</i></b>            | Mirror the OVERRUN status flag from the result register for ADC channel 0.  |
| <b><i>kADC_OverrunFlagForChn1</i></b>            | Mirror the OVERRUN status flag from the result register for ADC channel 1.  |
| <b><i>kADC_OverrunFlagForChn2</i></b>            | Mirror the OVERRUN status flag from the result register for ADC channel 2.  |
| <b><i>kADC_OverrunFlagForChn3</i></b>            | Mirror the OVERRUN status flag from the result register for ADC channel 3.  |
| <b><i>kADC_OverrunFlagForChn4</i></b>            | Mirror the OVERRUN status flag from the result register for ADC channel 4.  |
| <b><i>kADC_OverrunFlagForChn5</i></b>            | Mirror the OVERRUN status flag from the result register for ADC channel 5.  |
| <b><i>kADC_OverrunFlagForChn6</i></b>            | Mirror the OVERRUN status flag from the result register for ADC channel 6.  |
| <b><i>kADC_OverrunFlagForChn7</i></b>            | Mirror the OVERRUN status flag from the result register for ADC channel 7.  |
| <b><i>kADC_OverrunFlagForChn8</i></b>            | Mirror the OVERRUN status flag from the result register for ADC channel 8.  |
| <b><i>kADC_OverrunFlagForChn9</i></b>            | Mirror the OVERRUN status flag from the result register for ADC channel 9.  |
| <b><i>kADC_OverrunFlagForChn10</i></b>           | Mirror the OVERRUN status flag from the result register for ADC channel 10. |
| <b><i>kADC_OverrunFlagForChn11</i></b>           | Mirror the OVERRUN status flag from the result register for ADC channel 11. |
| <b><i>kADC_GlobalOverrunFlagForSeqA</i></b>      | Mirror the global OVERRUN status flag for conversion sequence A.            |
| <b><i>kADC_GlobalOverrunFlagForSeqB</i></b>      | Mirror the global OVERRUN status flag for conversion sequence B.            |
| <b><i>kADC_ConvSeqAInterruptFlag</i></b>         | Sequence A interrupt/DMA trigger.                                           |
| <b><i>kADC_ConvSeqBInterruptFlag</i></b>         | Sequence B interrupt/DMA trigger.                                           |
| <b><i>kADC_ThresholdCompareInterruptFlag</i></b> | Threshold comparison interrupt flag.                                        |
| <b><i>kADC_OverrunInterruptFlag</i></b>          | Overrun interrupt flag.                                                     |

### 5.5.2 enum \_adc\_interrupt\_enable

Note

Not all the interrupt options are listed here

Enumerator

***kADC\_ConvSeqAInterruptEnable*** Enable interrupt upon completion of each individual conversion in sequence A, or entire sequence.

***kADC\_ConvSeqBInterruptEnable*** Enable interrupt upon completion of each individual conversion in sequence B, or entire sequence.

***kADC\_OverrunInterruptEnable*** Enable the detection of an overrun condition on any of the channel data registers will cause an overrun interrupt/DMA trigger.

### 5.5.3 enum adc\_clock\_mode\_t

Enumerator

***kADC\_ClockSynchronousMode*** The ADC clock would be derived from the system clock based on "clockDividerNumber".

***kADC\_ClockAsynchronousMode*** The ADC clock would be based on the SYSCON block's divider.

### 5.5.4 enum adc\_resolution\_t

Enumerator

***kADC\_Resolution6bit*** 6-bit resolution.

***kADC\_Resolution8bit*** 8-bit resolution.

***kADC\_Resolution10bit*** 10-bit resolution.

***kADC\_Resolution12bit*** 12-bit resolution.

### 5.5.5 enum adc\_trigger\_polarity\_t

Enumerator

***kADC\_TriggerPolarityNegativeEdge*** A negative edge launches the conversion sequence on the trigger(s).

***kADC\_TriggerPolarityPositiveEdge*** A positive edge launches the conversion sequence on the trigger(s).

## Enumeration Type Documentation

### 5.5.6 enum adc\_priority\_t

Enumerator

***kADC\_PriorityLow*** This sequence would be preempted when another sequence is started.

***kADC\_PriorityHigh*** This sequence would preempt other sequence even when it is started.

### 5.5.7 enum adc\_seq\_interrupt\_mode\_t

Enumerator

***kADC\_InterruptForEachConversion*** The sequence interrupt/DMA trigger will be set at the end of each individual ADC conversion inside this conversion sequence.

***kADC\_InterruptForEachSequence*** The sequence interrupt/DMA trigger will be set when the entire set of this sequence conversions completes.

### 5.5.8 enum adc\_threshold\_compare\_status\_t

Enumerator

***kADC\_ThresholdCompareInRange*** LOW threshold  $\leq$  conversion value  $\leq$  HIGH threshold.

***kADC\_ThresholdCompareBelowRange*** conversion value  $<$  LOW threshold.

***kADC\_ThresholdCompareAboveRange*** conversion value  $>$  HIGH threshold.

### 5.5.9 enum adc\_threshold\_crossing\_status\_t

Enumerator

***kADC\_ThresholdCrossingNoDetected*** No threshold Crossing detected.

***kADC\_ThresholdCrossingDownward*** Downward Threshold Crossing detected.

***kADC\_ThresholdCrossingUpward*** Upward Threshold Crossing Detected.

### 5.5.10 enum adc\_threshold\_interrupt\_mode\_t

Enumerator

***kADC\_ThresholdInterruptDisabled*** Threshold comparison interrupt is disabled.

***kADC\_ThresholdInterruptOnOutside*** Threshold comparison interrupt is enabled on outside threshold.

***kADC\_ThresholdInterruptOnCrossing*** Threshold comparison interrupt is enabled on crossing threshold.

### 5.5.11 enum adc\_inforeresult\_t

Enumerator

***kADC\_Resolution12bitInfoResultShift*** Info result shift of Resolution12bit.  
***kADC\_Resolution10bitInfoResultShift*** Info result shift of Resolution10bit.  
***kADC\_Resolution8bitInfoResultShift*** Info result shift of Resolution8bit.  
***kADC\_Resolution6bitInfoResultShift*** Info result shift of Resolution6bit.

### 5.5.12 enum adc\_tempsensor\_common\_mode\_t

Enumerator

***kADC\_HighNegativeOffsetAdded*** Temperature sensor common mode: high negative offset added.  
***kADC\_IntermediateNegativeOffsetAdded*** Temperature sensor common mode: intermediate negative offset added.  
***kADC\_NoOffsetAdded*** Temperature sensor common mode: no offset added.  
***kADC\_LowPositiveOffsetAdded*** Temperature sensor common mode: low positive offset added.

### 5.5.13 enum adc\_second\_control\_t

Enumerator

***kADC\_Impedance6210hm*** Extand ADC sampling time according to source impedance 1: 0.621 kOhm.  
***kADC\_Impedance55kOhm*** Extand ADC sampling time according to source impedance 20 (default): 55 kOhm.  
***kADC\_Impedance87kOhm*** Extand ADC sampling time according to source impedance 31: 87 k-Ohm.  
***kADC\_NormalFunctionalMode*** TEST mode: Normal functional mode.  
***kADC\_MultiplexeTestMode*** TEST mode: Multiplexer test mode.  
***kADC\_ADCInUnityGainMode*** TEST mode: ADC in unity gain mode.

## 5.6 Function Documentation

### 5.6.1 void ADC\_Init ( ADC\_Type \* *base*, const adc\_config\_t \* *config* )

## Function Documentation

### Parameters

|               |                                                                           |
|---------------|---------------------------------------------------------------------------|
| <i>base</i>   | ADC peripheral base address.                                              |
| <i>config</i> | Pointer to configuration structure, see to <a href="#">adc_config_t</a> . |

### 5.6.2 void ADC\_Deinit ( ADC\_Type \* *base* )

#### Parameters

|             |                              |
|-------------|------------------------------|
| <i>base</i> | ADC peripheral base address. |
|-------------|------------------------------|

### 5.6.3 void ADC\_GetDefaultConfig ( adc\_config\_t \* *config* )

This function initializes the initial configuration structure with an available settings. The default values are:

```
* config->clockMode = kADC_ClockSynchronousMode;
* config->clockDividerNumber = 0U;
* config->resolution = kADC_Resolution12bit;
* config->enableBypassCalibration = false;
* config->sampleTimeNumber = 0U;
*
```

#### Parameters

|               |                                     |
|---------------|-------------------------------------|
| <i>config</i> | Pointer to configuration structure. |
|---------------|-------------------------------------|

### 5.6.4 void ADC\_EnableTemperatureSensor ( ADC\_Type \* *base*, bool *enable* )

When enabling the internal temperature sensor measurement, the channel 0 would be connected to internal sensor instead of external pin.

#### Parameters

|             |                              |
|-------------|------------------------------|
| <i>base</i> | ADC peripheral base address. |
|-------------|------------------------------|

|               |                                        |
|---------------|----------------------------------------|
| <i>enable</i> | Switcher to enable the feature or not. |
|---------------|----------------------------------------|

### 5.6.5 static void ADC\_EnableConvSeqA ( ADC\_Type \* *base*, bool *enable* ) [inline], [static]

In order to avoid spuriously triggering the sequence, the trigger to conversion sequence should be ready before the sequence is ready. when the sequence is disabled, the trigger would be ignored. Also, it is suggested to disable the sequence during changing the sequence's setting.

Parameters

|               |                                        |
|---------------|----------------------------------------|
| <i>base</i>   | ADC peripheral base address.           |
| <i>enable</i> | Switcher to enable the feature or not. |

### 5.6.6 void ADC\_SetConvSeqAConfig ( ADC\_Type \* *base*, const adc\_conv\_seq\_config\_t \* *config* )

Parameters

|               |                                                                                    |
|---------------|------------------------------------------------------------------------------------|
| <i>base</i>   | ADC peripheral base address.                                                       |
| <i>config</i> | Pointer to configuration structure, see to <a href="#">adc_conv_seq_config_t</a> . |

### 5.6.7 static void ADC\_DoSoftwareTriggerConvSeqA ( ADC\_Type \* *base* ) [inline], [static]

Parameters

|             |                              |
|-------------|------------------------------|
| <i>base</i> | ADC peripheral base address. |
|-------------|------------------------------|

### 5.6.8 static void ADC\_EnableConvSeqABurstMode ( ADC\_Type \* *base*, bool *enable* ) [inline], [static]

Enable the burst mode would cause the conversion sequence to be continuously cycled through. Other triggers would be ignored while this mode is enabled. Repeated conversions could be halted by disabling this mode. And the sequence currently in process will be completed before conversions are terminated. Note that a new sequence could begin just before the burst mode is disabled.

## Function Documentation

### Parameters

|               |                                  |
|---------------|----------------------------------|
| <i>base</i>   | ADC peripheral base address.     |
| <i>enable</i> | Switcher to enable this feature. |

### 5.6.9 static void ADC\_SetConvSeqAHighPriority ( ADC\_Type \* *base* ) [inline], [static]

### Parameters

|             |                              |
|-------------|------------------------------|
| <i>base</i> | ADC peripheral bass address. |
|-------------|------------------------------|

### 5.6.10 static void ADC\_EnableConvSeqB ( ADC\_Type \* *base*, bool *enable* ) [inline], [static]

In order to avoid spuriously triggering the sequence, the trigger to conversion sequence should be ready before the sequence is ready. when the sequence is disabled, the trigger would be ignored. Also, it is suggested to disable the sequence during changing the sequence's setting.

### Parameters

|               |                                        |
|---------------|----------------------------------------|
| <i>base</i>   | ADC peripheral base address.           |
| <i>enable</i> | Switcher to enable the feature or not. |

### 5.6.11 void ADC\_SetConvSeqBConfig ( ADC\_Type \* *base*, const adc\_conv\_seq\_config\_t \* *config* )

### Parameters

|               |                                                                                    |
|---------------|------------------------------------------------------------------------------------|
| <i>base</i>   | ADC peripheral base address.                                                       |
| <i>config</i> | Pointer to configuration structure, see to <a href="#">adc_conv_seq_config_t</a> . |

### 5.6.12 static void ADC\_DoSoftwareTriggerConvSeqB ( ADC\_Type \* *base* ) [inline], [static]



## Parameters

|             |                              |
|-------------|------------------------------|
| <i>base</i> | ADC peripheral base address. |
|-------------|------------------------------|

### 5.6.13 static void ADC\_EnableConvSeqBBurstMode ( ADC\_Type \* *base*, bool *enable* ) [inline], [static]

Enable the burst mode would cause the conversion sequence to be continuously cycled through. Other triggers would be ignored while this mode is enabled. Repeated conversions could be halted by disabling this mode. And the sequence currently in process will be completed before conversions are terminated. Note that a new sequence could begin just before the burst mode is disabled.

## Parameters

|               |                                  |
|---------------|----------------------------------|
| <i>base</i>   | ADC peripheral base address.     |
| <i>enable</i> | Switcher to enable this feature. |

### 5.6.14 static void ADC\_SetConvSeqBHPriority ( ADC\_Type \* *base* ) [inline], [static]

## Parameters

|             |                              |
|-------------|------------------------------|
| <i>base</i> | ADC peripheral bass address. |
|-------------|------------------------------|

### 5.6.15 bool ADC\_GetConvSeqAGlobalConversionResult ( ADC\_Type \* *base*, adc\_result\_info\_t \* *info* )

## Parameters

|             |                                                                              |
|-------------|------------------------------------------------------------------------------|
| <i>base</i> | ADC peripheral base address.                                                 |
| <i>info</i> | Pointer to information structure, see to <a href="#">adc_result_info_t</a> ; |

## Return values

## Function Documentation

|              |                                         |
|--------------|-----------------------------------------|
| <i>true</i>  | The conversion result is ready.         |
| <i>false</i> | The conversion result is not ready yet. |

### 5.6.16 **bool ADC\_GetConvSeqBGlobalConversionResult ( ADC\_Type \* *base*, adc\_result\_info\_t \* *info* )**

#### Parameters

|             |                                                                              |
|-------------|------------------------------------------------------------------------------|
| <i>base</i> | ADC peripheral base address.                                                 |
| <i>info</i> | Pointer to information structure, see to <a href="#">adc_result_info_t</a> ; |

#### Return values

|              |                                         |
|--------------|-----------------------------------------|
| <i>true</i>  | The conversion result is ready.         |
| <i>false</i> | The conversion result is not ready yet. |

### 5.6.17 **bool ADC\_GetChannelConversionResult ( ADC\_Type \* *base*, uint32\_t *channel*, adc\_result\_info\_t \* *info* )**

#### Parameters

|                |                                                                              |
|----------------|------------------------------------------------------------------------------|
| <i>base</i>    | ADC peripheral base address.                                                 |
| <i>channel</i> | The indicated channel number.                                                |
| <i>info</i>    | Pointer to information structure, see to <a href="#">adc_result_info_t</a> ; |

#### Return values

|              |                                         |
|--------------|-----------------------------------------|
| <i>true</i>  | The conversion result is ready.         |
| <i>false</i> | The conversion result is not ready yet. |

### 5.6.18 **static void ADC\_SetThresholdPair0 ( ADC\_Type \* *base*, uint32\_t *lowValue*, uint32\_t *highValue* ) [inline], [static]**

## Parameters

|                  |                              |
|------------------|------------------------------|
| <i>base</i>      | ADC peripheral base address. |
| <i>lowValue</i>  | LOW threshold value.         |
| <i>highValue</i> | HIGH threshold value.        |

**5.6.19 static void ADC\_SetThresholdPair1 ( ADC\_Type \* *base*, uint32\_t *lowValue*, uint32\_t *highValue* ) [inline], [static]**

## Parameters

|                  |                                                           |
|------------------|-----------------------------------------------------------|
| <i>base</i>      | ADC peripheral base address.                              |
| <i>lowValue</i>  | LOW threshold value. The available value is with 12-bit.  |
| <i>highValue</i> | HIGH threshold value. The available value is with 12-bit. |

**5.6.20 static void ADC\_SetChannelWithThresholdPair0 ( ADC\_Type \* *base*, uint32\_t *channelMask* ) [inline], [static]**

## Parameters

|                    |                              |
|--------------------|------------------------------|
| <i>base</i>        | ADC peripheral base address. |
| <i>channelMask</i> | Indicated channels' mask.    |

**5.6.21 static void ADC\_SetChannelWithThresholdPair1 ( ADC\_Type \* *base*, uint32\_t *channelMask* ) [inline], [static]**

## Parameters

|                    |                              |
|--------------------|------------------------------|
| <i>base</i>        | ADC peripheral base address. |
| <i>channelMask</i> | Indicated channels' mask.    |

**5.6.22 static void ADC\_EnableInterrupts ( ADC\_Type \* *base*, uint32\_t *mask* ) [inline], [static]**

## Function Documentation

### Parameters

|             |                                                                                                                   |
|-------------|-------------------------------------------------------------------------------------------------------------------|
| <i>base</i> | ADC peripheral base address.                                                                                      |
| <i>mask</i> | Mask of interrupt mask value for global block except each channel, see to <a href="#">_adc_interrupt_enable</a> . |

**5.6.23 static void ADC\_DisableInterrupts ( ADC\_Type \* *base*, uint32\_t *mask* ) [inline], [static]**

### Parameters

|             |                                                                                                                   |
|-------------|-------------------------------------------------------------------------------------------------------------------|
| <i>base</i> | ADC peripheral base address.                                                                                      |
| <i>mask</i> | Mask of interrupt mask value for global block except each channel, see to <a href="#">_adc_interrupt_enable</a> . |

**5.6.24 static void ADC\_EnableShresholdCompareInterrupt ( ADC\_Type \* *base*, uint32\_t *channel*, adc\_threshold\_interrupt\_mode\_t *mode* ) [inline], [static]**

**5.6.25 static void ADC\_EnableThresholdCompareInterrupt ( ADC\_Type \* *base*, uint32\_t *channel*, adc\_threshold\_interrupt\_mode\_t *mode* ) [inline], [static]**

### Parameters

|                |                                                                                                     |
|----------------|-----------------------------------------------------------------------------------------------------|
| <i>base</i>    | ADC peripheral base address.                                                                        |
| <i>channel</i> | Channel number.                                                                                     |
| <i>mode</i>    | Interrupt mode for threshold compare event, see to <a href="#">adc_threshold_interrupt_mode_t</a> . |

**5.6.26 static uint32\_t ADC\_GetStatusFlags ( ADC\_Type \* *base* ) [inline], [static]**

## Parameters

|             |                              |
|-------------|------------------------------|
| <i>base</i> | ADC peripheral base address. |
|-------------|------------------------------|

## Returns

Mask of status flags of module, see to [\\_adc\\_status\\_flags](#).

**5.6.27 static void ADC\_ClearStatusFlags ( ADC\_Type \* *base*, uint32\_t *mask* )**  
**[inline], [static]**

## Parameters

|             |                                                                            |
|-------------|----------------------------------------------------------------------------|
| <i>base</i> | ADC peripheral base address.                                               |
| <i>mask</i> | Mask of status flags of module, see to <a href="#">_adc_status_flags</a> . |



## Chapter 6

# AES: AES encryption decryption driver

### 6.1 Overview

The MCUXpresso SDK provides a peripheral driver for the AES module in MCUXpresso SDK devices.

The driver provides blocking synchronous APIs. The AES operations are complete (and results are made available for further usage) when a function returns. When called, these functions do not return until an AES operation is complete. These functions use main CPU for simple polling loops to determine operation complete or error status, as well as plaintext or ciphertext data movements. The driver functions are not re-entrant. These functions provide typical interface to upper layer or application software.

### 6.2 AES Driver Initialization and Configuration

Clock to the AES module has to be enabled before using the driver API. The function [AES\\_SetKey\(\)](#) has to be used to store encryption key into device registers prior to using other API.

### 6.3 Comments about API usage in RTOS

AES operations provided by this driver are not re-entrant. Because of this, the application software should ensure the AES module operation is not requested from different tasks or interrupt service routines while an operation is in progress.

### 6.4 AES Driver Examples

Encrypt plaintext and decrypt it back by AES engine Refer to the driver examples codes located at <SDK\_ROOT>/boards/<BOARD>/driver\_examples/aes Encrypts AES using CTR block mode. Refer to the driver examples codes located at <SDK\_ROOT>/boards/<BOARD>/driver\_examples/aes Generation of GCM tag only Refer to the driver examples codes located at <SDK\_ROOT>/boards/<BOARD>/driver\_examples/aes

#### Files

- file [fsl\\_aes.h](#)

#### Driver version

- `#define FSL_AES_DRIVER_VERSION (MAKE_VERSION(2, 0, 1))`  
*Defines LPC AES driver version 2.0.1.*

#### AES Functional Operation

- [status\\_t AES\\_SetKey](#) (AES\_Type \*base, const uint8\_t \*key, size\_t keySize)  
*Sets AES key.*

## Macro Definition Documentation

- **status\_t AES\_EncryptEcb** (AES\_Type \*base, const uint8\_t \*plaintext, uint8\_t \*ciphertext, size\_t size)  
*Encrypts AES using the ECB block mode.*
- **status\_t AES\_DecryptEcb** (AES\_Type \*base, const uint8\_t \*ciphertext, uint8\_t \*plaintext, size\_t size)  
*Decrypts AES using the ECB block mode.*
- **status\_t AES\_EncryptCbc** (AES\_Type \*base, const uint8\_t \*plaintext, uint8\_t \*ciphertext, size\_t size, const uint8\_t iv[AES\_IV\_SIZE])  
*Encrypts AES using CBC block mode.*
- **status\_t AES\_DecryptCbc** (AES\_Type \*base, const uint8\_t \*ciphertext, uint8\_t \*plaintext, size\_t size, const uint8\_t iv[AES\_IV\_SIZE])  
*Decrypts AES using CBC block mode.*
- **status\_t AES\_EncryptCfb** (AES\_Type \*base, const uint8\_t \*plaintext, uint8\_t \*ciphertext, size\_t size, const uint8\_t iv[AES\_IV\_SIZE])  
*Encrypts AES using CFB block mode.*
- **status\_t AES\_DecryptCfb** (AES\_Type \*base, const uint8\_t \*ciphertext, uint8\_t \*plaintext, size\_t size, const uint8\_t iv[AES\_IV\_SIZE])  
*Decrypts AES using CFB block mode.*
- **status\_t AES\_EncryptOfb** (AES\_Type \*base, const uint8\_t \*plaintext, uint8\_t \*ciphertext, size\_t size, const uint8\_t iv[AES\_IV\_SIZE])  
*Encrypts AES using OFB block mode.*
- **status\_t AES\_DecryptOfb** (AES\_Type \*base, const uint8\_t \*ciphertext, uint8\_t \*plaintext, size\_t size, const uint8\_t iv[AES\_IV\_SIZE])  
*Decrypts AES using OFB block mode.*
- **status\_t AES\_CryptCtr** (AES\_Type \*base, const uint8\_t \*input, uint8\_t \*output, size\_t size, uint8\_t counter[AES\_BLOCK\_SIZE], uint8\_t counterlast[AES\_BLOCK\_SIZE], size\_t \*szLeft)  
*Encrypts or decrypts AES using CTR block mode.*
- **status\_t AES\_EncryptTagGcm** (AES\_Type \*base, const uint8\_t \*plaintext, uint8\_t \*ciphertext, size\_t size, const uint8\_t \*iv, size\_t ivSize, const uint8\_t \*aad, size\_t aadSize, uint8\_t \*tag, size\_t tagSize)  
*Encrypts AES and tags using GCM block mode.*
- **status\_t AES\_DecryptTagGcm** (AES\_Type \*base, const uint8\_t \*ciphertext, uint8\_t \*plaintext, size\_t size, const uint8\_t \*iv, size\_t ivSize, const uint8\_t \*aad, size\_t aadSize, const uint8\_t \*tag, size\_t tagSize)  
*Decrypts AES and authenticates using GCM block mode.*
- **void AES\_Init** (AES\_Type \*base)
- **void AES\_Deinit** (AES\_Type \*base)
- **#define AES\_BLOCK\_SIZE** 16  
*AES block size in bytes.*
- **#define AES\_IV\_SIZE** 16  
*AES Input Vector size in bytes.*

## 6.5 Macro Definition Documentation

### 6.5.1 #define FSL\_AES\_DRIVER\_VERSION (MAKE\_VERSION(2, 0, 1))

Change log:

- Version 2.0.0



- initial version
- Version 2.0.1
  - GCM constant time tag comparison

## 6.6 Function Documentation

### 6.6.1 `status_t AES_SetKey ( AES_Type * base, const uint8_t * key, size_t keySize )`

Sets AES key.

Parameters

|                |                                                         |
|----------------|---------------------------------------------------------|
| <i>base</i>    | AES peripheral base address                             |
| <i>key</i>     | Input key to use for encryption or decryption           |
| <i>keySize</i> | Size of the input key, in bytes. Must be 16, 24, or 32. |

Returns

Status from Set Key operation

### 6.6.2 `status_t AES_EncryptEcb ( AES_Type * base, const uint8_t * plaintext, uint8_t * ciphertext, size_t size )`

Encrypts AES using the ECB block mode.

Parameters

|     |                   |                                                                       |
|-----|-------------------|-----------------------------------------------------------------------|
|     | <i>base</i>       | AES peripheral base address                                           |
|     | <i>plaintext</i>  | Input plain text to encrypt                                           |
| out | <i>ciphertext</i> | Output cipher text                                                    |
|     | <i>size</i>       | Size of input and output data in bytes. Must be multiple of 16 bytes. |

Returns

Status from encrypt operation

### 6.6.3 `status_t AES_DecryptEcb ( AES_Type * base, const uint8_t * ciphertext, uint8_t * plaintext, size_t size )`

Decrypts AES using the ECB block mode.

## Function Documentation

### Parameters

|     |                   |                                                                       |
|-----|-------------------|-----------------------------------------------------------------------|
|     | <i>base</i>       | AES peripheral base address                                           |
|     | <i>ciphertext</i> | Input ciphertext to decrypt                                           |
| out | <i>plaintext</i>  | Output plain text                                                     |
|     | <i>size</i>       | Size of input and output data in bytes. Must be multiple of 16 bytes. |

### Returns

Status from decrypt operation

**6.6.4** `status_t AES_EncryptCbc ( AES_Type * base, const uint8_t * plaintext,  
uint8_t * ciphertext, size_t size, const uint8_t iv[AES_IV_SIZE] )`

### Parameters

|     |                   |                                                                       |
|-----|-------------------|-----------------------------------------------------------------------|
|     | <i>base</i>       | AES peripheral base address                                           |
|     | <i>plaintext</i>  | Input plain text to encrypt                                           |
| out | <i>ciphertext</i> | Output cipher text                                                    |
|     | <i>size</i>       | Size of input and output data in bytes. Must be multiple of 16 bytes. |
|     | <i>iv</i>         | Input initial vector to combine with the first input block.           |

### Returns

Status from encrypt operation

**6.6.5** `status_t AES_DecryptCbc ( AES_Type * base, const uint8_t * ciphertext,  
uint8_t * plaintext, size_t size, const uint8_t iv[AES_IV_SIZE] )`

### Parameters

|  |             |                             |
|--|-------------|-----------------------------|
|  | <i>base</i> | AES peripheral base address |
|--|-------------|-----------------------------|

|     |                   |                                                                       |
|-----|-------------------|-----------------------------------------------------------------------|
|     | <i>ciphertext</i> | Input cipher text to decrypt                                          |
| out | <i>plaintext</i>  | Output plain text                                                     |
|     | <i>size</i>       | Size of input and output data in bytes. Must be multiple of 16 bytes. |
|     | <i>iv</i>         | Input initial vector to combine with the first input block.           |

## Returns

Status from decrypt operation

**6.6.6** `status_t AES_EncryptCfb ( AES_Type * base, const uint8_t * plaintext,  
uint8_t * ciphertext, size_t size, const uint8_t iv[AES_IV_SIZE] )`

## Parameters

|     |                   |                                                                       |
|-----|-------------------|-----------------------------------------------------------------------|
|     | <i>base</i>       | AES peripheral base address                                           |
|     | <i>plaintext</i>  | Input plain text to encrypt                                           |
| out | <i>ciphertext</i> | Output cipher text                                                    |
|     | <i>size</i>       | Size of input and output data in bytes. Must be multiple of 16 bytes. |
|     | <i>iv</i>         | Input Initial vector to be used as the first input block.             |

## Returns

Status from encrypt operation

**6.6.7** `status_t AES_DecryptCfb ( AES_Type * base, const uint8_t * ciphertext,  
uint8_t * plaintext, size_t size, const uint8_t iv[AES_IV_SIZE] )`

## Parameters

|  |                   |                              |
|--|-------------------|------------------------------|
|  | <i>base</i>       | AES peripheral base address  |
|  | <i>ciphertext</i> | Input cipher text to decrypt |

## Function Documentation

|     |                  |                                                                       |
|-----|------------------|-----------------------------------------------------------------------|
| out | <i>plaintext</i> | Output plain text                                                     |
|     | <i>size</i>      | Size of input and output data in bytes. Must be multiple of 16 bytes. |
|     | <i>iv</i>        | Input Initial vector to be used as the first input block.             |

### Returns

Status from decrypt operation

**6.6.8** `status_t AES_EncryptOfb ( AES_Type * base, const uint8_t * plaintext,  
uint8_t * ciphertext, size_t size, const uint8_t iv[AES_IV_SIZE] )`

### Parameters

|     |                   |                                                           |
|-----|-------------------|-----------------------------------------------------------|
|     | <i>base</i>       | AES peripheral base address                               |
|     | <i>plaintext</i>  | Input plain text to encrypt                               |
| out | <i>ciphertext</i> | Output cipher text                                        |
|     | <i>size</i>       | Size of input and output data in bytes.                   |
|     | <i>iv</i>         | Input Initial vector to be used as the first input block. |

### Returns

Status from encrypt operation

**6.6.9** `status_t AES_DecryptOfb ( AES_Type * base, const uint8_t * ciphertext,  
uint8_t * plaintext, size_t size, const uint8_t iv[AES_IV_SIZE] )`

### Parameters

|     |                   |                              |
|-----|-------------------|------------------------------|
|     | <i>base</i>       | AES peripheral base address  |
|     | <i>ciphertext</i> | Input cipher text to decrypt |
| out | <i>plaintext</i>  | Output plain text            |

|  |             |                                                           |
|--|-------------|-----------------------------------------------------------|
|  | <i>size</i> | Size of input and output data in bytes.                   |
|  | <i>iv</i>   | Input Initial vector to be used as the first input block. |

## Returns

Status from decrypt operation

#### 6.6.10 **status\_t AES\_CryptCtr ( AES\_Type \* *base*, const uint8\_t \* *input*, uint8\_t \* *output*, size\_t *size*, uint8\_t *counter*[AES\_BLOCK\_SIZE], uint8\_t *counterlast*[AES\_BLOCK\_SIZE], size\_t \* *szLeft* )**

Encrypts or decrypts AES using CTR block mode. AES CTR mode uses only forward AES cipher and same algorithm for encryption and decryption. The only difference between encryption and decryption is that, for encryption, the input argument is plain text and the output argument is cipher text. For decryption, the input argument is cipher text and the output argument is plain text.

## Parameters

|         |                    |                                                                                                               |
|---------|--------------------|---------------------------------------------------------------------------------------------------------------|
|         | <i>base</i>        | AES peripheral base address                                                                                   |
|         | <i>input</i>       | Input data for CTR block mode                                                                                 |
| out     | <i>output</i>      | Output data for CTR block mode                                                                                |
|         | <i>size</i>        | Size of input and output data in bytes                                                                        |
| in, out | <i>counter</i>     | Input counter (updates on return)                                                                             |
| out     | <i>counterlast</i> | Output cipher of last counter, for chained CTR calls. NULL can be passed if chained calls are not used.       |
| out     | <i>szLeft</i>      | Output number of bytes in left unused in counterlast block. NULL can be passed if chained calls are not used. |

## Returns

Status from crypt operation

#### 6.6.11 **status\_t AES\_EncryptTagGcm ( AES\_Type \* *base*, const uint8\_t \* *plaintext*, uint8\_t \* *ciphertext*, size\_t *size*, const uint8\_t \* *iv*, size\_t *ivSize*, const uint8\_t \* *aad*, size\_t *aadSize*, uint8\_t \* *tag*, size\_t *tagSize* )**

Encrypts AES and optionally tags using GCM block mode. If plaintext is NULL, only the GHASH is calculated and output in the 'tag' field.

## Function Documentation

### Parameters

|     |                   |                                                                             |
|-----|-------------------|-----------------------------------------------------------------------------|
|     | <i>base</i>       | AES peripheral base address                                                 |
|     | <i>plaintext</i>  | Input plain text to encrypt                                                 |
| out | <i>ciphertext</i> | Output cipher text.                                                         |
|     | <i>size</i>       | Size of input and output data in bytes                                      |
|     | <i>iv</i>         | Input initial vector                                                        |
|     | <i>ivSize</i>     | Size of the IV                                                              |
|     | <i>aad</i>        | Input additional authentication data                                        |
|     | <i>aadSize</i>    | Input size in bytes of AAD                                                  |
| out | <i>tag</i>        | Output hash tag. Set to NULL to skip tag processing.                        |
|     | <i>tagSize</i>    | Input size of the tag to generate, in bytes. Must be 4,8,12,13,14,15 or 16. |

### Returns

Status from encrypt operation

**6.6.12** `status_t AES_DecryptTagGcm ( AES_Type * base, const uint8_t * ciphertext, uint8_t * plaintext, size_t size, const uint8_t * iv, size_t ivSize, const uint8_t * aad, size_t aadSize, const uint8_t * tag, size_t tagSize )`

Decrypts AES and optionally authenticates using GCM block mode. If ciphertext is NULL, only the GHASH is calculated and compared with the received GHASH in 'tag' field.

### Parameters

|     |                   |                                        |
|-----|-------------------|----------------------------------------|
|     | <i>base</i>       | AES peripheral base address            |
|     | <i>ciphertext</i> | Input cipher text to decrypt           |
| out | <i>plaintext</i>  | Output plain text.                     |
|     | <i>size</i>       | Size of input and output data in bytes |
|     | <i>iv</i>         | Input initial vector                   |

|  |                |                                                                       |
|--|----------------|-----------------------------------------------------------------------|
|  | <i>ivSize</i>  | Size of the IV                                                        |
|  | <i>aad</i>     | Input additional authentication data                                  |
|  | <i>aadSize</i> | Input size in bytes of AAD                                            |
|  | <i>tag</i>     | Input hash tag to compare. Set to NULL to skip tag processing.        |
|  | <i>tagSize</i> | Input size of the tag, in bytes. Must be 4, 8, 12, 13, 14, 15, or 16. |

## Returns

Status from decrypt operation





## Chapter 7

### Clock: Clock driver

#### 7.1 Overview

The MCUXpresso SDK provides a clock driver for MCUXpresso SDK devices.

#### 7.2 Function groups

Clock driver provides these functions:

- Functions to obtain frequency of specified clock
- Functions to configure the clock selection muxes.
- Functions to setup peripheral clock dividers
- Functions to enable specific AHB clock channel

##### 7.2.1 SYSCON Clock frequency functions

SYSCON clock module provides clocks, such as ADCCLK, DMICCLK, FXCOMCLK, WDTOSC, RTCOSC and SYSPLL. The functions [CLOCK\\_EnableClock\(\)](#) and [CLOCK\\_DisableClock\(\)](#) enables and disables the various clocks. The SYSCON clock driver provides functions to get the frequency of clocks, such as [CLOCK\\_GetFreq\(\)](#),

##### 7.2.2 SYSCON clock Selection Muxes

The SYSCON clock driver provides the function to configure the clock selected. The function [CLOCK\\_AttachClk\(\)](#) is implemented for this. The function selects the clock source for a particular peripheral like MAINCLK, DMIC, FLEXCOMM, USB, ADC and PLL.

##### 7.2.3 SYSCON clock dividers

The SYSCON clock module provides the function to setup the peripheral clock dividers. The function [CLOCK\\_SetClkDiv\(\)](#) configures the CLKDIV registers for various peripherals like USB, DMIC, I2S, SYSTICK, AHB, ADC and also for CLKOUT and TRACE functions.

#### Files

- file [fsl\\_clock.h](#)

## Function groups

## Data Structures

- struct [ClockCapacitanceCompensation\\_t](#)  
*Board specific constant capacitance characteristics Should be supplied by board manufacturer for best performance. [More...](#)*

## Macros

- #define [FLEXCOMM\\_CLOCKS](#)  
*Clock ip name array for FLEXCOMM.*
- #define [CTIMER\\_CLOCKS](#)  
*Clock ip name array for CTIMER.*
- #define [GINT\\_CLOCKS](#)  
*Clock ip name array for GINT.*
- #define [WWDT\\_CLOCKS](#)  
*Clock ip name array for WWDT.*
- #define [DMIC\\_CLOCKS](#)  
*Clock ip name array for DMIC.*
- #define [ADC\\_CLOCKS](#)  
*Clock ip name array for ADC.*
- #define [SPIFI\\_CLOCKS](#)  
*Clock ip name array for SPIFI.*
- #define [GPIO\\_CLOCKS](#)  
*Clock ip name array for GPIO.*
- #define [DMA\\_CLOCKS](#)  
*Clock ip name array for DMA.*

## Enumerations

- enum [CHIP\\_SYSCON\\_MAINCLKSRC\\_T](#) {  
    [SYSCON\\_MAINCLKSRC\\_FRO12M](#),  
    [SYSCON\\_MAINCLKSRC\\_OSC32K](#),  
    [SYSCON\\_MAINCLKSRC\\_XTAL32M](#),  
    [SYSCON\\_MAINCLKSRC\\_FRO32M](#),  
    [SYSCON\\_MAINCLKSRC\\_FRO48M](#),  
    [SYSCON\\_MAINCLKSRC\\_EXT](#),  
    [SYSCON\\_MAINCLKSRC\\_FRO1M](#) }  
*Clock sources for main system clock.*
- enum [CHIP\\_SYSCON\\_FRGCLKSRC\\_T](#) {  
    [SYSCON\\_FRGCLKSRC\\_MAINCLK](#),  
    [SYSCON\\_FRGCLKSRC\\_OSC32M](#),  
    [SYSCON\\_FRGCLKSRC\\_FRO48MHZ](#),  
    [SYSCON\\_FRGCLKSRC\\_NONE](#) }  
*Fractional Divider clock sources.*
- enum [clock\\_name\\_t](#) {  
    [kCLOCK\\_Rom](#) = CLK\_GATE\_DEFINE(AHB\_CLK\_CTRL0, SYSCON\_AHBCLKCTRL0\_ROM\_SHIFT),  
    [kCLOCK\\_Sram0](#) = CLK\_GATE\_DEFINE(AHB\_CLK\_CTRL0, SYSCON\_AHBCLKCTRL0\_S-

```

RAM_CTRL0_SHIFT),
kCLOCK_Sram1 = CLK_GATE_DEFINE(AHB_CLK_CTRL0, SYSCON_AHBCLKCTRL0_S-
RAM_CTRL1_SHIFT),
kCLOCK_Flash = CLK_GATE_DEFINE(AHB_CLK_CTRL0, SYSCON_AHBCLKCTRL0_FL-
ASH_SHIFT),
kCLOCK_Spifi = CLK_GATE_DEFINE(AHB_CLK_CTRL0, SYSCON_AHBCLKCTRL0_SPI-
FI_SHIFT),
kCLOCK_InputMux = CLK_GATE_DEFINE(AHB_CLK_CTRL0, SYSCON_AHBCLKCTRL0-
_MUX_SHIFT),
kCLOCK_Iocon = CLK_GATE_DEFINE(AHB_CLK_CTRL0, SYSCON_AHBCLKCTRL0_IO-
CON_SHIFT),
kCLOCK_Gpio0 = CLK_GATE_DEFINE(AHB_CLK_CTRL0, SYSCON_AHBCLKCTRL0_G-
PIO_SHIFT),
kCLOCK_Pint = CLK_GATE_DEFINE(AHB_CLK_CTRL0, SYSCON_AHBCLKCTRL0_PIN-
T_SHIFT),
kCLOCK_Dma = CLK_GATE_DEFINE(AHB_CLK_CTRL0, SYSCON_AHBCLKCTRL0_DM-
A_SHIFT),
kCLOCK_Iso7816 = CLK_GATE_DEFINE(AHB_CLK_CTRL0, SYSCON_AHBCLKCTRL0_I-
SO7816_SHIFT),
kCLOCK_WdtOsc = CLK_GATE_DEFINE(AHB_CLK_CTRL0, SYSCON_AHBCLKCTRL0_-
WWDT_SHIFT),
kCLOCK_Rtc = CLK_GATE_DEFINE(AHB_CLK_CTRL0, SYSCON_AHBCLKCTRL0_RTC-
_SHIFT),
kCLOCK_Analog,
kCLOCK_WakeTmr,
kCLOCK_Adc0 = CLK_GATE_DEFINE(AHB_CLK_CTRL0, SYSCON_AHBCLKCTRL0_AD-
C_SHIFT),
kCLOCK_Efuse = CLK_GATE_DEFINE(AHB_CLK_CTRL0, SYSCON_AHBCLKCTRL0_EF-
USE_SHIFT),
kCLOCK_FlexComm0 = CLK_GATE_DEFINE(AHB_CLK_CTRL1, SYSCON_AHBCLKCTR-
L1_USART0_SHIFT),
kCLOCK_FlexComm1 = CLK_GATE_DEFINE(AHB_CLK_CTRL1, SYSCON_AHBCLKCTR-
L1_USART1_SHIFT),
kCLOCK_FlexComm2 = CLK_GATE_DEFINE(AHB_CLK_CTRL1, SYSCON_AHBCLKCTR-
L1_I2C0_SHIFT),
kCLOCK_FlexComm3 = CLK_GATE_DEFINE(AHB_CLK_CTRL1, SYSCON_AHBCLKCTR-
L1_I2C1_SHIFT),
kCLOCK_FlexComm4 = CLK_GATE_DEFINE(AHB_CLK_CTRL1, SYSCON_AHBCLKCTR-
L1_SPI0_SHIFT),
kCLOCK_FlexComm5 = CLK_GATE_DEFINE(AHB_CLK_CTRL1, SYSCON_AHBCLKCTR-
L1_SPI1_SHIFT),
kCLOCK_Ir = CLK_GATE_DEFINE(AHB_CLK_CTRL1, SYSCON_AHBCLKCTRL1_IR_SH-
IFT),
kCLOCK_Pwm = CLK_GATE_DEFINE(AHB_CLK_CTRL1, SYSCON_AHBCLKCTRL1_PW-

```

## Function groups

```
M_SHIFT),
kCLOCK_Rng = CLK_GATE_DEFINE(AHB_CLK_CTRL1, SYSCON_AHBCLKCTRL1_RN-
G_SHIFT),
kCLOCK_FlexComm6 = CLK_GATE_DEFINE(AHB_CLK_CTRL1, SYSCON_AHBCLKCTR-
L1_I2C2_SHIFT),
kCLOCK_Usart0 = CLK_GATE_DEFINE(AHB_CLK_CTRL1, SYSCON_AHBCLKCTRL1_U-
SART0_SHIFT),
kCLOCK_Usart1 = CLK_GATE_DEFINE(AHB_CLK_CTRL1, SYSCON_AHBCLKCTRL1_U-
SART1_SHIFT),
kCLOCK_I2c0 = CLK_GATE_DEFINE(AHB_CLK_CTRL1, SYSCON_AHBCLKCTRL1_I2-
C0_SHIFT),
kCLOCK_I2c1 = CLK_GATE_DEFINE(AHB_CLK_CTRL1, SYSCON_AHBCLKCTRL1_I2-
C1_SHIFT),
kCLOCK_Spi0 = CLK_GATE_DEFINE(AHB_CLK_CTRL1, SYSCON_AHBCLKCTRL1_SP-
IO_SHIFT),
kCLOCK_Spi1 = CLK_GATE_DEFINE(AHB_CLK_CTRL1, SYSCON_AHBCLKCTRL1_SP-
I1_SHIFT),
kCLOCK_I2c2 = CLK_GATE_DEFINE(AHB_CLK_CTRL1, SYSCON_AHBCLKCTRL1_I2-
C2_SHIFT),
kCLOCK_BLE = CLK_GATE_DEFINE(AHB_CLK_CTRL1, SYSCON_AHBCLKCTRL1_BL-
E_SHIFT),
kCLOCK_Modem = CLK_GATE_DEFINE(AHB_CLK_CTRL1, SYSCON_AHBCLKCTRL1_-
MODEM_MASTER_SHIFT),
kCLOCK_Aes = CLK_GATE_DEFINE(AHB_CLK_CTRL1, SYSCON_AHBCLKCTRL1_AES-
_SHIFT),
kCLOCK_Rfp = CLK_GATE_DEFINE(AHB_CLK_CTRL1, SYSCON_AHBCLKCTRL1_RFP-
_SHIFT),
kCLOCK_DMic = CLK_GATE_DEFINE(AHB_CLK_CTRL1, SYSCON_AHBCLKCTRL1_D-
MIC_SHIFT),
kCLOCK_Sha0 = CLK_GATE_DEFINE(AHB_CLK_CTRL1, SYSCON_AHBCLKCTRL1_HA-
```

```

SH_SHIFT),
kCLOCK_Timer0 = CLK_GATE_DEFINE(ASYNC_CLK_CTRL0, 1),
kCLOCK_Timer1 = CLK_GATE_DEFINE(ASYNC_CLK_CTRL0, 2),
kCLOCK_MainClk = (1 << 16),
kCLOCK_CoreSysClk,
kCLOCK_BusClk,
kCLOCK_Xtal32k,
kCLOCK_Xtal32M,
kCLOCK_Fro32k,
kCLOCK_Fro1M,
kCLOCK_Fro12M,
kCLOCK_Fro32M,
kCLOCK_Fro48M,
kCLOCK_Fro64M,
kCLOCK_ExtClk,
kCLOCK_WdtClk,
kCLOCK_Frg,
kCLOCK_ClkOut,
kCLOCK_Fmeas,
kCLOCK_Sha = CLK_GATE_DEFINE(AHB_CLK_CTRL1, SYSCON_AHBCLKCTRL1_HAS-
H_SHIFT) }

```

*Clock name definition.*

- enum `clock_sel_ofst_t` {
 

```

CM_MAINCLKSEL = REG_OFST(SYSCON, MAINCLKSEL),
CM_OSC32CLKSEL = REG_OFST(SYSCON, OSC32CLKSEL),
CM_CLKOUTCLKSEL = REG_OFST(SYSCON, CLKOUTSEL),
CM_SPIFICLKSEL = REG_OFST(SYSCON, SPIFICLKSEL),
CM_ADCCLKSEL = REG_OFST(SYSCON, ADCCLKSEL),
CM_USARTCLKSEL = REG_OFST(SYSCON, USARTCLKSEL),
CM_I2CCLKSEL = REG_OFST(SYSCON, I2CCLKSEL),
CM_SPICLKSEL = REG_OFST(SYSCON, SPICLKSEL),
CM_IRCLKSEL = REG_OFST(SYSCON, IRCLKSEL),
CM_PWMCLKSEL = REG_OFST(SYSCON, PWMCLKSEL),
CM_WDTCLKSEL = REG_OFST(SYSCON, WDTCLKSEL),
CM_MODEMCLKSEL = REG_OFST(SYSCON, MODEMCLKSEL),
CM_FRGCLKSEL = REG_OFST(SYSCON, FRGCLKSEL),
CM_DMICLKSEL = REG_OFST(SYSCON, DMICCLKSEL),
CM_WKTCLKSEL = REG_OFST(SYSCON, WKTCLKSEL) }

```

*Clock source selector definition.*

- enum `clock_attach_id_t` {

## Function groups

kFRO12M\_to\_MAIN\_CLK = MUX\_A(CM\_MAINCLKSEL, 0),  
kOSC32K\_to\_MAIN\_CLK = MUX\_A(CM\_MAINCLKSEL, 1),  
kXTAL32M\_to\_MAIN\_CLK = MUX\_A(CM\_MAINCLKSEL, 2),  
kFRO32M\_to\_MAIN\_CLK = MUX\_A(CM\_MAINCLKSEL, 3),  
kFRO48M\_to\_MAIN\_CLK = MUX\_A(CM\_MAINCLKSEL, 4),  
kEXT\_CLK\_to\_MAIN\_CLK = MUX\_A(CM\_MAINCLKSEL, 5),  
kFROM1M\_to\_MAIN\_CLK = MUX\_A(CM\_MAINCLKSEL, 6),  
kFRO32M\_to\_OSC32M\_CLK = MUX\_A(CM\_OSC32CLKSEL, 0),  
kXTAL32M\_to\_OSC32M\_CLK = MUX\_A(CM\_OSC32CLKSEL, 1),  
kFRO32K\_to\_OSC32K\_CLK = MUX\_A(CM\_OSC32CLKSEL, 2),  
kXTAL32K\_to\_OSC32K\_CLK = MUX\_A(CM\_OSC32CLKSEL, 3),  
kMAIN\_CLK\_to\_CLKOUT = MUX\_A(CM\_CLKOUTCLKSEL, 0),  
kXTAL32K\_to\_CLKOUT = MUX\_A(CM\_CLKOUTCLKSEL, 1),  
kFRO32K\_to\_CLKOUT = MUX\_A(CM\_CLKOUTCLKSEL, 2),  
kXTAL32M\_to\_CLKOUT = MUX\_A(CM\_CLKOUTCLKSEL, 3),  
kDCDC\_to\_CLKOUT = MUX\_A(CM\_CLKOUTCLKSEL, 4),  
kFRO48M\_to\_CLKOUT = MUX\_A(CM\_CLKOUTCLKSEL, 5),  
kFRO1M\_to\_CLKOUT = MUX\_A(CM\_CLKOUTCLKSEL, 6),  
kNONE\_to\_CLKOUT = MUX\_A(CM\_CLKOUTCLKSEL, 7),  
kMAIN\_CLK\_to\_SPIFI = MUX\_A(CM\_SPIFICLKSEL, 0),  
kXTAL32M\_to\_SPIFI = MUX\_A(CM\_SPIFICLKSEL, 1),  
kFRO64M\_to\_SPIFI = MUX\_A(CM\_SPIFICLKSEL, 2),  
kFRO48M\_to\_SPIFI = MUX\_A(CM\_SPIFICLKSEL, 3),  
kXTAL32M\_to\_ADC\_CLK = MUX\_A(CM\_ADCCLKSEL, 0),  
kFRO12M\_to\_ADC\_CLK = MUX\_A(CM\_ADCCLKSEL, 1),  
kNONE\_to\_ADC\_CLK = MUX\_A(CM\_ADCCLKSEL, 2),  
kOSC32M\_to\_USART\_CLK = MUX\_A(CM\_USARTCLKSEL, 0),  
kFRO48M\_to\_USART\_CLK = MUX\_A(CM\_USARTCLKSEL, 1),  
kFRG\_CLK\_to\_USART\_CLK = MUX\_A(CM\_USARTCLKSEL, 2),  
kNONE\_to\_USART\_CLK = MUX\_A(CM\_USARTCLKSEL, 3),  
kOSC32M\_to\_I2C\_CLK = MUX\_A(CM\_I2CCLKSEL, 0),  
kFRO48M\_to\_I2C\_CLK = MUX\_A(CM\_I2CCLKSEL, 1),  
kNONE\_to\_I2C\_CLK = MUX\_A(CM\_I2CCLKSEL, 2),  
kOSC32M\_to\_SPI\_CLK = MUX\_A(CM\_SPICLKSEL, 0),  
kFRO48M\_to\_SPI\_CLK = MUX\_A(CM\_SPICLKSEL, 1),  
kNONE\_to\_SPI\_CLK = MUX\_A(CM\_SPICLKSEL, 2),  
kOSC32M\_to\_IR\_CLK = MUX\_A(CM\_IRCLKSEL, 0),  
kFRO48M\_to\_IR\_CLK = MUX\_A(CM\_IRCLKSEL, 1),  
kNONE\_to\_IR\_CLK = MUX\_A(CM\_IRCLKSEL, 2),  
kOSC32M\_to\_PWM\_CLK = MUX\_A(CM\_PWMCLKSEL, 0),  
kFRO48M\_to\_PWM\_CLK = MUX\_A(CM\_PWMCLKSEL, 1),  
kNONE\_to\_PWM\_CLK = MUX\_A(CM\_PWMCLKSEL, 2),  
kOSC32M\_to\_WDT\_CLK = MUX\_A(CM\_WDTCLKSEL, 0),  
kOSC32K\_to\_WDT\_CLK = MUX\_A(CM\_WDTCLKSEL, 1),  
kFRO1M\_to\_WDT\_CLK = MUX\_A(CM\_WDTCLKSEL, 2),  
kMAIN\_CLK\_to\_FRG\_CLK = MUX\_A(CM\_FRGCLKSEL, 0),  
kOSC32M\_to\_FRG\_CLK = MUX\_A(CM\_FRGCLKSEL, 1),  
kFRO48M\_to\_FRG\_CLK = MUX\_A(CM\_FRGCLKSEL, 2),  
kNONE\_to\_FRG\_CLK = MUX\_A(CM\_FRGCLKSEL, 3),

```
kFRO48M_to_ASYNC_APB = MUX_A(CM_ASYNCAPB, 3) }
```

*Clock attach definition.*

- enum `clock_div_name_t` { ,  
`kCLOCK_DivSystickClk` = (offsetof(SYSCON\_Type, SYSTICKCLKDIV) / sizeof(uint32\_t)),  
`kCLOCK_DivTraceClk` = (offsetof(SYSCON\_Type, TRACECLKDIV) / sizeof(uint32\_t)),  
`kCLOCK_DivWdtClk` = (offsetof(SYSCON\_Type, WDTCLKDIV) / sizeof(uint32\_t)),  
`kCLOCK_DivIrClk` = (offsetof(SYSCON\_Type, IRCLKDIV) / sizeof(uint32\_t)),  
`kCLOCK_DivAhbClk` = (offsetof(SYSCON\_Type, AHBCLKDIV) / sizeof(uint32\_t)),  
`kCLOCK_DivClkout` = (offsetof(SYSCON\_Type, CLKOUTDIV) / sizeof(uint32\_t)),  
`kCLOCK_DivSpifiClk` = (offsetof(SYSCON\_Type, SPIFICLKDIV) / sizeof(uint32\_t)),  
`kCLOCK_DivAdcClk` = (offsetof(SYSCON\_Type, ADCCLKDIV) / sizeof(uint32\_t)),  
`kCLOCK_DivRtcClk` = (offsetof(SYSCON\_Type, RTCCLKDIV) / sizeof(uint32\_t)),  
`kCLOCK_DivDmicClk` = (offsetof(SYSCON\_Type, DMICCLKDIV) / sizeof(uint32\_t)),  
`kCLOCK_DivRtc1HzClk` = (offsetof(SYSCON\_Type, RTC1HZCLKDIV) / sizeof(uint32\_t)),  
`kCLOCK_DivFrg` = (offsetof(SYSCON\_Type, FRGCTRL) / sizeof(uint32\_t)) }

*Clock divider definition.*

- enum `main_clock_src_t` {  
`kCLOCK_MainFro12M` = 0,  
`kCLOCK_MainOsc32k` = 1,  
`kCLOCK_MainXtal32M` = 2,  
`kCLOCK_MainFro32M` = 3,  
`kCLOCK_MainFro48M` = 4,  
`kCLOCK_MainExtClk` = 5,  
`kCLOCK_MainFro1M` = 6 }

*Clock source selections for the Main Clock.*

- enum `clkout_clock_src_t` {  
`kCLOCK_ClkoutMainClk` = 0,  
`kCLOCK_ClkoutXtal32k` = 1,  
`kCLOCK_ClkoutFro32k` = 2,  
`kCLOCK_ClkoutXtal32M` = 3,  
`kCLOCK_ClkoutDcDcTest` = 4,  
`kCLOCK_ClkoutFro48M` = 5,  
`kCLOCK_ClkoutFro1M` = 6,  
`kCLOCK_ClkoutNoClock` = 7 }

*Clock source selections for CLKOUT.*

- enum `wdt_clock_src_t` {  
`kCLOCK_WdtOsc32MClk` = 0,  
`kCLOCK_WdtOsc32kClk` = 1,  
`kCLOCK_WdtFro1M` = 2,  
`kCLOCK_WdtNoClock` = 3 }

*Clock source definition for Watchdog timer.*

- enum `frg_clock_src_t` {  
`kCLOCK_FrgMainClk` = 0,  
`kCLOCK_FrgOsc32MClk` = 1,  
`kCLOCK_FrgFro48M` = 2,  
`kCLOCK_FrgNoClock` = 3 }

## Function groups

- Clock source definition for fractional divider.*
  - enum `apb_clock_src_t` {  
    `kCLOCK_ApbMainClk` = 0,  
    `kCLOCK_ApbXtal32M` = 1,  
    `kCLOCK_ApbFro32M` = 2,  
    `kCLOCK_ApbFro48M` = 3 }
- Clock source definition for the APB.*
  - enum `fmeas_clock_src_t` {  
    `kCLOCK_fmeasClkIn` = 0,  
    `kCLOCK_fmeasXtal32Mhz` = 1,  
    `kCLOCK_fmeasFRO1Mhz` = 2,  
    `kCLOCK_fmeasXtal32kHz` = 3,  
    `kCLOCK_fmeasMainClock` = 4,  
    `kCLOCK_fmeasGPIO_0_4` = 5,  
    `kCLOCK_fmeasGPIO_0_20` = 6,  
    `kCLOCK_fmeasGPIO_0_16` = 7,  
    `kCLOCK_fmeasGPIO_0_15` = 8 }
- Clock source definition for frequency measure.*
  - enum `spifi_clock_src_t` {  
    `kCLOCK_SpifiMainClk` = 0,  
    `kCLOCK_SpifiXtal32M` = 1,  
    `kCLOCK_SpifiFro64M` = 2,  
    `kCLOCK_SpifiFro48M` = 3,  
    `kCLOCK_SpifiNoClock` = 4 }
- Clock source selection for SPIFI.*
  - enum `adc_clock_src_t` {  
    `kCLOCK_AdcXtal32M` = 0,  
    `kCLOCK_AdcFro12M` = 1,  
    `kCLOCK_AdcNoClock` = 2 }
- Clock definition for ADC.*
  - enum `pwm_clock_source_t` {  
    `kCLOCK_PWM Osc32Mclk` = 0x0,  
    `kCLOCK_PWMFro48Mclk` = 0x1,  
    `kCLOCK_PWMNoClkSel` = 0x2,  
    `kCLOCK_PWMTestClk` = 0x3 }
- PWM Clock source selection values.*
  - enum `Fro_ClkSel_t` {  
    `FRO12M_ENA` = (1 << 0),  
    `FRO32M_ENA` = (1 << 1),  
    `FRO48M_ENA` = (1 << 2),  
    `FRO64M_ENA` = (1 << 3),  
    `FRO96M_ENA` = (1 << 4) }
- FRO clock selection values.*

## Functions

- uint32\_t `CLOCK_GetFreq` (`clock_name_t` clock)



- *Obtains frequency of specified clock.*
- void **CLOCK\_AttachClk** (clock\_attach\_id\_t connection)
  - Selects clock source using <name>SEL register in syscon.*
- void **CLOCK\_SetClkDiv** (clock\_div\_name\_t div\_name, uint32\_t divided\_by\_value, bool reset)
  - Selects clock divider using <name>DIV register in syscon.*
- void **CLOCK\_EnableClock** (clock\_ip\_name\_t clk)
  - Enables specific AHB clock channel.*
- void **CLOCK\_DisableClock** (clock\_ip\_name\_t clk)
  - Disables specific AHB clock channel.*
- bool **CLOCK\_IsClockEnable** (clock\_ip\_name\_t clk)
  - Check if clock is enabled.*
- uint32\_t **CLOCK\_GetApbClkFreq** (void)
  - Obtains frequency of APB Bus clock.*
- uint32\_t **CLOCK\_GetSpifiClkFreq** (void)
  - Return Frequency of Spifi Clock.*
- void **CLOCK\_uDelay** (uint32\_t delayUs)
  - Delay execution by busy waiting.*
- void **CLOCK\_XtalBasicTrim** (void)
  - Sets default trim values for 32MHz XTAL.*
- void **CLOCK\_Xtal32M\_Trim** (int32\_t XO\_32M\_OSC\_CAP\_Delta\_x1000, const **ClockCapacitanceCompensation\_t** \*capa\_charac)
  - Sets board-specific trim values for 32MHz XTAL.*
- void **CLOCK\_Xtal32k\_Trim** (int32\_t XO\_32k\_OSC\_CAP\_Delta\_x1000, const **ClockCapacitanceCompensation\_t** \*capa\_charac)
  - Sets board-specific trim values for 32kHz XTAL.*
- void **CLOCK\_SetXtal32M\_LDO** (void)
  - Enables and sets LDO for 32MHz XTAL.*
- void **CLOCK\_Xtal32M\_WaitUntilStable** (uint32\_t u32AdditionalWait\_us)
  - Waits for 32MHz XTAL to stabilise.*

## Driver version

- #define **FSL\_CLOCK\_DRIVER\_VERSION** (MAKE\_VERSION(2, 1, 0))
  - CLOCK driver version 2.1.0.*

## 7.3 Data Structure Documentation

### 7.3.1 struct ClockCapacitanceCompensation\_t

Capacitances are expressed in hundreds of pF

## 7.4 Macro Definition Documentation

### 7.4.1 #define FSL\_CLOCK\_DRIVER\_VERSION (MAKE\_VERSION(2, 1, 0))

### 7.4.2 #define FLEXCOMM\_CLOCKS

Value:

## Macro Definition Documentation

```
{
 \ kCLOCK_Usart0, kCLOCK_Usart1,
 kCLOCK_I2c0, kCLOCK_I2c1, kCLOCK_Spi0,
 kCLOCK_Spi1, kCLOCK_I2c2 \
}
```

### 7.4.3 #define CTIMER\_CLOCKS

**Value:**

```
{
 \ kCLOCK_Timer0, kCLOCK_Timer1 \
}
```

### 7.4.4 #define GINT\_CLOCKS

**Value:**

```
{
 \ kCLOCK_Gint \
}
```

### 7.4.5 #define WWDT\_CLOCKS

**Value:**

```
{
 \ kCLOCK_WdtOsc \
}
```

### 7.4.6 #define DMIC\_CLOCKS

**Value:**

```
{
 \ kCLOCK_DMic \
}
```

### 7.4.7 #define ADC\_CLOCKS

Value:

```
{
 \
 kCLOCK_Adc0 \
}
```

### 7.4.8 #define SPIFI\_CLOCKS

Value:

```
{
 \
 kCLOCK_Spifi \
}
```

### 7.4.9 #define GPIO\_CLOCKS

Value:

```
{
 \
 kCLOCK_Gpio0 \
}
```

### 7.4.10 #define DMA\_CLOCKS

Value:

```
{
 \
 kCLOCK_Dma \
}
```

## 7.5 Enumeration Type Documentation

### 7.5.1 enum CHIP\_SYSCON\_MAINCLKSRC\_T

Enumerator

***SYSCON\_MAINCLKSRC\_FRO12M*** FRO 12MHz.  
***SYSCON\_MAINCLKSRC\_OSC32K*** OSC 32kHz.  
***SYSCON\_MAINCLKSRC\_XTAL32M*** XTAL 32MHz.  
***SYSCON\_MAINCLKSRC\_FRO32M*** FRO 32MHz.  
***SYSCON\_MAINCLKSRC\_FRO48M*** FRO 48MHz.  
***SYSCON\_MAINCLKSRC\_EXT*** External clock.  
***SYSCON\_MAINCLKSRC\_FRO1M*** FRO 1MHz.

### 7.5.2 enum CHIP\_SYSCON\_FRGCLKSRC\_T

Enumerator

***SYSCON\_FRGCLKSRC\_MAINCLK*** Main Clock.  
***SYSCON\_FRGCLKSRC\_OSC32M*** 32MHz Clock (XTAL or FRO)  
***SYSCON\_FRGCLKSRC\_FRO48MHZ*** FRO 48-MHz.  
***SYSCON\_FRGCLKSRC\_NONE*** FRO 48-MHz.

### 7.5.3 enum clock\_name\_t

Enumerator

***kCLOCK\_Rom*** ROM clock.  
***kCLOCK\_Sram0*** SRAM0 clock.  
***kCLOCK\_Sram1*** SRAM1 clock.  
***kCLOCK\_Flash*** Flash clock.  
***kCLOCK\_Spifi*** SPIFI clock.  
***kCLOCK\_InputMux*** InputMux clock.  
***kCLOCK\_Iocon*** IOCON clock.  
***kCLOCK\_Gpio0*** GPIO0 clock.  
***kCLOCK\_Pint*** PINT clock.  
***kCLOCK\_Dma*** DMA clock.  
***kCLOCK\_Iso7816*** ISO7816 clock.  
***kCLOCK\_WdtOsc*** WDTOSC clock.  
***kCLOCK\_Rtc*** RTC clock.  
***kCLOCK\_AnaInt*** Analog Interrupt Control module clock.  
***kCLOCK\_WakeTmr*** Wake up Timers clock.  
***kCLOCK\_Adc0*** ADC0 clock.  
***kCLOCK\_Efuse*** EFuse clock.  
***kCLOCK\_FlexComm0*** FlexComm0 clock.  
***kCLOCK\_FlexComm1*** FlexComm1 clock.  
***kCLOCK\_FlexComm2*** FlexComm2 clock.  
***kCLOCK\_FlexComm3*** FlexComm3 clock.  
***kCLOCK\_FlexComm4*** FlexComm4 clock.  
***kCLOCK\_FlexComm5*** FlexComm5 clock.  
***kCLOCK\_Ir*** Infra Red clock.  
***kCLOCK\_Pwm*** PWM clock.  
***kCLOCK\_Rng*** RNG clock.  
***kCLOCK\_FlexComm6*** FlexComm6 clock.  
***kCLOCK\_Usart0*** USART0 clock.  
***kCLOCK\_Usart1*** USART1 clock.  
***kCLOCK\_I2c0*** I2C0 clock.  
***kCLOCK\_I2c1*** I2C1 clock.

***kCLOCK\_Spi0*** SPI0 clock.  
***kCLOCK\_Spi1*** SPI1 clock.  
***kCLOCK\_I2c2*** I2C2 clock.  
***kCLOCK\_BLE*** BLE clock.  
***kCLOCK\_Modem*** MODEM clock.  
***kCLOCK\_Aes*** AES clock.  
***kCLOCK\_Rfp*** RFP clock.  
***kCLOCK\_DMic*** DMIC clock.  
***kCLOCK\_Sha0*** SHA0 clock.  
***kCLOCK\_Timer0*** Timer0 clock.  
***kCLOCK\_Timer1*** Timer1 clock.  
***kCLOCK\_MainClk*** MAIN\_CLK.  
***kCLOCK\_CoreSysClk*** Core/system clock.  
***kCLOCK\_BusClk*** AHB bus clock.  
***kCLOCK\_Xtal32k*** 32kHz crystal oscillator  
***kCLOCK\_Xtal32M*** 32MHz crystal oscillator  
***kCLOCK\_Fro32k*** 32kHz free running oscillator  
***kCLOCK\_Fro1M*** 1MHz Free Running Oscillator  
***kCLOCK\_Fro12M*** 12MHz Free Running Oscillator  
***kCLOCK\_Fro32M*** 32MHz Free Running Oscillator  
***kCLOCK\_Fro48M*** 48MHz Free Running Oscillator  
***kCLOCK\_Fro64M*** 64Mhz Free Running Oscillator  
***kCLOCK\_ExtClk*** External clock.  
***kCLOCK\_WdtClk*** Watchdog clock.  
***kCLOCK\_Frg*** Fractional divider.  
***kCLOCK\_ClkOut*** Clock out.  
***kCLOCK\_Fmeas*** FMEAS clock.  
***kCLOCK\_Sha*** Hash clock.

#### 7.5.4 enum clock\_sel\_ofst\_t

Enumerator

***CM\_MAINCLKSEL*** Clock source selector of Main clock source.  
***CM\_OSC32CLKSEL*** Clock source selector of OSC32KCLK and OSC32MCLK.  
***CM\_CLKOUTCLKSEL*** Clock source selector of CLKOUT.  
***CM\_SPIFICLKSEL*** Clock source selector of SPIFI.  
***CM\_ADCCLKSEL*** Clock source selector of ADC.  
***CM\_USARTCLKSEL*** Clock source selector of USART0 & 1.  
***CM\_I2CCLKSEL*** Clock source selector of I2C0, 1 and 2.  
***CM\_SPICLKSEL*** Clock source selector of SPI0 & 1.  
***CM\_IRCLKSEL*** Clock source selector of Infra Red.  
***CM\_PWMCLKSEL*** Clock source selector of PWM.  
***CM\_WDTCLKSEL*** Clock source selector of Watchdog Timer.

## Enumeration Type Documentation

***CM\_MODEMCLKSEL*** Clock source selector of Modem.  
***CM\_FRGCLKSEL*** Clock source selector of Fractional Rate Generator (FRG)  
***CM\_DMICLKSEL*** Clock source selector of Digital microphone (DMIC)  
***CM\_WKTCLKSEL*** Clock source selector of Wake-up Timer.

### 7.5.5 enum clock\_attach\_id\_t

Enumerator

***kFRO12M\_to\_MAIN\_CLK*** Select FRO 12M for main clock.  
***kOSC32K\_to\_MAIN\_CLK*** Select OSC 32K for main clock.  
***kXTAL32M\_to\_MAIN\_CLK*** Select XTAL 32M for main clock.  
***kFRO32M\_to\_MAIN\_CLK*** Select FRO 32M for main clock.  
***kFRO48M\_to\_MAIN\_CLK*** Select FRO 48M for main clock.  
***kEXT\_CLK\_to\_MAIN\_CLK*** Select external clock for main clock.  
***kFROM1M\_to\_MAIN\_CLK*** Select FRO 1M for main clock.  
***kFRO32M\_to\_OSC32M\_CLK*** Select FRO 32M for OSC32KCLK and OSC32MCLK.  
***kXTAL32M\_to\_OSC32M\_CLK*** Select XTAL 32M for OSC32KCLK and OSC32MCLK.  
***kFRO32K\_to\_OSC32K\_CLK*** Select FRO 32K for OSC32KCLK and OSC32MCLK.  
***kXTAL32K\_to\_OSC32K\_CLK*** Select XTAL 32K for OSC32KCLK and OSC32MCLK.  
***kMAIN\_CLK\_to\_CLKOUT*** Select main clock for CLKOUT.  
***kXTAL32K\_to\_CLKOUT*** Select XTAL 32K for CLKOUT.  
***kFRO32K\_to\_CLKOUT*** Select FRO 32K for CLKOUT.  
***kXTAL32M\_to\_CLKOUT*** Select XTAL 32M for CLKOUT.  
***kDCDC\_to\_CLKOUT*** Select DCDC for CLKOUT.  
***kFRO48M\_to\_CLKOUT*** Select FRO 48M for CLKOUT.  
***kFRO1M\_to\_CLKOUT*** Select FRO 1M for CLKOUT.  
***kNONE\_to\_CLKOUT*** No clock for CLKOUT.  
***kMAIN\_CLK\_to\_SPIFI*** Select main clock for SPIFI.  
***kXTAL32M\_to\_SPIFI*** Select XTAL 32M for SPIFI.  
***kFRO64M\_to\_SPIFI*** Select FRO 64M for SPIFI.  
***kFRO48M\_to\_SPIFI*** Select FRO 48M for SPIFI.  
***kXTAL32M\_to\_ADC\_CLK*** Select XTAL 32M for ADC.  
***kFRO12M\_to\_ADC\_CLK*** Select FRO 12M for ADC.  
***kNONE\_to\_ADC\_CLK*** No clock for ADC.  
***kOSC32M\_to\_USART\_CLK*** Select OSC 32M for USART0 & 1.  
***kFRO48M\_to\_USART\_CLK*** Select FRO 48M for USART0 & 1.  
***kFRG\_CLK\_to\_USART\_CLK*** Select FRG clock for USART0 & 1.  
***kNONE\_to\_USART\_CLK*** No clock for USART0 & 1.  
***kOSC32M\_to\_I2C\_CLK*** Select OSC 32M for I2C0, 1 and 2.  
***kFRO48M\_to\_I2C\_CLK*** Select FRO 48M for I2C0, 1 and 2.  
***kNONE\_to\_I2C\_CLK*** No clock for I2C0, 1 and 2.  
***kOSC32M\_to\_SPI\_CLK*** Select OSC 32M for SPI0 & 1.  
***kFRO48M\_to\_SPI\_CLK*** Select FRO 48M for SPI0 & 1.

*kNONE\_to\_SPI\_CLK* No clock for SPI0 & 1.  
*kOSC32M\_to\_IR\_CLK* Select OSC 32M for Infra Red.  
*kFRO48M\_to\_IR\_CLK* Select FRO 48M for Infra Red.  
*kNONE\_to\_IR\_CLK* No clock for Infra Red.  
*kOSC32M\_to\_PWM\_CLK* Select OSC 32M for PWM.  
*kFRO48M\_to\_PWM\_CLK* Select FRO 48M for PWM.  
*kNONE\_to\_PWM\_CLK* No clock for PWM.  
*kOSC32M\_to\_WDT\_CLK* Select OSC 32M for Watchdog Timer.  
*kOSC32K\_to\_WDT\_CLK* Select FRO 32K for Watchdog Timer.  
*kFRO1M\_to\_WDT\_CLK* Select FRO 1M for Watchdog Timer.  
*kMAIN\_CLK\_to\_FRG\_CLK* Select main clock for FRG.  
*kOSC32M\_to\_FRG\_CLK* Select OSC 32M for FRG.  
*kFRO48M\_to\_FRG\_CLK* Select FRO 48M for FRG.  
*kNONE\_to\_FRG\_CLK* No clock for FRG.  
*kMAIN\_CLK\_to\_DMI\_CLK* Select main clock for DMIC.  
*kOSC32K\_to\_DMI\_CLK* Select OSC 32K for DMIC.  
*kFRO48M\_to\_DMI\_CLK* Select FRO 48M for DMIC.  
*kMCLK\_to\_DMI\_CLK* Select external clock for DMIC.  
*kFRO1M\_to\_DMI\_CLK* Select FRO 1M for DMIC.  
*kFRO12M\_to\_DMI\_CLK* Select FRO 12M for DMIC.  
*kNONE\_to\_DMI\_CLK* No clock for DMIC.  
*kOSC32K\_to\_WKT\_CLK* Select OSC 32K for for WKT.  
*kNONE\_to\_WKT\_CLK* No clock for WKT.  
*kXTAL32M\_to\_BLE\_CLK* Select XTAL 32M for BLE.  
*kNONE\_to\_BLE\_CLK* No clock for BLE.  
*kMAIN\_CLK\_to\_ASYNC\_APB* Select main clock for Asynchronous APB.  
*kXTAL32M\_to\_ASYNC\_APB* Select XTAL 32M for Asynchronous APB.  
*kFRO32M\_to\_ASYNC\_APB* Select FRO 32M for Asynchronous APB.  
*kFRO48M\_to\_ASYNC\_APB* Select FRO 48M for Asynchronous APB.

## 7.5.6 enum clock\_div\_name\_t

Enumerator

*kCLOCK\_DivSystickClk* SYSTICK clock divider.  
*kCLOCK\_DivTraceClk* TRACE clock divider.  
*kCLOCK\_DivWdtClk* Watchdog Timer clock divider.  
*kCLOCK\_DivIrClk* Infra Red clock divider.  
*kCLOCK\_DivAhbClk* System clock divider.  
*kCLOCK\_DivClkout* CLKOUT clock divider.  
*kCLOCK\_DivSpifiClk* SPIFI clock divider.  
*kCLOCK\_DivAdcClk* ADC clock divider.  
*kCLOCK\_DivRtcClk* Real Time Clock divider.  
*kCLOCK\_DivDmicClk* DMIC clock divider.

## Enumeration Type Documentation

*kCLOCK\_DivRtc1HzClk* Real Time Clock divider.

*kCLOCK\_DivFrg* FRG Clock divider.

### 7.5.7 enum main\_clock\_src\_t

Enumerator

*kCLOCK\_MainFro12M* FRO 12M for main clock.

*kCLOCK\_MainOsc32k* OSC 32K for main clock.

*kCLOCK\_MainXtal32M* XTAL 32M for main clock.

*kCLOCK\_MainFro32M* FRO 32M for main clock.

*kCLOCK\_MainFro48M* FRO 48M for main clock.

*kCLOCK\_MainExtClk* External clock for main clock.

*kCLOCK\_MainFro1M* FRO 1M for main clock.

### 7.5.8 enum clkout\_clock\_src\_t

Enumerator

*kCLOCK\_ClkoutMainClk* CPU & System Bus clock for CLKOUT.

*kCLOCK\_ClkoutXtal32k* XTAL 32K for CLKOUT.

*kCLOCK\_ClkoutFro32k* FRO 32K for CLKOUT.

*kCLOCK\_ClkoutXtal32M* XTAL 32M for CLKOUT.

*kCLOCK\_ClkoutDcDcTest* DCDC Test for CLKOUT.

*kCLOCK\_ClkoutFro48M* FRO 48M for CLKOUT.

*kCLOCK\_ClkoutFro1M* FRO 1M for CLKOUT.

*kCLOCK\_ClkoutNoClock* No clock for CLKOUT.

### 7.5.9 enum wdt\_clock\_src\_t

Enumerator

*kCLOCK\_WdtOsc32MClk* OSC 32M for WDT.

*kCLOCK\_WdtOsc32kClk* OSC 32K for WDT.

*kCLOCK\_WdtFro1M* FRO 1M for WDT.

*kCLOCK\_WdtNoClock* No clock for WDT.

### 7.5.10 enum frg\_clock\_src\_t

Enumerator

*kCLOCK\_FrgMainClk* CPU & System Bus clock for FRG.



***kCLOCK\_FrgOsc32MClk*** OSC 32M clock for FRG.  
***kCLOCK\_FrgFro48M*** FRO 48M for FRG.  
***kCLOCK\_FrgNoClock*** No clock for FRG.

### 7.5.11 enum apb\_clock\_src\_t

Enumerator

***kCLOCK\_ApbMainClk*** CPU & System Bus clock for APB bridge.  
***kCLOCK\_ApbXtal32M*** XTAL 32M for APB bridge.  
***kCLOCK\_ApbFro32M*** FRO 32M for APB bridge.  
***kCLOCK\_ApbFro48M*** FRO 48M for APB bridge.

### 7.5.12 enum fmeas\_clock\_src\_t

Enumerator

***kCLOCK\_fmeasClkIn*** Clock in for FMEAS.  
***kCLOCK\_fmeasXtal32Mhz*** XTAL 32M for FMEAS.  
***kCLOCK\_fmeasFRO1Mhz*** FRO 1M for FMEAS.  
***kCLOCK\_fmeasXtal32kHz*** XTAL 32K for FMEAS.  
***kCLOCK\_fmeasMainClock*** CPU & System Bus clock for FMEAS.  
***kCLOCK\_fmeasGPIO\_0\_4*** GPIO0\_4 input for FMEAS.  
***kCLOCK\_fmeasGPIO\_0\_20*** GPIO0\_20 input for FMEAS.  
***kCLOCK\_fmeasGPIO\_0\_16*** GPIO0\_16 input for FMEAS.  
***kCLOCK\_fmeasGPIO\_0\_15*** GPIO0\_15 input for FMEAS.

### 7.5.13 enum spifi\_clock\_src\_t

Enumerator

***kCLOCK\_SpifiMainClk*** CPU & System Bus clock for SPIFI.  
***kCLOCK\_SpifiXtal32M*** XTAL 32M for SPIFI.  
***kCLOCK\_SpifiFro64M*** FRO 64M for SPIFI.  
***kCLOCK\_SpifiFro48M*** FRO 48M for SPIFI.  
***kCLOCK\_SpifiNoClock*** No clock for SPIFI.

## Function Documentation

### 7.5.14 enum adc\_clock\_src\_t

Enumerator

*kCLOCK\_AdcXtal32M* XTAL 32MHz for ADC.  
*kCLOCK\_AdcFro12M* FRO 12MHz for ADC.  
*kCLOCK\_AdcNoClock* No clock for ADC.

### 7.5.15 enum pwm\_clock\_source\_t

Enumerator

*kCLOCK\_PWM Osc32Mclk* 32MHz FRO or XTAL clock  
*kCLOCK\_PWM Fro48Mclk* FRO 48MHz clock.  
*kCLOCK\_PWM NoClkSel* No clock selected - Shutdown functional PWM clock for power saving.  
*kCLOCK\_PWM TestClk* Test clock input - Shutdown functional PWM clock for power saving.

### 7.5.16 enum Fro\_ClkSel\_t

Enumerator

*FRO12M\_ENA* FRO12M.  
*FRO32M\_ENA* FRO32M.  
*FRO48M\_ENA* FRO48M.  
*FRO64M\_ENA* FRO64M.  
*FRO96M\_ENA* FRO96M.

## 7.6 Function Documentation

### 7.6.1 uint32\_t CLOCK\_GetFreq ( clock\_name\_t clock )

Parameters

|                     |                          |
|---------------------|--------------------------|
| <i>clock_name_t</i> | specify clock to be read |
|---------------------|--------------------------|

Returns

uint32\_t frequency

Note

### 7.6.2 void CLOCK\_AttachClk ( clock\_attach\_id\_t connection )

## Parameters

|                          |                       |
|--------------------------|-----------------------|
| <i>clock_attach_id_t</i> | specify clock mapping |
|--------------------------|-----------------------|

## Returns

none

## Note

### 7.6.3 void CLOCK\_SetClkDiv ( clock\_div\_name\_t *div\_name*, uint32\_t *divided\_by\_value*, bool *reset* )

## Parameters

|                         |                                                         |
|-------------------------|---------------------------------------------------------|
| <i>clock_div_name_t</i> | specifies which DIV register we are accessing           |
| <i>uint32_t</i>         | specifies divisor                                       |
| <i>bool</i>             | true if a syscon clock reset should also be carried out |

## Returns

none

## Note

### 7.6.4 void CLOCK\_EnableClock ( clock\_ip\_name\_t *clk* )

## Parameters

---

## Function Documentation

|                        |                                                     |
|------------------------|-----------------------------------------------------|
| <i>clock_ip_name_t</i> | specifies which peripheral clock we are controlling |
|------------------------|-----------------------------------------------------|

Returns

none

Note

clock\_ip\_name\_t is a typedef clone of clock\_name\_t

### 7.6.5 void CLOCK\_DisableClock ( clock\_ip\_name\_t *clk* )

Parameters

|                        |                                                     |
|------------------------|-----------------------------------------------------|
| <i>clock_ip_name_t</i> | specifies which peripheral clock we are controlling |
|------------------------|-----------------------------------------------------|

Returns

none

Note

clock\_ip\_name\_t is a typedef clone of clock\_name\_t

### 7.6.6 bool CLOCK\_IsClockEnable ( clock\_ip\_name\_t *clk* )

Parameters

|                        |                                                     |
|------------------------|-----------------------------------------------------|
| <i>clock_ip_name_t</i> | specifies which peripheral clock we are controlling |
|------------------------|-----------------------------------------------------|

Returns

bool

Note

clock\_ip\_name\_t is a typedef clone of clock\_name\_t

### 7.6.7 uint32\_t CLOCK\_GetApbCLKFreq ( void )

## Parameters

|             |  |
|-------------|--|
| <i>none</i> |  |
|-------------|--|

## Returns

uint32\_t frequency

## Note

### 7.6.8 uint32\_t CLOCK\_GetSpifiClkFreq ( void )

## Returns

Frequency of Spifi.

### 7.6.9 void CLOCK\_uDelay ( uint32\_t *delayUs* )

## Parameters

|                |                                 |
|----------------|---------------------------------|
| <i>delayUs</i> | delay duration in micro seconds |
|----------------|---------------------------------|

## Returns

none

### 7.6.10 void CLOCK\_XtalBasicTrim ( void )

## Parameters

|             |  |
|-------------|--|
| <i>none</i> |  |
|-------------|--|

## Returns

none

## Note

Has no effect if CLOCK\_Xtal32M\_Trim has been called

```
7.6.11 void CLOCK_Xtal32M_Trim (int32_t XO_32M_OSC_CAP_Delta_x1000,
const ClockCapacitanceCompensation_t * capa_charac)
```

## Parameters

|                                    |                                               |
|------------------------------------|-----------------------------------------------|
| <i>XO_32M_OS-C_CAP_Delta_x1000</i> | capacitance correction in fF (femtoFarad)     |
| <i>capa_charac</i>                 | board 32M capacitance characteristics pointer |

## Returns

none

## Note

*capa\_charac* must point to a struct set in board.c using `CLOCK_32MfXtalIecLoadpF` Load capacitance, pF `CLOCK_32MfXtalPPcbParCappF` PCB +ve parasitic capacitance, pF `CLOCK_32MfXtalNPcbParCappF` PCB -ve parasitic capacitance, pF

### 7.6.12 void CLOCK\_Xtal32k\_Trim ( int32\_t XO\_32k\_OSC\_CAP\_Delta\_x1000, const ClockCapacitanceCompensation\_t \* capa\_charac )

## Parameters

|                                    |                                               |
|------------------------------------|-----------------------------------------------|
| <i>XO_32k_OSC-_CAP_Delta_x1000</i> | capacitance correction in fF                  |
| <i>capa_charac</i>                 | board 32k capacitance characteristics pointer |

## Returns

none

## Note

*capa\_charac* must point to a struct set in board.c using `CLOCK_32kfXtalIecLoadpF` Load capacitance, pF `CLOCK_32kfXtalPPcbParCappF` PCB +ve parasitic capacitance, pF `CLOCK_32kfXtalNPcbParCappF` PCB -ve parasitic capacitance, pF

### 7.6.13 void CLOCK\_SetXtal32M\_LDO ( void )

## Function Documentation

### Parameters

|             |  |
|-------------|--|
| <i>none</i> |  |
|-------------|--|

### Returns

*none*

## 7.6.14 void CLOCK\_Xtal32M\_WaitUntilStable ( uint32\_t *u32AdditionalWait\_us* )

### Parameters

|                                   |                                                                          |
|-----------------------------------|--------------------------------------------------------------------------|
| <i>u32Additional-<br/>Wait_us</i> | Additional wait after hardware indicates that stability has been reached |
|-----------------------------------|--------------------------------------------------------------------------|

### Returns

*none*

### Note

Operates as a tight loop. Worst case would be ~600ms



## Chapter 8 Common Driver

### 8.1 Overview

The MCUXpresso SDK provides a driver for the common module of MCUXpresso SDK devices.

#### Macros

- #define [ADC\\_RSTS](#)
- #define [MAKE\\_STATUS](#)(group, code) (((group)\*100) + (code))  
*Construct a status code value from a group and code number.*
- #define [MAKE\\_VERSION](#)(major, minor, bugfix) (((major) << 16) | ((minor) << 8) | (bugfix))  
*Construct the version number for drivers.*
- #define [DEBUG\\_CONSOLE\\_DEVICE\\_TYPE\\_NONE](#) 0U  
*No debug console.*
- #define [DEBUG\\_CONSOLE\\_DEVICE\\_TYPE\\_UART](#) 1U  
*Debug console based on UART.*
- #define [DEBUG\\_CONSOLE\\_DEVICE\\_TYPE\\_LPUART](#) 2U  
*Debug console based on LPUART.*
- #define [DEBUG\\_CONSOLE\\_DEVICE\\_TYPE\\_LPSCI](#) 3U  
*Debug console based on LPSCI.*
- #define [DEBUG\\_CONSOLE\\_DEVICE\\_TYPE\\_USBCDC](#) 4U  
*Debug console based on USBCDC.*
- #define [DEBUG\\_CONSOLE\\_DEVICE\\_TYPE\\_FLEXCOMM](#) 5U  
*Debug console based on FLEXCOMM.*
- #define [DEBUG\\_CONSOLE\\_DEVICE\\_TYPE\\_IUART](#) 6U  
*Debug console based on i.MX UART.*
- #define [DEBUG\\_CONSOLE\\_DEVICE\\_TYPE\\_VUSART](#) 7U  
*Debug console based on LPC\_VUSART.*
- #define [DEBUG\\_CONSOLE\\_DEVICE\\_TYPE\\_MINI\\_USART](#) 8U  
*Debug console based on LPC\_USART.*
- #define [DEBUG\\_CONSOLE\\_DEVICE\\_TYPE\\_SWO](#) 9U  
*Debug console based on SWO.*
- #define [ARRAY\\_SIZE](#)(x) (sizeof(x) / sizeof((x)[0]))  
*Computes the number of elements in an array.*

#### Typedefs

- typedef int32\_t [status\\_t](#)  
*Type used for all status and error return values.*

## Enumerations

- enum `SYSCON_RSTn_t` {
  - `kFLASH_RST_SHIFT_RSTn` = (0 | `SYSCON_PRESETCTRL0_FLASH_RST_SHIFT`),
  - `kSPIFI_RST_SHIFT_RSTn` = (0 | `SYSCON_PRESETCTRL0_SPIFI_RST_SHIFT`),
  - `kMUX_RST_SHIFT_RSTn` = (0 | `SYSCON_PRESETCTRL0_MUX_RST_SHIFT`),
  - `kBLE_TG_RST_SHIFT_RSTn` = (0 | `SYSCON_PRESETCTRL0_BLE_TIMING_GEN_RST_SHIFT`),
  - `kIOCON_RST_SHIFT_RSTn` = (0 | `SYSCON_PRESETCTRL0_IOCON_RST_SHIFT`),
  - `kGPIO0_RST_SHIFT_RSTn` = (0 | `SYSCON_PRESETCTRL0_GPIO_RST_SHIFT`),
  - `kPINT_RST_SHIFT_RSTn` = (0 | `SYSCON_PRESETCTRL0_PINT_RST_SHIFT`),
  - `kGINT_RST_SHIFT_RSTn` = (0 | `SYSCON_PRESETCTRL0_GINT_RST_SHIFT`),
  - `kDMA_RST_SHIFT_RSTn` = (0 | `SYSCON_PRESETCTRL0_DMA_RST_SHIFT`),
  - `kWWDT_RST_SHIFT_RSTn` = (0 | `SYSCON_PRESETCTRL0_WWDT_RST_SHIFT`),
  - `kRTC_RST_SHIFT_RSTn` = (0 | `SYSCON_PRESETCTRL0_RTC_RST_SHIFT`),
  - `kANA_INT_RST_SHIFT_RSTn` = (0 | `SYSCON_PRESETCTRL0_ANA_INT_CTRL_RST_SHIFT`),
  - `kWKT_RST_SHIFT_RSTn` = (0 | `SYSCON_PRESETCTRL0_WAKE_UP_TIMERS_RST_SHIFT`),
  - `kADC0_RST_SHIFT_RSTn` = (0 | `SYSCON_PRESETCTRL0_ADC_RST_SHIFT`),
  - `kEFUSE_RST_SHIFT_RSTn` = (0 | `SYSCON_PRESETCTRL0_EFUSE_RST_SHIFT`),
  - `kFC0_RST_SHIFT_RSTn`,
  - `kFC1_RST_SHIFT_RSTn`,
  - `kFC2_RST_SHIFT_RSTn` = ((1UL << 16) | `SYSCON_PRESETCTRL1_I2C0_RST_SHIFT`),
  - `kFC3_RST_SHIFT_RSTn` = ((1UL << 16) | `SYSCON_PRESETCTRL1_I2C1_RST_SHIFT`),
  - `kFC4_RST_SHIFT_RSTn` = ((1UL << 16) | `SYSCON_PRESETCTRL1_SPI0_RST_SHIFT`),
  - `kFC5_RST_SHIFT_RSTn` = ((1UL << 16) | `SYSCON_PRESETCTRL1_SPI1_RST_SHIFT`),
  - `kIRB_RST_SHIFT_RSTn` = ((1UL << 16) | `SYSCON_PRESETCTRL1_IR_RST_SHIFT`),
  - `kPWM_RST_SHIFT_RSTn` = ((1UL << 16) | `SYSCON_PRESETCTRL1_PWM_RST_SHIFT`),
  - `kRNG_RST_SHIFT_RSTn`,
  - `kFC6_RST_SHIFT_RSTn` = ((1UL << 16) | `SYSCON_PRESETCTRL1_I2C2_RST_SHIFT`),
  - `kUSART0_RST_SHIFT_RSTn` = `kFC0_RST_SHIFT_RSTn`,
  - `kUSART1_RST_SHIFT_RSTn` = `kFC1_RST_SHIFT_RSTn`,
  - `kI2C0_RST_SHIFT_RSTn` = `kFC2_RST_SHIFT_RSTn`,
  - `kI2C1_RST_SHIFT_RSTn` = `kFC3_RST_SHIFT_RSTn`,
  - `kSPI0_RST_SHIFT_RSTn` = `kFC4_RST_SHIFT_RSTn`,
  - `kSPI1_RST_SHIFT_RSTn` = `kFC5_RST_SHIFT_RSTn`,
  - `kI2C2_RST_SHIFT_RSTn` = `kFC6_RST_SHIFT_RSTn`,
  - `kBLE_RST_SHIFT_RSTn` = ((1UL << 16) | `SYSCON_PRESETCTRL1_BLE_RST_SHIFT`),
  - `kMODEM_MASTER_SHIFT_RSTn`,
  - `kAES_RST_SHIFT_RSTn` = ((1UL << 16) | `SYSCON_PRESETCTRL1_AES_RST_SHIFT`),
  - `kRFP_RST_SHIFT_RSTn` = ((1UL << 16) | `SYSCON_PRESETCTRL1_RFP_RST_SHIFT`),
  - `kDMIC_RST_SHIFT_RSTn`,
  - `kHASH_RST_SHIFT_RSTn` = ((1UL << 16) | `SYSCON_PRESETCTRL1_HASH_RST_SHIFT`),
  - `kCTIMER0_RST_SHIFT_RSTn` = ((2UL << 16) | `ASYNC_SYSCON_ASYNCPRESETCTRL_`

```
CT32B0_SHIFT),
kCTIMER1_RST_SHIFT_RSTn = ((2UL << 16) | ASYNC_SYSCON_ASYNCPRESETCTRL_
CT32B1_SHIFT) }
```

*Enumeration for peripheral reset control bits.*

- `enum _status_groups` {

## Overview

```
kStatusGroup_Generic = 0,
kStatusGroup_FLASH = 1,
kStatusGroup_LPSPI = 4,
kStatusGroup_FLEXIO_SPI = 5,
kStatusGroup_DSPI = 6,
kStatusGroup_FLEXIO_UART = 7,
kStatusGroup_FLEXIO_I2C = 8,
kStatusGroup_LPI2C = 9,
kStatusGroup_UART = 10,
kStatusGroup_I2C = 11,
kStatusGroup_LPSCI = 12,
kStatusGroup_LPUART = 13,
kStatusGroup_SPI = 14,
kStatusGroup_XRDC = 15,
kStatusGroup_SEMA42 = 16,
kStatusGroup_SDHC = 17,
kStatusGroup_SDMMC = 18,
kStatusGroup_SAI = 19,
kStatusGroup_MCG = 20,
kStatusGroup_SCG = 21,
kStatusGroup_SDSPI = 22,
kStatusGroup_FLEXIO_I2S = 23,
kStatusGroup_FLEXIO_MCULCD = 24,
kStatusGroup_FLASHIAP = 25,
kStatusGroup_FLEXCOMM_I2C = 26,
kStatusGroup_I2S = 27,
kStatusGroup_IUART = 28,
kStatusGroup_CSI = 29,
kStatusGroup_MIPI_DSI = 30,
kStatusGroup_SDRAMC = 35,
kStatusGroup_POWER = 39,
kStatusGroup_ENET = 40,
kStatusGroup_PHY = 41,
kStatusGroup_TRGMUX = 42,
kStatusGroup_SMARTCARD = 43,
kStatusGroup_LMEM = 44,
kStatusGroup_QSPI = 45,
kStatusGroup_DMA = 50,
kStatusGroup_EDMA = 51,
kStatusGroup_DMAMGR = 52,
kStatusGroup_FLEXCAN = 53,
kStatusGroup_LTC = 54,
kStatusGroup_FLEXIO_CAMERA = 55,
kStatusGroup_LPC_SPI = 56,
kStatusGroup_LPC_USART = 57,
kStatusGroup_DMIC = 58,
kStatusGroup_SDIF = 59,
kStatusGroup_SPIFI = 60,
kStatusGroup_OTP = 61,
```

`kStatusGroup_CODEC = 148 }`

*Status group numbers.*

- enum

*Generic status return codes.*

## Functions

- void `RESET_SetPeripheralReset` (`reset_ip_name_t` peripheral)  
*Assert reset to peripheral.*
- void `RESET_ClearPeripheralReset` (`reset_ip_name_t` peripheral)  
*Clear reset to peripheral.*
- void `RESET_PeripheralReset` (`reset_ip_name_t` peripheral)  
*Reset peripheral module.*
- void `RESET_SystemReset` (void)  
*Reset the chip.*
- static `status_t` `EnableIRQ` (`IRQn_Type` interrupt)  
*Enable specific interrupt.*
- static `status_t` `DisableIRQ` (`IRQn_Type` interrupt)  
*Disable specific interrupt.*
- static `uint32_t` `DisableGlobalIRQ` (void)  
*Disable the global IRQ.*
- static void `EnableGlobalIRQ` (`uint32_t` primask)  
*Enable the global IRQ.*
- void `EnableDeepSleepIRQ` (`IRQn_Type` interrupt)  
*Enable specific interrupt for wake-up from deep-sleep mode.*
- void `DisableDeepSleepIRQ` (`IRQn_Type` interrupt)  
*Disable specific interrupt for wake-up from deep-sleep mode.*
- void \* `SDK_Malloc` (`size_t` size, `size_t` alignbytes)  
*Allocate memory with given alignment and aligned size.*
- void `SDK_Free` (void \*ptr)  
*Free memory.*
- void `SDK_DelayAtLeastUs` (`uint32_t` delay\_us, `uint32_t` coreClock\_Hz)  
*Delay at least for some time.*

## Driver version

- `#define FSL_RESET_DRIVER_VERSION (MAKE_VERSION(2, 0, 1))`  
*RESET driver version 2.0.1.*

## Driver version

- `#define FSL_COMMON_DRIVER_VERSION (MAKE_VERSION(2, 2, 0))`  
*common driver version 2.2.0.*

## Min/max macros

- `#define MIN(a, b) (((a) < (b)) ? (a) : (b))`
- `#define MAX(a, b) (((a) > (b)) ? (a) : (b))`

### UINT16\_MAX/UINT32\_MAX value

- #define **UINT16\_MAX** ((uint16\_t)-1)
- #define **UINT32\_MAX** ((uint32\_t)-1)

### Timer utilities

- #define **USEC\_TO\_COUNT**(us, clockFreqInHz) (uint64\_t)((uint64\_t)(us) \* (clockFreqInHz)) / 1000000U)  
*Macro to convert a microsecond period to raw count value.*
- #define **COUNT\_TO\_USEC**(count, clockFreqInHz) (uint64\_t)((uint64\_t)count \* 1000000U / clockFreqInHz)  
*Macro to convert a raw count value to microsecond.*
- #define **MSEC\_TO\_COUNT**(ms, clockFreqInHz) (uint64\_t)((uint64\_t)ms \* clockFreqInHz / 1000U)  
*Macro to convert a millisecond period to raw count value.*
- #define **COUNT\_TO\_MSEC**(count, clockFreqInHz) (uint64\_t)((uint64\_t)count \* 1000U / clockFreqInHz)  
*Macro to convert a raw count value to millisecond.*

### Alignment variable definition macros

- #define **SDK\_ALIGN**(var, alignbytes) var
- #define **SDK\_SIZEALIGN**(var, alignbytes) ((unsigned int)((var) + ((alignbytes)-1)) & (unsigned int)(~(unsigned int)((alignbytes)-1)))  
*Macro to change a value to a given size aligned value.*

### Non-cacheable region definition macros

- #define **AT\_NONCACHEABLE\_SECTION**(var) var
- #define **AT\_NONCACHEABLE\_SECTION\_ALIGN**(var, alignbytes) var
- #define **AT\_NONCACHEABLE\_SECTION\_INIT**(var) var
- #define **AT\_NONCACHEABLE\_SECTION\_ALIGN\_INIT**(var, alignbytes) var

### Suppress fallthrough warning macro

- #define **SUPPRESS\_FALL\_THROUGH\_WARNING**()

## 8.2 Macro Definition Documentation

### 8.2.1 #define FSL\_RESET\_DRIVER\_VERSION (MAKE\_VERSION(2, 0, 1))

### 8.2.2 #define ADC\_RSTS

Value:

```
{
 \kADC0_RST_SHIFT_RSTn \
} /* Reset bits for ADC peripheral */
```

Array initializers with peripheral reset bits

**8.2.3** `#define MAKE_STATUS( group, code ) (((group)*100) + (code))`

**8.2.4** `#define MAKE_VERSION( major, minor, bugfix ) (((major) << 16) | ((minor) << 8) | (bugfix))`

**8.2.5** `#define FSL_COMMON_DRIVER_VERSION (MAKE_VERSION(2, 2, 0))`

**8.2.6** `#define DEBUG_CONSOLE_DEVICE_TYPE_NONE 0U`

**8.2.7** `#define DEBUG_CONSOLE_DEVICE_TYPE_UART 1U`

**8.2.8** `#define DEBUG_CONSOLE_DEVICE_TYPE_LPUART 2U`

**8.2.9** `#define DEBUG_CONSOLE_DEVICE_TYPE_LPSCI 3U`

**8.2.10** `#define DEBUG_CONSOLE_DEVICE_TYPE_USBCDC 4U`

**8.2.11** `#define DEBUG_CONSOLE_DEVICE_TYPE_FLEXCOMM 5U`

**8.2.12** `#define DEBUG_CONSOLE_DEVICE_TYPE_IUART 6U`

**8.2.13** `#define DEBUG_CONSOLE_DEVICE_TYPE_VUSART 7U`

**8.2.14** `#define DEBUG_CONSOLE_DEVICE_TYPE_MINI_USART 8U`

**8.2.15** `#define DEBUG_CONSOLE_DEVICE_TYPE_SWO 9U`

**8.2.16** `#define ARRAY_SIZE( x ) (sizeof(x) / sizeof((x)[0]))`

## **8.3 Typedef Documentation**

**8.3.1** `typedef int32_t status_t`

## **8.4 Enumeration Type Documentation**

**8.4.1** `enum SYSCON_RSTn_t`

Defines the enumeration for peripheral reset control bits in PRESETCTRL/ASYNCPRESETCTRL registers

## Enumeration Type Documentation

### Enumerator

***kFLASH\_RST\_SHIFT\_RSTn*** Flash controller reset control  
***kSPIFI\_RST\_SHIFT\_RSTn*** SpiFi reset control  
***kMUX\_RST\_SHIFT\_RSTn*** Input mux reset control  
***kBLE\_TG\_RST\_SHIFT\_RSTn*** BLE power module reset control  
***kIOCON\_RST\_SHIFT\_RSTn*** IOCON reset control  
***kGPIO0\_RST\_SHIFT\_RSTn*** GPIO0 reset control  
***kPINT\_RST\_SHIFT\_RSTn*** Pin interrupt (PINT) reset control  
***kGINT\_RST\_SHIFT\_RSTn*** Grouped interrupt (PINT) reset control.  
***kDMA\_RST\_SHIFT\_RSTn*** DMA reset control  
***kWWDT\_RST\_SHIFT\_RSTn*** Watchdog timer reset control  
***kRTC\_RST\_SHIFT\_RSTn*** RTC reset control  
***kANA\_INT\_RST\_SHIFT\_RSTn*** Analog interrupt controller reset  
***kWKT\_RST\_SHIFT\_RSTn*** Wakeup timer reset  
***kADC0\_RST\_SHIFT\_RSTn*** ADC0 reset control  
***kEFUSE\_RST\_SHIFT\_RSTn*** EFUSE reset control  
***kFC0\_RST\_SHIFT\_RSTn*** Flexcomm Interface 0 reset control  
***kFC1\_RST\_SHIFT\_RSTn*** Flexcomm Interface 1 reset control  
***kFC2\_RST\_SHIFT\_RSTn*** Flexcomm Interface 2 reset control  
***kFC3\_RST\_SHIFT\_RSTn*** Flexcomm Interface 3 reset control  
***kFC4\_RST\_SHIFT\_RSTn*** Flexcomm Interface 4 reset control  
***kFC5\_RST\_SHIFT\_RSTn*** Flexcomm Interface 5 reset control  
***kIRB\_RST\_SHIFT\_RSTn*** IR Blaster reset control  
***kPWM\_RST\_SHIFT\_RSTn*** PWM reset control  
***kRNG\_RST\_SHIFT\_RSTn*** Random number generator reset control  
***kFC6\_RST\_SHIFT\_RSTn*** Flexcomm Interface 6 reset control  
***kUSART0\_RST\_SHIFT\_RSTn*** USART0 reset control == Flexcomm0  
***kUSART1\_RST\_SHIFT\_RSTn*** USART0 reset control == Flexcomm1  
***kI2C0\_RST\_SHIFT\_RSTn*** I2C0 reset control == Flexcomm 2  
***kI2C1\_RST\_SHIFT\_RSTn*** I2C1 reset control == Flexcomm 3  
***kSPI0\_RST\_SHIFT\_RSTn*** SPI0 reset control == Flexcomm 4  
***kSPI1\_RST\_SHIFT\_RSTn*** SPI1 reset control == Flexcomm 5  
***kI2C2\_RST\_SHIFT\_RSTn*** I2C2 reset control == Flexcomm 6  
***kBLE\_RST\_SHIFT\_RSTn*** Bluetooth LE modules reset control  
***kMODEM\_MASTER\_SHIFT\_RSTn*** AHB Modem master interface reset  
***kAES\_RST\_SHIFT\_RSTn*** Encryption module reset control  
***kRFP\_RST\_SHIFT\_RSTn*** Radio front end controller reset  
***kDMIC\_RST\_SHIFT\_RSTn*** Digital microphone interface reset control  
***kHASH\_RST\_SHIFT\_RSTn*** Hash SHA reset  
***kCTIMER0\_RST\_SHIFT\_RSTn*** CT32B0 reset control  
***kCTIMER1\_RST\_SHIFT\_RSTn*** CT32B1 reset control



## 8.4.2 enum \_status\_groups

Enumerator

*kStatusGroup\_Generic* Group number for generic status codes.  
*kStatusGroup\_FLASH* Group number for FLASH status codes.  
*kStatusGroup\_LPSPI* Group number for LPSPI status codes.  
*kStatusGroup\_FLEXIO\_SPI* Group number for FLEXIO SPI status codes.  
*kStatusGroup\_DSPI* Group number for DSPI status codes.  
*kStatusGroup\_FLEXIO\_UART* Group number for FLEXIO UART status codes.  
*kStatusGroup\_FLEXIO\_I2C* Group number for FLEXIO I2C status codes.  
*kStatusGroup\_LPI2C* Group number for LPI2C status codes.  
*kStatusGroup\_UART* Group number for UART status codes.  
*kStatusGroup\_I2C* Group number for UART status codes.  
*kStatusGroup\_LPSCI* Group number for LPSCI status codes.  
*kStatusGroup\_LPUART* Group number for LPUART status codes.  
*kStatusGroup\_SPI* Group number for SPI status code.  
*kStatusGroup\_XRDC* Group number for XRDC status code.  
*kStatusGroup\_SEMA42* Group number for SEMA42 status code.  
*kStatusGroup\_SDHC* Group number for SDHC status code.  
*kStatusGroup\_SDMMC* Group number for SDMMC status code.  
*kStatusGroup\_SAI* Group number for SAI status code.  
*kStatusGroup\_MCG* Group number for MCG status codes.  
*kStatusGroup\_SCG* Group number for SCG status codes.  
*kStatusGroup\_SDSPI* Group number for SDSPI status codes.  
*kStatusGroup\_FLEXIO\_I2S* Group number for FLEXIO I2S status codes.  
*kStatusGroup\_FLEXIO\_MCULCD* Group number for FLEXIO LCD status codes.  
*kStatusGroup\_FLASHIAP* Group number for FLASHIAP status codes.  
*kStatusGroup\_FLEXCOMM\_I2C* Group number for FLEXCOMM I2C status codes.  
*kStatusGroup\_I2S* Group number for I2S status codes.  
*kStatusGroup\_IUART* Group number for IUART status codes.  
*kStatusGroup\_CSI* Group number for CSI status codes.  
*kStatusGroup\_MIPI\_DSI* Group number for MIPI DSI status codes.  
*kStatusGroup\_SDRAMC* Group number for SDRAMC status codes.  
*kStatusGroup\_POWER* Group number for POWER status codes.  
*kStatusGroup\_ENET* Group number for ENET status codes.  
*kStatusGroup\_PHY* Group number for PHY status codes.  
*kStatusGroup\_TRGMUX* Group number for TRGMUX status codes.  
*kStatusGroup\_SMARTCARD* Group number for SMARTCARD status codes.  
*kStatusGroup\_LMEM* Group number for LMEM status codes.  
*kStatusGroup\_QSPI* Group number for QSPI status codes.  
*kStatusGroup\_DMA* Group number for DMA status codes.  
*kStatusGroup\_EDMA* Group number for EDMA status codes.  
*kStatusGroup\_DMAMGR* Group number for DMAMGR status codes.  
*kStatusGroup\_FLEXCAN* Group number for FlexCAN status codes.

## Enumeration Type Documentation

*kStatusGroup\_LTC* Group number for LTC status codes.

*kStatusGroup\_FLEXIO\_CAMERA* Group number for FLEXIO CAMERA status codes.

*kStatusGroup\_LPC\_SPI* Group number for LPC\_SPI status codes.

*kStatusGroup\_LPC\_USART* Group number for LPC\_USART status codes.

*kStatusGroup\_DMIC* Group number for DMIC status codes.

*kStatusGroup\_SDIF* Group number for SDIF status codes.

*kStatusGroup\_SPIFI* Group number for SPIFI status codes.

*kStatusGroup\_OTP* Group number for OTP status codes.

*kStatusGroup\_MCAN* Group number for MCAN status codes.

*kStatusGroup\_CAAM* Group number for CAAM status codes.

*kStatusGroup\_ECSPi* Group number for ECSPi status codes.

*kStatusGroup\_USDHC* Group number for USDHC status codes.

*kStatusGroup\_LPC\_I2C* Group number for LPC\_I2C status codes.

*kStatusGroup\_DCP* Group number for DCP status codes.

*kStatusGroup\_MSCAN* Group number for MSCAN status codes.

*kStatusGroup\_ESAI* Group number for ESAI status codes.

*kStatusGroup\_FLEXSPI* Group number for FLEXSPI status codes.

*kStatusGroup\_MMDC* Group number for MMDC status codes.

*kStatusGroup\_PDM* Group number for MIC status codes.

*kStatusGroup\_SDMA* Group number for SDMA status codes.

*kStatusGroup\_ICS* Group number for ICS status codes.

*kStatusGroup\_SPDIF* Group number for SPDIF status codes.

*kStatusGroup\_LPC\_MINISPI* Group number for LPC\_MINISPI status codes.

*kStatusGroup\_HASHCRYPT* Group number for Hashcrypt status codes.

*kStatusGroup\_LPC\_SPI\_SSP* Group number for LPC\_SPI\_SSP status codes.

*kStatusGroup\_I3C* Group number for I3C status codes.

*kStatusGroup\_LPC\_I2C\_1* Group number for LPC\_I2C\_1 status codes.

*kStatusGroup\_NOTIFIER* Group number for NOTIFIER status codes.

*kStatusGroup\_DebugConsole* Group number for debug console status codes.

*kStatusGroup\_SEMC* Group number for SEMC status codes.

*kStatusGroup\_ApplicationRangeStart* Starting number for application groups.

*kStatusGroup\_IAP* Group number for IAP status codes.

*kStatusGroup\_HAL\_GPIO* Group number for HAL GPIO status codes.

*kStatusGroup\_HAL\_UART* Group number for HAL UART status codes.

*kStatusGroup\_HAL\_TIMER* Group number for HAL TIMER status codes.

*kStatusGroup\_HAL\_SPI* Group number for HAL SPI status codes.

*kStatusGroup\_HAL\_I2C* Group number for HAL I2C status codes.

*kStatusGroup\_HAL\_FLASH* Group number for HAL FLASH status codes.

*kStatusGroup\_HAL\_PWM* Group number for HAL PWM status codes.

*kStatusGroup\_HAL\_RNG* Group number for HAL RNG status codes.

*kStatusGroup\_TIMERMANAGER* Group number for TiMER MANAGER status codes.

*kStatusGroup\_SERIALIZEDMANAGER* Group number for SERIAL MANAGER status codes.

*kStatusGroup\_LED* Group number for LED status codes.

*kStatusGroup\_BUTTON* Group number for BUTTON status codes.

*kStatusGroup\_EXTERN\_EEPROM* Group number for EXTERN EEPROM status codes.

***kStatusGroup\_SHELL*** Group number for SHELL status codes.

***kStatusGroup\_MEM\_MANAGER*** Group number for MEM MANAGER status codes.

***kStatusGroup\_LIST*** Group number for List status codes.

***kStatusGroup\_OSA*** Group number for OSA status codes.

***kStatusGroup\_COMMON\_TASK*** Group number for Common task status codes.

***kStatusGroup\_MSG*** Group number for messaging status codes.

***kStatusGroup\_SDK\_OCOTP*** Group number for OCOTP status codes.

***kStatusGroup\_SDK\_FLEXSPINOR*** Group number for FLEXSPINOR status codes.

***kStatusGroup\_CODEC*** Group number for codec status codes.

### 8.4.3 anonymous enum

## 8.5 Function Documentation

### 8.5.1 void RESET\_SetPeripheralReset ( reset\_ip\_name\_t *peripheral* )

Asserts reset signal to specified peripheral module.

Parameters

|                   |                                                                                                                                      |
|-------------------|--------------------------------------------------------------------------------------------------------------------------------------|
| <i>peripheral</i> | Assert reset to this peripheral. The enum argument contains encoding of reset register and reset bit position in the reset register. |
|-------------------|--------------------------------------------------------------------------------------------------------------------------------------|

### 8.5.2 void RESET\_ClearPeripheralReset ( reset\_ip\_name\_t *peripheral* )

Clears reset signal to specified peripheral module, allows it to operate.

Parameters

|                   |                                                                                                                                     |
|-------------------|-------------------------------------------------------------------------------------------------------------------------------------|
| <i>peripheral</i> | Clear reset to this peripheral. The enum argument contains encoding of reset register and reset bit position in the reset register. |
|-------------------|-------------------------------------------------------------------------------------------------------------------------------------|

### 8.5.3 void RESET\_PeripheralReset ( reset\_ip\_name\_t *peripheral* )

Reset peripheral module.

Parameters

---

## Function Documentation

|                   |                                                                                                                          |
|-------------------|--------------------------------------------------------------------------------------------------------------------------|
| <i>peripheral</i> | Peripheral to reset. The enum argument contains encoding of reset register and reset bit position in the reset register. |
|-------------------|--------------------------------------------------------------------------------------------------------------------------|

### 8.5.4 void RESET\_SystemReset ( void )

Full software reset of the chip. On reboot, function [POWER\\_GetResetCause\(\)](#) from `fsl_power.h` will return `RESET_SYS_REQ`

### 8.5.5 static status\_t EnableIRQ ( IRQn\_Type *interrupt* ) [inline], [static]

Enable LEVEL1 interrupt. For some devices, there might be multiple interrupt levels. For example, there are NVIC and intmux. Here the interrupts connected to NVIC are the LEVEL1 interrupts, because they are routed to the core directly. The interrupts connected to intmux are the LEVEL2 interrupts, they are routed to NVIC first then routed to core.

This function only enables the LEVEL1 interrupts. The number of LEVEL1 interrupts is indicated by the feature macro `FSL_FEATURE_NUMBER_OF_LEVEL1_INT_VECTORS`.

Parameters

|                  |                 |
|------------------|-----------------|
| <i>interrupt</i> | The IRQ number. |
|------------------|-----------------|

Return values

|                        |                                |
|------------------------|--------------------------------|
| <i>kStatus_Success</i> | Interrupt enabled successfully |
| <i>kStatus_Fail</i>    | Failed to enable the interrupt |

### 8.5.6 static status\_t DisableIRQ ( IRQn\_Type *interrupt* ) [inline], [static]

Disable LEVEL1 interrupt. For some devices, there might be multiple interrupt levels. For example, there are NVIC and intmux. Here the interrupts connected to NVIC are the LEVEL1 interrupts, because they are routed to the core directly. The interrupts connected to intmux are the LEVEL2 interrupts, they are routed to NVIC first then routed to core.

This function only disables the LEVEL1 interrupts. The number of LEVEL1 interrupts is indicated by the feature macro `FSL_FEATURE_NUMBER_OF_LEVEL1_INT_VECTORS`.

## Parameters

|                  |                 |
|------------------|-----------------|
| <i>interrupt</i> | The IRQ number. |
|------------------|-----------------|

## Return values

|                        |                                 |
|------------------------|---------------------------------|
| <i>kStatus_Success</i> | Interrupt disabled successfully |
| <i>kStatus_Fail</i>    | Failed to disable the interrupt |

### 8.5.7 static uint32\_t DisableGlobalIRQ ( void ) [inline], [static]

Disable the global interrupt and return the current primask register. User is required to provided the primask register for the [EnableGlobalIRQ\(\)](#).

## Returns

Current primask value.

### 8.5.8 static void EnableGlobalIRQ ( uint32\_t primask ) [inline], [static]

Set the primask register with the provided primask value but not just enable the primask. The idea is for the convenience of integration of RTOS. some RTOS get its own management mechanism of primask. User is required to use the [EnableGlobalIRQ\(\)](#) and [DisableGlobalIRQ\(\)](#) in pair.

## Parameters

|                |                                                                                                                                    |
|----------------|------------------------------------------------------------------------------------------------------------------------------------|
| <i>primask</i> | value of primask register to be restored. The primask value is supposed to be provided by the <a href="#">DisableGlobalIRQ()</a> . |
|----------------|------------------------------------------------------------------------------------------------------------------------------------|

### 8.5.9 void EnableDeepSleepIRQ ( IRQn\_Type interrupt )

Enable the interrupt for wake-up from deep sleep mode. Some interrupts are typically used in sleep mode only and will not occur during deep-sleep mode because relevant clocks are stopped. However, it is possible to enable those clocks (significantly increasing power consumption in the reduced power mode), making these wake-ups possible.

## Note

This function also enables the interrupt in the NVIC ([EnableIRQ\(\)](#) is called internally).

## Function Documentation

### Parameters

|                  |                 |
|------------------|-----------------|
| <i>interrupt</i> | The IRQ number. |
|------------------|-----------------|

### 8.5.10 void DisableDeepSleepIRQ ( IRQn\_Type *interrupt* )

Disable the interrupt for wake-up from deep sleep mode. Some interrupts are typically used in sleep mode only and will not occur during deep-sleep mode because relevant clocks are stopped. However, it is possible to enable those clocks (significantly increasing power consumption in the reduced power mode), making these wake-ups possible.

### Note

This function also disables the interrupt in the NVIC ([DisableIRQ\(\)](#) is called internally).

### Parameters

|                  |                 |
|------------------|-----------------|
| <i>interrupt</i> | The IRQ number. |
|------------------|-----------------|

### 8.5.11 void\* SDK\_Malloc ( size\_t *size*, size\_t *alignbytes* )

This is provided to support the dynamically allocated memory used in cache-able region.

### Parameters

|                   |                                |
|-------------------|--------------------------------|
| <i>size</i>       | The length required to malloc. |
| <i>alignbytes</i> | The alignment size.            |

### Return values

|            |                   |
|------------|-------------------|
| <i>The</i> | allocated memory. |
|------------|-------------------|

### 8.5.12 void SDK\_Free ( void \* *ptr* )

### Parameters

---

|            |                           |
|------------|---------------------------|
| <i>ptr</i> | The memory to be release. |
|------------|---------------------------|

### 8.5.13 void SDK\_DelayAtLeastUs ( uint32\_t *delay\_us*, uint32\_t *coreClock\_Hz* )

Please note that, this API uses while loop for delay, different run-time environments make the time not precise, if precise delay count was needed, please implement a new delay function with hardware timer.

Parameters

|                     |                                    |
|---------------------|------------------------------------|
| <i>delay_us</i>     | Delay time in unit of microsecond. |
| <i>coreClock_Hz</i> | Core clock frequency with Hz.      |





## Chapter 9

### CTIMER: Standard counter/timers

#### 9.1 Overview

The MCUXpresso SDK provides a driver for the cTimer module of MCUXpresso SDK devices.

#### 9.2 Function groups

The cTimer driver supports the generation of PWM signals, input capture, and setting up the timer match conditions.

##### 9.2.1 Initialization and deinitialization

The function `CTIMER_Init()` initializes the cTimer with specified configurations. The function `CTIMER_GetDefaultConfig()` gets the default configurations. The initialization function configures the counter/timer mode and input selection when running in counter mode.

The function `CTIMER_Deinit()` stops the timer and turns off the module clock.

##### 9.2.2 PWM Operations

The function `CTIMER_SetupPwm()` sets up channels for PWM output. Each channel has its own duty cycle, however the same PWM period is applied to all channels requesting the PWM output. The signal duty cycle is provided as a percentage of the PWM period. Its value should be between 0 and 100 0=inactive signal(0% duty cycle) and 100=always active signal (100% duty cycle).

The function `CTIMER_UpdatePwmDutycycle()` updates the PWM signal duty cycle of a particular channel.

##### 9.2.3 Match Operation

The function `CTIMER_SetupMatch()` sets up channels for match operation. Each channel is configured with a match value: if the counter should stop on match, if counter should reset on match, and output pin action. The output signal can be cleared, set, or toggled on match.

##### 9.2.4 Input capture operations

The function `CTIMER_SetupCapture()` sets up an channel for input capture. The user can specify the capture edge and if a interrupt should be generated when processing the input signal.

## Typical use case

### 9.3 Typical use case

#### 9.3.1 Match example

Set up a match channel to toggle output when a match occurs. Refer to the driver examples codes located at `<SDK_ROOT>/boards/<BOARD>/driver_examples/ctimer`

#### 9.3.2 PWM output example

Set up a channel for PWM output. Refer to the driver examples codes located at `<SDK_ROOT>/boards/<BOARD>/driver_examples/ctimer`

## Files

- file [fsl\\_ctimer.h](#)

## Data Structures

- struct [ctimer\\_match\\_config\\_t](#)  
*Match configuration. [More...](#)*
- struct [ctimer\\_config\\_t](#)  
*Timer configuration structure. [More...](#)*

## Enumerations

- enum [ctimer\\_capture\\_channel\\_t](#) {  
    [kCTIMER\\_Capture\\_0](#) = 0U,  
    [kCTIMER\\_Capture\\_1](#),  
    [kCTIMER\\_Capture\\_2](#),  
    [kCTIMER\\_Capture\\_3](#) }  
*List of Timer capture channels.*
- enum [ctimer\\_capture\\_edge\\_t](#) {  
    [kCTIMER\\_Capture\\_RiseEdge](#) = 1U,  
    [kCTIMER\\_Capture\\_FallEdge](#) = 2U,  
    [kCTIMER\\_Capture\\_BothEdge](#) = 3U }  
*List of capture edge options.*
- enum [ctimer\\_match\\_t](#) {  
    [kCTIMER\\_Match\\_0](#) = 0U,  
    [kCTIMER\\_Match\\_1](#),  
    [kCTIMER\\_Match\\_2](#),  
    [kCTIMER\\_Match\\_3](#) }  
*List of Timer match registers.*
- enum [ctimer\\_match\\_output\\_control\\_t](#) {  
    [kCTIMER\\_Output\\_NoAction](#) = 0U,  
    [kCTIMER\\_Output\\_Clear](#),  
    [kCTIMER\\_Output\\_Set](#),  
    [kCTIMER\\_Output\\_Toggle](#) }

- *List of output control options.*
- enum `ctimer_timer_mode_t`
  - *List of Timer modes.*
  - enum `ctimer_interrupt_enable_t` {
    - `kCTIMER_Match0InterruptEnable` = `CTIMER_MCR_MR0I_MASK`,
    - `kCTIMER_Match1InterruptEnable` = `CTIMER_MCR_MR1I_MASK`,
    - `kCTIMER_Match2InterruptEnable` = `CTIMER_MCR_MR2I_MASK`,
    - `kCTIMER_Match3InterruptEnable` = `CTIMER_MCR_MR3I_MASK`,
    - `kCTIMER_Capture0InterruptEnable` = `CTIMER_CCR_CAP0I_MASK`,
    - `kCTIMER_Capture1InterruptEnable` = `CTIMER_CCR_CAP1I_MASK`,
    - `kCTIMER_Capture2InterruptEnable` = `CTIMER_CCR_CAP2I_MASK`,
    - `kCTIMER_Capture3InterruptEnable` = `CTIMER_CCR_CAP3I_MASK` }
  - *List of Timer interrupts.*
  - enum `ctimer_status_flags_t` {
    - `kCTIMER_Match0Flag` = `CTIMER_IR_MR0INT_MASK`,
    - `kCTIMER_Match1Flag` = `CTIMER_IR_MR1INT_MASK`,
    - `kCTIMER_Match2Flag` = `CTIMER_IR_MR2INT_MASK`,
    - `kCTIMER_Match3Flag` = `CTIMER_IR_MR3INT_MASK`,
    - `kCTIMER_Capture0Flag` = `CTIMER_IR_CR0INT_MASK`,
    - `kCTIMER_Capture1Flag` = `CTIMER_IR_CR1INT_MASK`,
    - `kCTIMER_Capture2Flag` = `CTIMER_IR_CR2INT_MASK`,
    - `kCTIMER_Capture3Flag` = `CTIMER_IR_CR3INT_MASK` }
  - *List of Timer flags.*
  - enum `ctimer_callback_type_t` {
    - `kCTIMER_SingleCallback`,
    - `kCTIMER_MultipleCallback` }
  - *Callback type when registering for a callback.*

## Functions

- void `CTIMER_SetupMatch` (`CTIMER_Type *base`, `ctimer_match_t matchChannel`, const `ctimer_match_config_t *config`)
  - *Setup the match register.*
- void `CTIMER_SetupCapture` (`CTIMER_Type *base`, `ctimer_capture_channel_t capture`, `ctimer_capture_edge_t edge`, bool `enableInt`)
  - *Setup the capture.*
- static uint32\_t `CTIMER_GetTimerCountValue` (`CTIMER_Type *base`)
  - *Get the timer count value from TC register.*
- void `CTIMER_RegisterCallBack` (`CTIMER_Type *base`, `ctimer_callback_t *cb_func`, `ctimer_callback_type_t cb_type`)
  - *Register callback.*
- static void `CTIMER_Reset` (`CTIMER_Type *base`)
  - *Reset the counter.*

## Driver version

- #define `FSL_CTIMER_DRIVER_VERSION` (`MAKE_VERSION(2, 0, 2)`)
  - *Version 2.0.2.*

## Typical use case

### Initialization and deinitialization

- void [CTIMER\\_Init](#) (CTIMER\_Type \*base, const [ctimer\\_config\\_t](#) \*config)  
*Ungates the clock and configures the peripheral for basic operation.*
- void [CTIMER\\_Deinit](#) (CTIMER\_Type \*base)  
*Gates the timer clock.*
- void [CTIMER\\_GetDefaultConfig](#) ([ctimer\\_config\\_t](#) \*config)  
*Fills in the timers configuration structure with the default settings.*

### PWM setup operations

- [status\\_t CTIMER\\_SetupPwmPeriod](#) (CTIMER\_Type \*base, [ctimer\\_match\\_t](#) matchChannel, uint32\_t pwmPeriod, uint32\_t pulsePeriod, bool enableInt)  
*Configures the PWM signal parameters.*
- [status\\_t CTIMER\\_SetupPwm](#) (CTIMER\_Type \*base, [ctimer\\_match\\_t](#) matchChannel, uint8\_t dutyCyclePercent, uint32\_t pwmFreq\_Hz, uint32\_t srcClock\_Hz, bool enableInt)  
*Configures the PWM signal parameters.*
- static void [CTIMER\\_UpdatePwmPulsePeriod](#) (CTIMER\_Type \*base, [ctimer\\_match\\_t](#) matchChannel, uint32\_t pulsePeriod)  
*Updates the pulse period of an active PWM signal.*
- void [CTIMER\\_UpdatePwmDutycycle](#) (CTIMER\_Type \*base, [ctimer\\_match\\_t](#) matchChannel, uint8\_t dutyCyclePercent)  
*Updates the duty cycle of an active PWM signal.*

### Interrupt Interface

- static void [CTIMER\\_EnableInterrupts](#) (CTIMER\_Type \*base, uint32\_t mask)  
*Enables the selected Timer interrupts.*
- static void [CTIMER\\_DisableInterrupts](#) (CTIMER\_Type \*base, uint32\_t mask)  
*Disables the selected Timer interrupts.*
- static uint32\_t [CTIMER\\_GetEnabledInterrupts](#) (CTIMER\_Type \*base)  
*Gets the enabled Timer interrupts.*

### Status Interface

- static uint32\_t [CTIMER\\_GetStatusFlags](#) (CTIMER\_Type \*base)  
*Gets the Timer status flags.*
- static void [CTIMER\\_ClearStatusFlags](#) (CTIMER\_Type \*base, uint32\_t mask)  
*Clears the Timer status flags.*

### Counter Start and Stop

- static void [CTIMER\\_StartTimer](#) (CTIMER\_Type \*base)  
*Starts the Timer counter.*
- static void [CTIMER\\_StopTimer](#) (CTIMER\_Type \*base)  
*Stops the Timer counter.*

## 9.4 Data Structure Documentation

### 9.4.1 struct ctimer\_match\_config\_t

This structure holds the configuration settings for each match register.

#### Data Fields

- uint32\_t [matchValue](#)  
*This is stored in the match register.*
- bool [enableCounterReset](#)  
*true: Match will reset the counter false: Match will not reset the counter*
- bool [enableCounterStop](#)  
*true: Match will stop the counter false: Match will not stop the counter*
- [ctimer\\_match\\_output\\_control\\_t](#) [outControl](#)  
*Action to be taken on a match on the EM bit/output.*
- bool [outPinInitState](#)  
*Initial value of the EM bit/output.*
- bool [enableInterrupt](#)  
*true: Generate interrupt upon match false: Do not generate interrupt on match*

### 9.4.2 struct ctimer\_config\_t

This structure holds the configuration settings for the Timer peripheral. To initialize this structure to reasonable defaults, call the [CTIMER\\_GetDefaultConfig\(\)](#) function and pass a pointer to the configuration structure instance.

The configuration structure can be made constant so as to reside in flash.

#### Data Fields

- [ctimer\\_timer\\_mode\\_t](#) [mode](#)  
*Timer mode.*
- [ctimer\\_capture\\_channel\\_t](#) [input](#)  
*Input channel to increment the timer, used only in timer modes that rely on this input signal to increment TC.*
- uint32\_t [prescale](#)  
*Prescale value.*

## 9.5 Enumeration Type Documentation

### 9.5.1 enum ctimer\_capture\_channel\_t

Enumerator

***kCTIMER\_Capture\_0*** Timer capture channel 0.

## Enumeration Type Documentation

***kCTIMER\_Capture\_1*** Timer capture channel 1.  
***kCTIMER\_Capture\_2*** Timer capture channel 2.  
***kCTIMER\_Capture\_3*** Timer capture channel 3.

### 9.5.2 enum ctimer\_capture\_edge\_t

Enumerator

***kCTIMER\_Capture\_RiseEdge*** Capture on rising edge.  
***kCTIMER\_Capture\_FallEdge*** Capture on falling edge.  
***kCTIMER\_Capture\_BothEdge*** Capture on rising and falling edge.

### 9.5.3 enum ctimer\_match\_t

Enumerator

***kCTIMER\_Match\_0*** Timer match register 0.  
***kCTIMER\_Match\_1*** Timer match register 1.  
***kCTIMER\_Match\_2*** Timer match register 2.  
***kCTIMER\_Match\_3*** Timer match register 3.

### 9.5.4 enum ctimer\_match\_output\_control\_t

Enumerator

***kCTIMER\_Output\_NoAction*** No action is taken.  
***kCTIMER\_Output\_Clear*** Clear the EM bit/output to 0.  
***kCTIMER\_Output\_Set*** Set the EM bit/output to 1.  
***kCTIMER\_Output\_Toggle*** Toggle the EM bit/output.

### 9.5.5 enum ctimer\_interrupt\_enable\_t

Enumerator

***kCTIMER\_Match0InterruptEnable*** Match 0 interrupt.  
***kCTIMER\_Match1InterruptEnable*** Match 1 interrupt.  
***kCTIMER\_Match2InterruptEnable*** Match 2 interrupt.  
***kCTIMER\_Match3InterruptEnable*** Match 3 interrupt.  
***kCTIMER\_Capture0InterruptEnable*** Capture 0 interrupt.  
***kCTIMER\_Capture1InterruptEnable*** Capture 1 interrupt.  
***kCTIMER\_Capture2InterruptEnable*** Capture 2 interrupt.  
***kCTIMER\_Capture3InterruptEnable*** Capture 3 interrupt.

### 9.5.6 enum ctimer\_status\_flags\_t

Enumerator

***kCTIMER\_Match0Flag*** Match 0 interrupt flag.  
***kCTIMER\_Match1Flag*** Match 1 interrupt flag.  
***kCTIMER\_Match2Flag*** Match 2 interrupt flag.  
***kCTIMER\_Match3Flag*** Match 3 interrupt flag.  
***kCTIMER\_Capture0Flag*** Capture 0 interrupt flag.  
***kCTIMER\_Capture1Flag*** Capture 1 interrupt flag.  
***kCTIMER\_Capture2Flag*** Capture 2 interrupt flag.  
***kCTIMER\_Capture3Flag*** Capture 3 interrupt flag.

### 9.5.7 enum ctimer\_callback\_type\_t

When registering a callback an array of function pointers is passed the size could be 1 or 8, the callback type will tell that.

Enumerator

***kCTIMER\_SingleCallback*** Single Callback type where there is only one callback for the timer.  
 based on the status flags different channels needs to be handled differently  
***kCTIMER\_MultipleCallback*** Multiple Callback type where there can be 8 valid callbacks, one per  
 channel. for both match/capture

## 9.6 Function Documentation

### 9.6.1 void CTIMER\_Init ( CTIMER\_Type \* *base*, const ctimer\_config\_t \* *config* )

Note

This API should be called at the beginning of the application before using the driver.

Parameters

|               |                                              |
|---------------|----------------------------------------------|
| <i>base</i>   | Ctimer peripheral base address               |
| <i>config</i> | Pointer to the user configuration structure. |

### 9.6.2 void CTIMER\_Deinit ( CTIMER\_Type \* *base* )

## Function Documentation

### Parameters

|             |                                |
|-------------|--------------------------------|
| <i>base</i> | Ctimer peripheral base address |
|-------------|--------------------------------|

### 9.6.3 void CTIMER\_GetDefaultConfig ( ctimer\_config\_t \* config )

The default values are:

```
* config->mode = kCTIMER_TimerMode;
* config->input = kCTIMER_Capture_0;
* config->prescale = 0;
*
```

### Parameters

|               |                                              |
|---------------|----------------------------------------------|
| <i>config</i> | Pointer to the user configuration structure. |
|---------------|----------------------------------------------|

### 9.6.4 status\_t CTIMER\_SetupPwmPeriod ( CTIMER\_Type \* base, ctimer\_match\_t matchChannel, uint32\_t pwmPeriod, uint32\_t pulsePeriod, bool enableInt )

Enables PWM mode on the match channel passed in and will then setup the match value and other match parameters to generate a PWM signal. This function will assign match channel 3 to set the PWM cycle.

### Note

When setting PWM output from multiple output pins, all should use the same PWM period

### Parameters

|                     |                                                                                                                           |
|---------------------|---------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>         | Ctimer peripheral base address                                                                                            |
| <i>matchChannel</i> | Match pin to be used to output the PWM signal                                                                             |
| <i>pwmPeriod</i>    | PWM period match value                                                                                                    |
| <i>pulsePeriod</i>  | Pulse width match value                                                                                                   |
| <i>enableInt</i>    | Enable interrupt when the timer value reaches the match value of the PWM pulse, if it is 0 then no interrupt is generated |

### Returns

kStatus\_Success on success kStatus\_Fail If matchChannel passed in is 3; this channel is reserved to set the PWM period



### 9.6.5 **status\_t CTIMER\_SetupPwm ( CTIMER\_Type \* *base*, ctimer\_match\_t *matchChannel*, uint8\_t *dutyCyclePercent*, uint32\_t *pwmFreq\_Hz*, uint32\_t *srcClock\_Hz*, bool *enableInt* )**

Enables PWM mode on the match channel passed in and will then setup the match value and other match parameters to generate a PWM signal. This function will assign match channel 3 to set the PWM cycle.

#### Note

When setting PWM output from multiple output pins, all should use the same PWM frequency. Please use CTIMER\_SetupPwmPeriod to set up the PWM with high resolution.

#### Parameters

|                          |                                                                                                                           |
|--------------------------|---------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>              | Ctimer peripheral base address                                                                                            |
| <i>matchChannel</i>      | Match pin to be used to output the PWM signal                                                                             |
| <i>dutyCycle-Percent</i> | PWM pulse width; the value should be between 0 to 100                                                                     |
| <i>pwmFreq_Hz</i>        | PWM signal frequency in Hz                                                                                                |
| <i>srcClock_Hz</i>       | Timer counter clock in Hz                                                                                                 |
| <i>enableInt</i>         | Enable interrupt when the timer value reaches the match value of the PWM pulse, if it is 0 then no interrupt is generated |

#### Returns

kStatus\_Success on success kStatus\_Fail If matchChannel passed in is 3; this channel is reserved to set the PWM cycle

### 9.6.6 **static void CTIMER\_UpdatePwmPulsePeriod ( CTIMER\_Type \* *base*, ctimer\_match\_t *matchChannel*, uint32\_t *pulsePeriod* ) [inline], [static]**

#### Parameters

|             |                                |
|-------------|--------------------------------|
| <i>base</i> | Ctimer peripheral base address |
|-------------|--------------------------------|

## Function Documentation

|                     |                                               |
|---------------------|-----------------------------------------------|
| <i>matchChannel</i> | Match pin to be used to output the PWM signal |
| <i>pulsePeriod</i>  | New PWM pulse width match value               |

### 9.6.7 void CTIMER\_UpdatePwmDutycycle ( CTIMER\_Type \* *base*, ctimer\_match\_t *matchChannel*, uint8\_t *dutyCyclePercent* )

Note

Please use CTIMER\_UpdatePwmPulsePeriod to update the PWM with high resolution.

Parameters

|                          |                                                           |
|--------------------------|-----------------------------------------------------------|
| <i>base</i>              | Ctimer peripheral base address                            |
| <i>matchChannel</i>      | Match pin to be used to output the PWM signal             |
| <i>dutyCycle-Percent</i> | New PWM pulse width; the value should be between 0 to 100 |

### 9.6.8 void CTIMER\_SetupMatch ( CTIMER\_Type \* *base*, ctimer\_match\_t *matchChannel*, const ctimer\_match\_config\_t \* *config* )

User configuration is used to setup the match value and action to be taken when a match occurs.

Parameters

|                     |                                              |
|---------------------|----------------------------------------------|
| <i>base</i>         | Ctimer peripheral base address               |
| <i>matchChannel</i> | Match register to configure                  |
| <i>config</i>       | Pointer to the match configuration structure |

### 9.6.9 void CTIMER\_SetupCapture ( CTIMER\_Type \* *base*, ctimer\_capture\_channel\_t *capture*, ctimer\_capture\_edge\_t *edge*, bool *enableInt* )

Parameters

|                  |                                                                                                    |
|------------------|----------------------------------------------------------------------------------------------------|
| <i>base</i>      | Ctimer peripheral base address                                                                     |
| <i>capture</i>   | Capture channel to configure                                                                       |
| <i>edge</i>      | Edge on the channel that will trigger a capture                                                    |
| <i>enableInt</i> | Flag to enable channel interrupts, if enabled then the registered call back is called upon capture |

#### 9.6.10 static uint32\_t CTIMER\_GetTimerCountValue ( CTIMER\_Type \* *base* ) [inline], [static]

Parameters

|             |                                 |
|-------------|---------------------------------|
| <i>base</i> | Ctimer peripheral base address. |
|-------------|---------------------------------|

Returns

return the timer count value.

#### 9.6.11 void CTIMER\_RegisterCallBack ( CTIMER\_Type \* *base*, ctimer\_callback\_t \* *cb\_func*, ctimer\_callback\_type\_t *cb\_type* )

Parameters

|                |                                              |
|----------------|----------------------------------------------|
| <i>base</i>    | Ctimer peripheral base address               |
| <i>cb_func</i> | callback function                            |
| <i>cb_type</i> | callback function type, singular or multiple |

#### 9.6.12 static void CTIMER\_EnableInterrupts ( CTIMER\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

Parameters

|             |                                                                                                                         |
|-------------|-------------------------------------------------------------------------------------------------------------------------|
| <i>base</i> | Ctimer peripheral base address                                                                                          |
| <i>mask</i> | The interrupts to enable. This is a logical OR of members of the enumeration <a href="#">ctimer-_interrupt_enable_t</a> |

**9.6.13** `static void CTIMER_DisableInterrupts ( CTIMER_Type * base, uint32_t mask ) [inline], [static]`

## Parameters

|             |                                                                                                                         |
|-------------|-------------------------------------------------------------------------------------------------------------------------|
| <i>base</i> | Ctimer peripheral base address                                                                                          |
| <i>mask</i> | The interrupts to enable. This is a logical OR of members of the enumeration <a href="#">ctimer-_interrupt_enable_t</a> |

#### 9.6.14 static uint32\_t CTIMER\_GetEnabledInterrupts ( CTIMER\_Type \* *base* ) [inline], [static]

## Parameters

|             |                                |
|-------------|--------------------------------|
| <i>base</i> | Ctimer peripheral base address |
|-------------|--------------------------------|

## Returns

The enabled interrupts. This is the logical OR of members of the enumeration [ctimer\\_interrupt\\_enable\\_t](#)

#### 9.6.15 static uint32\_t CTIMER\_GetStatusFlags ( CTIMER\_Type \* *base* ) [inline], [static]

## Parameters

|             |                                |
|-------------|--------------------------------|
| <i>base</i> | Ctimer peripheral base address |
|-------------|--------------------------------|

## Returns

The status flags. This is the logical OR of members of the enumeration [ctimer\\_status\\_flags\\_t](#)

#### 9.6.16 static void CTIMER\_ClearStatusFlags ( CTIMER\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

## Parameters

|             |                                                                                                                      |
|-------------|----------------------------------------------------------------------------------------------------------------------|
| <i>base</i> | Ctimer peripheral base address                                                                                       |
| <i>mask</i> | The status flags to clear. This is a logical OR of members of the enumeration <a href="#">ctimer-_status_flags_t</a> |

**9.6.17** `static void CTIMER_StartTimer ( CTIMER_Type * base ) [inline],  
[static]`

Parameters

|             |                                |
|-------------|--------------------------------|
| <i>base</i> | Ctimer peripheral base address |
|-------------|--------------------------------|

**9.6.18 static void CTIMER\_StopTimer ( CTIMER\_Type \* *base* ) [inline], [static]**

Parameters

|             |                                |
|-------------|--------------------------------|
| <i>base</i> | Ctimer peripheral base address |
|-------------|--------------------------------|

**9.6.19 static void CTIMER\_Reset ( CTIMER\_Type \* *base* ) [inline], [static]**

The timer counter and prescale counter are reset on the next positive edge of the APB clock.

Parameters

|             |                                |
|-------------|--------------------------------|
| <i>base</i> | Ctimer peripheral base address |
|-------------|--------------------------------|





## Chapter 10 Debug Console

### 10.1 Overview

This chapter describes the programming interface of the debug console driver.

The debug console enables debug log messages to be output via the specified peripheral with frequency of the peripheral source clock and base address at the specified baud rate. Additionally, it provides input and output functions to scan and print formatted data. The below picture shows the layout of debug console.

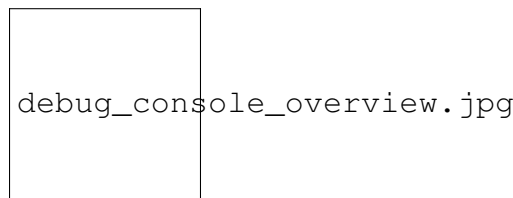


Figure 10.1.1: Debug console overview

### 10.2 Function groups

#### 10.2.1 Initialization

To initialize the debug console, call the [DbgConsole\\_Init\(\)](#) function with these parameters. This function automatically enables the module and the clock.

```
status_t DbgConsole_Init(uint8_t instance, uint32_t baudRate, serial_port_type_t
 device, uint32_t clkSrcFreq);
```

Select the supported debug console hardware device type, such as

```
typedef enum _serial_port_type
{
 kSerialPort_Uart = 1U,
 kSerialPort_UsbCdc,
 kSerialPort_Swo,
 kSerialPort_UsbCdcVirtual,
} serial_port_type_t;
```

After the initialization is successful, stdout and stdin are connected to the selected peripheral.

This example shows how to call the [DbgConsole\\_Init\(\)](#) given the user configuration structure.

```
DbgConsole_Init(BOARD_DEBUG_UART_INSTANCE, BOARD_DEBUG_UART_BAUDRATE, BOARD_DEBUG_UART_TYPE,
 BOARD_DEBUG_UART_CLK_FREQ);
```

## Function groups

### 10.2.2 Advanced Feature

The debug console provides input and output functions to scan and print formatted data.

- Support a format specifier for PRINTF following this prototype "`%[flags][width][.precision][length]specifier`", which is explained below

| flags   | Description                                                                                                                                                                                                                                                                                                                                                                             |
|---------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| -       | Left-justified within the given field width. Right-justified is the default.                                                                                                                                                                                                                                                                                                            |
| +       | Forces to precede the result with a plus or minus sign (+ or -) even for positive numbers. By default, only negative numbers are preceded with a - sign.                                                                                                                                                                                                                                |
| (space) | If no sign is written, a blank space is inserted before the value.                                                                                                                                                                                                                                                                                                                      |
| #       | Used with o, x, or X specifiers the value is preceded with 0, 0x, or 0X respectively for values other than zero. Used with e, E and f, it forces the written output to contain a decimal point even if no digits would follow. By default, if no digits follow, no decimal point is written. Used with g or G the result is the same as with e or E but trailing zeros are not removed. |
| 0       | Left-pads the number with zeroes (0) instead of spaces, where padding is specified (see width sub-specifier).                                                                                                                                                                                                                                                                           |

| Width    | Description                                                                                                                                                                                            |
|----------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| (number) | A minimum number of characters to be printed. If the value to be printed is shorter than this number, the result is padded with blank spaces. The value is not truncated even if the result is larger. |
| *        | The width is not specified in the format string, but as an additional integer value argument preceding the argument that has to be formatted.                                                          |

| <b>.precision</b> | <b>Description</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
|-------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| .number           | For integer specifiers (d, i, o, u, x, X) precision specifies the minimum number of digits to be written. If the value to be written is shorter than this number, the result is padded with leading zeros. The value is not truncated even if the result is longer. A precision of 0 means that no character is written for the value 0. For e, E, and f specifiers this is the number of digits to be printed after the decimal point. For g and G specifiers This is the maximum number of significant digits to be printed. For s this is the maximum number of characters to be printed. By default, all characters are printed until the ending null character is encountered. For c type it has no effect. When no precision is specified, the default is 1. If the period is specified without an explicit value for precision, 0 is assumed. |
| .*                | The precision is not specified in the format string, but as an additional integer value argument preceding the argument that has to be formatted.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |

| <b>length</b>  | <b>Description</b> |
|----------------|--------------------|
| Do not support |                    |

| <b>specifier</b> | <b>Description</b>                           |
|------------------|----------------------------------------------|
| d or i           | Signed decimal integer                       |
| f                | Decimal floating point                       |
| F                | Decimal floating point capital letters       |
| x                | Unsigned hexadecimal integer                 |
| X                | Unsigned hexadecimal integer capital letters |
| o                | Signed octal                                 |
| b                | Binary value                                 |
| p                | Pointer address                              |
| u                | Unsigned decimal integer                     |
| c                | Character                                    |
| s                | String of characters                         |
| n                | Nothing printed                              |

## Function groups

- Support a format specifier for SCANF following this prototype " %[\*][width][length]specifier", which is explained below

| *                                                                                                                                                                | Description |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------|
| An optional starting asterisk indicates that the data is to be read from the stream but ignored. In other words, it is not stored in the corresponding argument. |             |

| width                                                                                        | Description |
|----------------------------------------------------------------------------------------------|-------------|
| This specifies the maximum number of characters to be read in the current reading operation. |             |

| length      | Description                                                                                                                                                                                             |
|-------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| hh          | The argument is interpreted as a signed character or unsigned character (only applies to integer specifiers: i, d, o, u, x, and X).                                                                     |
| h           | The argument is interpreted as a short integer or unsigned short integer (only applies to integer specifiers: i, d, o, u, x, and X).                                                                    |
| l           | The argument is interpreted as a long integer or unsigned long integer for integer specifiers (i, d, o, u, x, and X) and as a wide character or wide character string for specifiers c and s.           |
| ll          | The argument is interpreted as a long long integer or unsigned long long integer for integer specifiers (i, d, o, u, x, and X) and as a wide character or wide character string for specifiers c and s. |
| L           | The argument is interpreted as a long double (only applies to floating point specifiers: e, E, f, g, and G).                                                                                            |
| j or z or t | Not supported                                                                                                                                                                                           |

| specifier | Qualifying Input                                                                                                                                                                                                                                 | Type of argument |
|-----------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------|
| c         | Single character: Reads the next character. If a width different from 1 is specified, the function reads width characters and stores them in the successive locations of the array passed as argument. No null character is appended at the end. | char *           |

| specifier              | Qualifying Input                                                                                                                                                                                                            | Type of argument |
|------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------|
| i                      | Integer: : Number optionally preceded with a + or - sign                                                                                                                                                                    | int *            |
| d                      | Decimal integer: Number optionally preceded with a + or - sign                                                                                                                                                              | int *            |
| a, A, e, E, f, F, g, G | Floating point: Decimal number containing a decimal point, optionally preceded by a + or - sign and optionally followed by the e or E character and a decimal number. Two examples of valid entries are -732.103 and 7.12e4 | float *          |
| o                      | Octal Integer:                                                                                                                                                                                                              | int *            |
| s                      | String of characters. This reads subsequent characters until a white space is found (white space characters are considered to be blank, newline, and tab).                                                                  | char *           |
| u                      | Unsigned decimal integer.                                                                                                                                                                                                   | unsigned int *   |

The debug console has its own printf/scanf/putchar/getchar functions which are defined in the header file.

```
int DbgConsole_Printf(const char *fmt_s, ...);
int DbgConsole_Putchar(int ch);
int DbgConsole_Scanf(char *fmt_ptr, ...);
int DbgConsole_Getchar(void);
```

This utility supports selecting toolchain's printf/scanf or the MCUXpresso SDK printf/scanf.

```
#if SDK_DEBUGCONSOLE /* Select printf, scanf, putchar, getchar of SDK version. */
#define PRINTF DbgConsole_Printf
#define SCANF DbgConsole_Scanf
#define PUTCHAR DbgConsole_Putchar
#define GETCHAR DbgConsole_Getchar
#else /* Select printf, scanf, putchar, getchar of toolchain. */
#define PRINTF printf
#define SCANF scanf
#define PUTCHAR putchar
#define GETCHAR getchar
#endif /* SDK_DEBUGCONSOLE */
```

## 10.3 Typical use case

Some examples use the PUTCHAR & GETCHAR function

```
ch = GETCHAR();
PUTCHAR(ch);
```

## Typical use case

### Some examples use the PRINTF function

Statement prints the string format.

```
PRINTF("%s %s\r\n", "Hello", "world!");
```

Statement prints the hexadecimal format/

```
PRINTF("0x%02X hexadecimal number equivalent 255", 255);
```

Statement prints the decimal floating point and unsigned decimal.

```
PRINTF("Execution timer: %s\n\rTime: %u ticks %2.5f milliseconds\n\rDONE\n\r", "1 day", 86400, 86.4);
```

### Some examples use the SCANF function

```
PRINTF("Enter a decimal number: ");
SCANF("%d", &i);
PRINTF("\r\nYou have entered %d.\r\n", i, i);
PRINTF("Enter a hexadecimal number: ");
SCANF("%x", &i);
PRINTF("\r\nYou have entered 0x%X (%d).\r\n", i, i);
```

### Print out failure messages using MCUXpresso SDK \_\_assert\_func:

```
void __assert_func(const char *file, int line, const char *func, const char *failedExpr)
{
 PRINTF("ASSERT ERROR \" %s \": file \"%s\" Line \"%d\" function name \"%s\" \n", failedExpr, file
 , line, func);
 for (;;)
 {}
}
```

### Note:

To use 'printf' and 'scanf' for GNUC Base, add file 'fsl\_sbrk.c' in path: ..\{package}\devices\{subset}\utilities\fsl-  
\_sbrk.c to your project.

This chapter describes the programming interface of the debug console driver.

The debug console enables debug log messages to be output via the specified peripheral with frequency of the peripheral source clock and base address at the specified baud rate. Additionally, it provides input and output functions to scan and print formatted data.

## 10.4 Function groups

### 10.4.1 Initialization

To initialize the debug console, call the [DbgConsole\\_Init\(\)](#) function with these parameters. This function automatically enables the module and the clock.

```
status_t DbgConsole_Init(uint8_t instance, uint32_t baudRate, serial_port_type_t
 device, uint32_t clkSrcFreq);
```

Selects the supported debug console hardware device type, such as

```
typedef enum _serial_port_type
{
 kSerialPort_None = 0U,
 kSerialPort_Uart = 1U,
} serial_port_type_t;
```

After the initialization is successful, stdout and stdin are connected to the selected peripheral. The debug console state is stored in the `debug_console_state_t` structure, such as shown here.

```
typedef struct DebugConsoleState
{
 uint8_t uartHandleBuffer[HAL_UART_HANDLE_SIZE];
 hal_uart_status_t (*putChar)(hal_uart_handle_t handle, const uint8_t *data, size_t
 length);
 hal_uart_status_t (*getChar)(hal_uart_handle_t handle, uint8_t *data, size_t length);

 serial_port_type_t type;
} debug_console_state_t;
```

This example shows how to call the [DbgConsole\\_Init\(\)](#) given the user configuration structure.

```
DbgConsole_Init(BOARD_DEBUG_USART_INSTANCE, BOARD_DEBUG_USART_BAUDRATE,
 BOARD_DEBUG_USART_TYPE,
 BOARD_DEBUG_USART_CLK_FREQ);
```

### 10.4.2 Advanced Feature

The debug console provides input and output functions to scan and print formatted data.

- Support a format specifier for PRINTF following this prototype " `%[flags][width][.precision][length]specifier`", which is explained below

| flags | Description                                                                  |
|-------|------------------------------------------------------------------------------|
| -     | Left-justified within the given field width. Right-justified is the default. |

## Function groups

| flags   | Description                                                                                                                                                                                                                                                                                                                                                                             |
|---------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| +       | Forces to precede the result with a plus or minus sign (+ or -) even for positive numbers. By default, only negative numbers are preceded with a - sign.                                                                                                                                                                                                                                |
| (space) | If no sign is written, a blank space is inserted before the value.                                                                                                                                                                                                                                                                                                                      |
| #       | Used with o, x, or X specifiers the value is preceded with 0, 0x, or 0X respectively for values other than zero. Used with e, E and f, it forces the written output to contain a decimal point even if no digits would follow. By default, if no digits follow, no decimal point is written. Used with g or G the result is the same as with e or E but trailing zeros are not removed. |
| 0       | Left-pads the number with zeroes (0) instead of spaces, where padding is specified (see width sub-specifier).                                                                                                                                                                                                                                                                           |

| Width    | Description                                                                                                                                                                                            |
|----------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| (number) | A minimum number of characters to be printed. If the value to be printed is shorter than this number, the result is padded with blank spaces. The value is not truncated even if the result is larger. |
| *        | The width is not specified in the format string, but as an additional integer value argument preceding the argument that has to be formatted.                                                          |



| <b>.precision</b> | <b>Description</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
|-------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| .number           | For integer specifiers (d, i, o, u, x, X) precision specifies the minimum number of digits to be written. If the value to be written is shorter than this number, the result is padded with leading zeros. The value is not truncated even if the result is longer. A precision of 0 means that no character is written for the value 0. For e, E, and f specifiers this is the number of digits to be printed after the decimal point. For g and G specifiers This is the maximum number of significant digits to be printed. For s this is the maximum number of characters to be printed. By default, all characters are printed until the ending null character is encountered. For c type it has no effect. When no precision is specified, the default is 1. If the period is specified without an explicit value for precision, 0 is assumed. |
| .*                | The precision is not specified in the format string, but as an additional integer value argument preceding the argument that has to be formatted.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |

| <b>length</b>  | <b>Description</b> |
|----------------|--------------------|
| Do not support |                    |

| <b>specifier</b> | <b>Description</b>                           |
|------------------|----------------------------------------------|
| d or i           | Signed decimal integer                       |
| f                | Decimal floating point                       |
| F                | Decimal floating point capital letters       |
| x                | Unsigned hexadecimal integer                 |
| X                | Unsigned hexadecimal integer capital letters |
| o                | Signed octal                                 |
| b                | Binary value                                 |
| p                | Pointer address                              |
| u                | Unsigned decimal integer                     |
| c                | Character                                    |
| s                | String of characters                         |
| n                | Nothing printed                              |

## Function groups

- Support a format specifier for SCANF following this prototype " %[\*][width][length]specifier", which is explained below

| *                                                                                                                                                                | Description |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------|
| An optional starting asterisk indicates that the data is to be read from the stream but ignored. In other words, it is not stored in the corresponding argument. |             |

| width                                                                                        | Description |
|----------------------------------------------------------------------------------------------|-------------|
| This specifies the maximum number of characters to be read in the current reading operation. |             |

| length      | Description                                                                                                                                                                                             |
|-------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| hh          | The argument is interpreted as a signed character or unsigned character (only applies to integer specifiers: i, d, o, u, x, and X).                                                                     |
| h           | The argument is interpreted as a short integer or unsigned short integer (only applies to integer specifiers: i, d, o, u, x, and X).                                                                    |
| l           | The argument is interpreted as a long integer or unsigned long integer for integer specifiers (i, d, o, u, x, and X) and as a wide character or wide character string for specifiers c and s.           |
| ll          | The argument is interpreted as a long long integer or unsigned long long integer for integer specifiers (i, d, o, u, x, and X) and as a wide character or wide character string for specifiers c and s. |
| L           | The argument is interpreted as a long double (only applies to floating point specifiers: e, E, f, g, and G).                                                                                            |
| j or z or t | Not supported                                                                                                                                                                                           |

| specifier | Qualifying Input                                                                                                                                                                                                                                 | Type of argument |
|-----------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------|
| c         | Single character: Reads the next character. If a width different from 1 is specified, the function reads width characters and stores them in the successive locations of the array passed as argument. No null character is appended at the end. | char *           |

| specifier              | Qualifying Input                                                                                                                                                                                                            | Type of argument |
|------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------|
| i                      | Integer: : Number optionally preceded with a + or - sign                                                                                                                                                                    | int *            |
| d                      | Decimal integer: Number optionally preceded with a + or - sign                                                                                                                                                              | int *            |
| a, A, e, E, f, F, g, G | Floating point: Decimal number containing a decimal point, optionally preceded by a + or - sign and optionally followed by the e or E character and a decimal number. Two examples of valid entries are -732.103 and 7.12e4 | float *          |
| o                      | Octal Integer:                                                                                                                                                                                                              | int *            |
| s                      | String of characters. This reads subsequent characters until a white space is found (white space characters are considered to be blank, newline, and tab).                                                                  | char *           |
| u                      | Unsigned decimal integer.                                                                                                                                                                                                   | unsigned int *   |

The debug console has its own printf/scanf/putchar/getchar functions which are defined in the header file.

```
int DbgConsole_Printf(const char *fmt_s, ...);
int DbgConsole_Putchar(int ch);
int DbgConsole_Scanf(const char *fmt_ptr, ...);
int DbgConsole_Getchar(void);
```

This utility supports selecting toolchain's printf/scanf or the MCUXpresso SDK printf/scanf.

```
#if SDK_DEBUGCONSOLE /* Select printf, scanf, putchar, getchar of SDK version. */
#define PRINTF DbgConsole_Printf
#define SCANF DbgConsole_Scanf
#define PUTCHAR DbgConsole_Putchar
#define GETCHAR DbgConsole_Getchar
#else /* Select printf, scanf, putchar, getchar of toolchain. */
#define PRINTF printf
#define SCANF scanf
#define PUTCHAR putchar
#define GETCHAR getchar
#endif /* SDK_DEBUGCONSOLE */
```

## 10.5 Typical use case

### Some examples use the PUTCHAR & GETCHAR function

```
ch = GETCHAR();
PUTCHAR(ch);
```

## Typical use case

### Some examples use the PRINTF function

Statement prints the string format.

```
PRINTF("%s %s\r\n", "Hello", "world!");
```

Statement prints the hexadecimal format/

```
PRINTF("0x%02X hexadecimal number equivalent 255", 255);
```

Statement prints the decimal floating point and unsigned decimal.

```
PRINTF("Execution timer: %s\n\rTime: %u ticks %2.5f milliseconds\n\rDONE\n\r", "1 day", 86400, 86.4);
```

### Some examples use the SCANF function

```
PRINTF("Enter a decimal number: ");
SCANF("%d", &i);
PRINTF("\r\nYou have entered %d.\r\n", i, i);
PRINTF("Enter a hexadecimal number: ");
SCANF("%x", &i);
PRINTF("\r\nYou have entered 0x%X (%d).\r\n", i, i);
```

### Print out failure messages using MCUXpresso SDK \_\_assert\_func:

```
void __assert_func(const char *file, int line, const char *func, const char *failedExpr)
{
 PRINTF("ASSERT ERROR \" %s \": file \"%s\" Line \"%d\" function name \"%s\" \n", failedExpr, file
 , line, func);
 for (;;)
 {}
}
```

### Note:

To use 'printf' and 'scanf' for GNUC Base, add file 'fsl\_sbrk.c' in path: ..\{package}\devices\{subset}\utilities\fsl-  
\_sbrk.c to your project.

## Modules

- [SWO](#)
- [Semihosting](#)

## Macros

- `#define DEBUGCONSOLE_REDIRECT_TO_TOOLCHAIN 0U`  
*Definition select redirect toolchain printf, scanf to uart or not.*
- `#define DEBUGCONSOLE_REDIRECT_TO_SDK 1U`  
*Select SDK version printf, scanf.*
- `#define DEBUGCONSOLE_DISABLE 2U`  
*Disable debugconsole function.*
- `#define SDK_DEBUGCONSOLE 1U`  
*Definition to select sdk or toolchain printf, scanf.*
- `#define SDK_DEBUGCONSOLE_UART`  
*whether provide low level IO implementation to toolchain printf and scanf.*
- `#define PRINTF DbgConsole_Printf`  
*Definition to select redirect toolchain printf, scanf to uart or not.*

## Typedefs

- `typedef void(* printfCb )(char *buf, int32_t *indicator, char val, int len)`  
*A function pointer which is used when format printf log.*

## Functions

- `int StrFormatPrintf (const char *fmt, va_list ap, char *buf, printfCb cb)`  
*This function outputs its parameters according to a formatted string.*
- `int StrFormatScanf (const char *line_ptr, char *format, va_list args_ptr)`  
*Converts an input line of ASCII characters based upon a provided string format.*

## Variables

- `serial_handle_t g_serialHandle`  
*serial manager handle*

## Initialization

- `status_t DbgConsole_Init (uint8_t instance, uint32_t baudRate, serial_port_type_t device, uint32_t clkSrcFreq)`  
*Initializes the peripheral used for debug messages.*
- `status_t DbgConsole_Deinit (void)`  
*De-initializes the peripheral used for debug messages.*
- `int DbgConsole_Printf (const char *formatString,...)`  
*Writes formatted output to the standard output stream.*
- `int DbgConsole_Putchar (int ch)`  
*Writes a character to stdout.*
- `int DbgConsole_Scanf (char *formatString,...)`  
*Reads formatted data from the standard input stream.*
- `int DbgConsole_Getchar (void)`  
*Reads a character from standard input.*
- `status_t DbgConsole_Flush (void)`  
*Debug console flush.*

## Function Documentation

### 10.6 Macro Definition Documentation

#### 10.6.1 **#define DEBUGCONSOLE\_REDIRECT\_TO\_TOOLCHAIN 0U**

Select toolchain printf and scanf.

#### 10.6.2 **#define DEBUGCONSOLE\_REDIRECT\_TO\_SDK 1U**

#### 10.6.3 **#define DEBUGCONSOLE\_DISABLE 2U**

#### 10.6.4 **#define SDK\_DEBUGCONSOLE 1U**

The macro only support to be redefined in project setting.

#### 10.6.5 **#define SDK\_DEBUGCONSOLE\_UART**

For example, within MCUXpresso, if the macro SDK\_DEBUGCONSOLE\_UART is defined, **sys\_write** and **\_\_sys\_readc** will be used when **\_\_REDLIB** is defined; **\_write** and **\_read** will be used in other cases. If the macro SDK\_DEBUGCONSOLE\_UART is not defined, the semihost will be used.

#### 10.6.6 **#define PRINTF DbgConsole\_Printf**

if SDK\_DEBUGCONSOLE defined to 0,it represents select toolchain printf, scanf. if SDK\_DEBUGCONSOLE defined to 1,it represents select SDK version printf, scanf. if SDK\_DEBUGCONSOLE defined to 2,it represents disable debugconsole function.

### 10.7 Function Documentation

#### 10.7.1 **status\_t DbgConsole\_Init ( uint8\_t *instance*, uint32\_t *baudRate*, serial\_port\_type\_t *device*, uint32\_t *clkSrcFreq* )**

Call this function to enable debug log messages to be output via the specified peripheral initialized by the serial manager module. After this function has returned, stdout and stdin are connected to the selected peripheral.

Parameters

---

|                   |                                                                                                                                                                                                                     |
|-------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>instance</i>   | The instance of the module.                                                                                                                                                                                         |
| <i>baudRate</i>   | The desired baud rate in bits per second.                                                                                                                                                                           |
| <i>device</i>     | Low level device type for the debug console, can be one of the following. <ul style="list-style-type: none"> <li>• kSerialPort_Uart,</li> <li>• kSerialPort_UsbCdc</li> <li>• kSerialPort_UsbCdcVirtual.</li> </ul> |
| <i>clkSrcFreq</i> | Frequency of peripheral source clock.                                                                                                                                                                               |

Returns

Indicates whether initialization was successful or not.

Return values

|                        |                        |
|------------------------|------------------------|
| <i>kStatus_Success</i> | Execution successfully |
|------------------------|------------------------|

### 10.7.2 status\_t DbgConsole\_Deinit ( void )

Call this function to disable debug log messages to be output via the specified peripheral initialized by the serial manager module.

Returns

Indicates whether de-initialization was successful or not.

### 10.7.3 int DbgConsole\_Printf ( const char \* *formatString*, ... )

Call this function to write a formatted output to the standard output stream.

Parameters

|                     |                        |
|---------------------|------------------------|
| <i>formatString</i> | Format control string. |
|---------------------|------------------------|

Returns

Returns the number of characters printed or a negative value if an error occurs.

### 10.7.4 int DbgConsole\_Putchar ( int *ch* )

Call this function to write a character to stdout.

## Function Documentation

### Parameters

|           |                          |
|-----------|--------------------------|
| <i>ch</i> | Character to be written. |
|-----------|--------------------------|

### Returns

Returns the character written.

## 10.7.5 int DbgConsole\_Scanf ( char \* *formatString*, ... )

Call this function to read formatted data from the standard input stream.

### Note

Due the limitation in the BM OSA environment (CPU is blocked in the function, other tasks will not be scheduled), the function cannot be used when the `DEBUG_CONSOLE_TRANSFER_NON_BLOCKING` is set in the BM OSA environment. And an error is returned when the function called in this case. The suggestion is that polling the non-blocking function `DbgConsole_TryGetchar` to get the input char.

### Parameters

|                     |                        |
|---------------------|------------------------|
| <i>formatString</i> | Format control string. |
|---------------------|------------------------|

### Returns

Returns the number of fields successfully converted and assigned.

## 10.7.6 int DbgConsole\_Getchar ( void )

Call this function to read a character from standard input.

### Note

Due the limitation in the BM OSA environment (CPU is blocked in the function, other tasks will not be scheduled), the function cannot be used when the `DEBUG_CONSOLE_TRANSFER_NON_BLOCKING` is set in the BM OSA environment. And an error is returned when the function called in this case. The suggestion is that polling the non-blocking function `DbgConsole_TryGetchar` to get the input char.

### Returns

Returns the character read.



### 10.7.7 status\_t DbgConsole\_Flush ( void )

Call this function to wait the tx buffer empty. If interrupt transfer is using, make sure the global IRQ is enable before call this function This function should be called when 1, before enter power down mode 2, log is required to print to terminal immediately

Returns

Indicates whether wait idle was successful or not.

### 10.7.8 int StrFormatPrintf ( const char \* *fmt*, va\_list *ap*, char \* *buf*, printfCb *cb* )

Note

I/O is performed by calling given function pointer using following (\*func\_ptr)(c);

Parameters

|    |            |                                |
|----|------------|--------------------------------|
| in | <i>fmt</i> | Format string for printf.      |
| in | <i>ap</i>  | Arguments to printf.           |
| in | <i>buf</i> | pointer to the buffer          |
|    | <i>cb</i>  | print callbck function pointer |

Returns

Number of characters to be print

### 10.7.9 int StrFormatScanf ( const char \* *line\_ptr*, char \* *format*, va\_list *args\_ptr* )

Parameters

|    |                 |                                           |
|----|-----------------|-------------------------------------------|
| in | <i>line_ptr</i> | The input line of ASCII data.             |
| in | <i>format</i>   | Format first points to the format string. |
| in | <i>args_ptr</i> | The list of parameters.                   |

Returns

Number of input items converted and assigned.

## Function Documentation

Return values

|               |                                   |
|---------------|-----------------------------------|
| <i>IO_EOF</i> | When line_ptr is empty string "". |
|---------------|-----------------------------------|

## 10.8 Semihosting

Semihosting is a mechanism for ARM targets to communicate input/output requests from application code to a host computer running a debugger. This mechanism can be used, for example, to enable functions in the C library, such as `printf()` and `scanf()`, to use the screen and keyboard of the host rather than having a screen and keyboard on the target system.

### 10.8.1 Guide Semihosting for IAR

**NOTE:** After the setting both "printf" and "scanf" are available for debugging, if you want use PRINTF with semihosting, please make sure the `SDK_DEBUGCONSOLE` is disabled.

#### Step 1: Setting up the environment

1. To set debugger options, choose Project>Options. In the Debugger category, click the Setup tab.
2. Select Run to main and click OK. This ensures that the debug session starts by running the main function.
3. The project is now ready to be built.

#### Step 2: Building the project

1. Compile and link the project by choosing Project>Make or F7.
2. Alternatively, click the Make button on the tool bar. The Make command compiles and links those files that have been modified.

#### Step 3: Starting semihosting

1. Choose "Semihosting\_IAR" project -> "Options" -> "Debugger" -> "J-Link/J-Trace".
2. Choose tab "J-Link/J-Trace" -> "Connection" tab -> "SWD".
3. Choose tab "General Options" -> "Library Configurations", select Semihosted, select Via semihosting.
1. Make sure the `SDK_DEBUGCONSOLE_UART` is not defined, remove the default definition in `fsl_debug_console.h`.
1. Start the project by choosing Project>Download and Debug.
2. Choose View>Terminal I/O to display the output from the I/O operations.

### 10.8.2 Guide Semihosting for Keil $\mu$ Vision

**NOTE:** Semihosting is not support by MDK-ARM, use the retargeting functionality of MDK-ARM instead.

### 10.8.3 Guide Semihosting for MCUXpresso IDE

#### Step 1: Setting up the environment

1. To set debugger options, choose Project>Properties. select the setting category.
2. Select Tool Settings, unfold MCU C Compile.
3. Select Preprocessor item.
4. Set SDK\_DEBUGCONSOLE=0, if set SDK\_DEBUGCONSOLE=1, the log will be redirect to the UART.

#### Step 2: Building the project

1. Compile and link the project.

#### Step 3: Starting semihosting

1. Download and debug the project.
2. When the project runs successfully, the result can be seen in the Console window.

Semihosting can also be selected through the "Quick settings" menu in the left bottom window, Quick settings->SDK Debug Console->Semihost console.

### 10.8.4 Guide Semihosting for ARMGCC

#### Step 1: Setting up the environment

1. Turn on "J-LINK GDB Server" -> Select suitable "Target device" -> "OK".
2. Turn on "PuTTY". Set up as follows.
  - "Host Name (or IP address)" : localhost
  - "Port" :2333
  - "Connection type" : Telet.
  - Click "Open".
3. Increase "Heap/Stack" for GCC to 0x2000:

#### Add to "CMakeLists.txt"

```
SET(CMAKE_EXE_LINKER_FLAGS_RELEASE "${CMAKE_EXE_LINKER_FLAGS_RELEASE}
--defsym=__stack_size__=0x2000")

SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} --
defsym=__stack_size__=0x2000")

SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} --
defsym=__heap_size__=0x2000")

SET(CMAKE_EXE_LINKER_FLAGS_RELEASE "${CMAKE_EXE_LINKER_FLAGS_RELEASE}
--defsym=__heap_size__=0x2000")
```

**Step 2: Building the project**

1. Change "CMakeLists.txt":

**Change** "SET(CMAKE\_EXE\_LINKER\_FLAGS\_RELEASE "\${CMAKE\_EXE\_LINKER\_FLAGS\_RELEASE} -specs=nano.specs")"

**to** "SET(CMAKE\_EXE\_LINKER\_FLAGS\_RELEASE "\${CMAKE\_EXE\_LINKER\_FLAGS\_RELEASE} -specs=rdimon.specs")"

**Replace paragraph**

SET(CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG "\${CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG} -fno-common")

SET(CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG "\${CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG} -ffunction-sections")

SET(CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG "\${CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG} -fdata-sections")

SET(CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG "\${CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG} -ffreestanding")

SET(CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG "\${CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG} -fno-builtin")

SET(CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG "\${CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG} -mthumb")

SET(CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG "\${CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG} -mapcs")

SET(CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG "\${CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG} -Xlinker")

SET(CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG "\${CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG} --gc-sections")

SET(CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG "\${CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG} -Xlinker")

SET(CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG "\${CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG} -static")

SET(CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG "\${CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG} -Xlinker")

SET(CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG "\${CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG} -z")

SET(CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG "\${CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG} -Xlinker")

SET(CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG "\${CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG} muldefs")

**To**

SET(CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG "\${CMAKE\_EXE\_LINKER\_FLAGS\_DEBUG} --specs=rdimon.specs ")

**Remove**

target\_link\_libraries(semihosting\_ARMGCC.elf debug nosys)

2. Run "build\_debug.bat" to build project

### Step 3: Starting semihosting

(a) Download the image and set as follows.

```
cd D:\mcu-sdk-2.0-origin\boards\trkr64f120m\driver_examples\semihosting\armgcc\debug
d:
C:\PROGRA~2\GNUTOO~1\4BD65~1.920\bin\arm-none-eabi-gdb.exe
target remote localhost:2331
monitor reset
monitor semihosting enable
monitor semihosting thumbSWI 0xAB
monitor semihosting IOClient 1
monitor flash device = MK64FN1M0xxx12
load semihosting_ARMGCC.elf
monitor reg pc = (0x00000004)
monitor reg sp = (0x00000000)
continue
```

(b) After the setting, press "enter". The PuTTY window now shows the printf() output.

Semihosting is a mechanism for ARM targets to communicate input/output requests from application code to a host computer running a debugger. This mechanism can be used, for example, to enable functions in the C library, such as printf() and scanf(), to use the screen and keyboard of the host rather than having a screen and keyboard on the target system.

### 10.8.5 Guide Semihosting for IAR

**NOTE:** After the setting both "printf" and "scanf" are available for debugging.

#### Step 1: Setting up the environment

1. To set debugger options, choose Project>Options. In the Debugger category, click the Setup tab.
2. Select Run to main and click OK. This ensures that the debug session starts by running the main function.
3. The project is now ready to be built.

#### Step 2: Building the project

1. Compile and link the project by choosing Project>Make or F7.
2. Alternatively, click the Make button on the tool bar. The Make command compiles and links those files that have been modified.

#### Step 3: Starting semihosting

1. Choose "Semihosting\_IAR" project -> "Options" -> "Debugger" -> "J-Link/J-Trace".
2. Choose tab "J-Link/J-Trace" -> "Connection" tab -> "SWD".
3. Start the project by choosing Project>Download and Debug.
4. Choose View>Terminal I/O to display the output from the I/O operations.

## 10.8.6 Guide Semihosting for Keil $\mu$ Vision

**NOTE:** Semihosting is not support by MDK-ARM, use the retargeting functionality of MDK-ARM instead.

## 10.8.7 Guide Semihosting for ARMGCC

### Step 1: Setting up the environment

1. Turn on "J-LINK GDB Server" -> Select suitable "Target device" -> "OK".
2. Turn on "PuTTY". Set up as follows.
  - "Host Name (or IP address)" : localhost
  - "Port" :2333
  - "Connection type" : Telet.
  - Click "Open".
3. Increase "Heap/Stack" for GCC to 0x2000:

#### Add to "CMakeLists.txt"

```
SET(CMAKE_EXE_LINKER_FLAGS_RELEASE "${CMAKE_EXE_LINKER_FLAGS_RELEASE}
--defsym=__stack_size__=0x2000")

SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} --
defsym=__stack_size__=0x2000")

SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG} --
defsym=__heap_size__=0x2000")

SET(CMAKE_EXE_LINKER_FLAGS_RELEASE "${CMAKE_EXE_LINKER_FLAGS_RELEASE}
--defsym=__heap_size__=0x2000")
```

### Step 2: Building the project

1. Change "CMakeLists.txt":
 

**Change** "SET(CMAKE\_EXE\_LINKER\_FLAGS\_RELEASE "\${CMAKE\_EXE\_LINKER\_FLAGS\_RELEASE} -specs=nano.specs")"

**to** "SET(CMAKE\_EXE\_LINKER\_FLAGS\_RELEASE "\${CMAKE\_EXE\_LINKER\_FLAGS\_RELEASE} -specs=rdimon.specs")"

**Replace paragraph**

```
SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG}
-fno-common")
SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG}
-ffunction-sections")
SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG}
-fdata-sections")
SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBUG}
-ffreestanding")
```

## Semihosting

```
SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBU-
G} -fno-builtin")
SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBU-
G} -mthumb")
SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBU-
G} -mapcs")
SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBU-
G} -Xlinker")
SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBU-
G} --gc-sections")
SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBU-
G} -Xlinker")
SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBU-
G} -static")
SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBU-
G} -Xlinker")
SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBU-
G} -z")
SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBU-
G} -Xlinker")
SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBU-
G} muldefs")
```

### To

```
SET(CMAKE_EXE_LINKER_FLAGS_DEBUG "${CMAKE_EXE_LINKER_FLAGS_DEBU-
G} --specs=rdimon.specs ")
```

### Remove

```
target_link_libraries(semihosting_ARMGCC.elf debug nosys)
```

2. Run "build\_debug.bat" to build project

## Step 3: Starting semihosting

- (a) Download the image and set as follows.

```
cd D:\mcu-sdk-2.0-origin\boards\trkr64f120m\driver_examples\semihosting\armgcc\debug
d:
C:\PROGRA~2\GNUTOO~1\4BD65~1.920\bin\arm-none-eabi-gdb.exe
target remote localhost:2331
monitor reset
monitor semihosting enable
monitor semihosting thumbSWI 0xAB
monitor semihosting IOClient 1
monitor flash device = MK64FN1M0xxx12
load semihosting_ARMGCC.elf
monitor reg pc = (0x00000004)
monitor reg sp = (0x00000000)
continue
```

- (b) After the setting, press "enter". The PuTTY window now shows the printf() output.



## 10.9 SWO

Serial wire output is a mechanism for ARM targets to output signal from core through a single pin. Some IDEs also support SWO, such IAR and KEIL, both input and output are supported, see below for details.

### 10.9.1 Guide SWO for SDK

**NOTE:** After the setting both "printf" and "PRINTF" are available for debugging, JlinkSWOViewer can be used to capture the output log.

#### Step 1: Setting up the environment

1. Define SERIAL\_PORT\_TYPE\_SWO in your project settings.
2. Prepare code, the port and baudrate can be decided by application, clkSrcFreq should be mcu core clock frequency:

```
DbgConsole_Init(instance, baudRate, SERIAL_PORT_TYPE_SWO, clkSrcFreq);
```

3. Use PRINTF or printf to print some thing in application.

#### Step 2: Building the project

#### Step 3: Download and run project

#### 10.9.1.1 Guide SWO for IAR

**NOTE:** After the setting both "printf" and "scanf" are available for debugging.

#### Step 1: Setting up the environment

1. Choose project -> "Options" -> "Debugger" -> "J-Link/J-Trace".
2. Choose tab "J-Link/J-Trace" -> "Connection" tab -> "SWD".
3. Choose tab "General Options" -> "Library Configurations", select Semihosted, select Via SWO.
4. To configure the hardware's generation of trace data, click the SWO Configuration button available in the SWO Configuration dialog box. The value of the CPU clock option must reflect the frequency of the CPU clock speed at which the application executes. Note also that the settings you make are preserved between debug sessions. To decrease the amount of transmissions on the communication channel, you can disable the Timestamp option. Alternatively, set a lower rate for PC Sampling or use a higher SWO clock frequency.
5. Open the SWO Trace window from J-LINK, and click the Activate button to enable trace data collection.
6. There are three cases for this SDK\_DEBUGCONSOLE\_UART whether or not defined. a: if use uppercase PRINTF to output log, The SDK\_DEBUGCONSOLE\_UART defined or not defined will not effect debug function. b: if use lowercase printf to output log and defined SDK\_DEBUGCONSOLE\_UART to zero, then debug function ok. c: if use lowercase printf to output log and defined SDK\_DEBUGCONSOLE\_UART to one, then debug function ok.

## SWO

**NOTE:** Case a or c only apply at example which enable swo function,the SDK\_DEBUGCONSOLE\_UART definition in fsl\_debug\_console.h. For case a and c, Do and not do the above third step will be not affect function.

1. Start the project by choosing Project>Download and Debug.

### Step 2: Building the project

### Step 3: Starting swo

1. Download and debug application.
2. Choose View -> Terminal I/O to display the output from the I/O operations.
3. Run application.

## 10.9.2 Guide SWO for Keil µVision

**NOTE:** After the setting both "printf" and "scanf" are available for debugging.

### Step 1: Setting up the environment

1. There are three cases for this SDK\_DEBUGCONSOLE\_UART whether or not defined. a: if use uppercase PRINTF to output log,the SDK\_DEBUGCONSOLE\_UART definition does not affect the functionality and skip the second step directly. b: if use lowercase printf to output log and defined SDK\_DEBUGCONSOLE\_UART to zero,then start the second step. c: if use lowercase printf to output log and defined SDK\_DEBUGCONSOLE\_UART to one,then skip the second step directly.

**NOTE:** Case a or c only apply at example which enable swo function,the SDK\_DEBUGCONSOLE\_UART definition in fsl\_debug\_console.h.

1. In menu bar, click Management Run-Time Environment icon, select Compiler, unfold I/O, enable STDERR/STDIN/STDOUT and set the variant to ITM.
2. Open Project>Options for target or using Alt+F7 or click.
3. Select "Debug" tab, select "J-Link/J-Trace Cortex" and click "Setting button".
4. Select "Debug" tab and choose Port:SW, then select "Trace" tab, choose "Enable" and click O-K, please make sure the Core clock is set correctly, enable autodetect max SWO clk, enable ITM Stimulus Ports 0.

### Step 3: Building the project

1. Compile and link the project by choosing Project>Build Target or using F7.

### Step 4: Run the project

1. Choose "Debug" on menu bar or Ctrl F5.
2. In menu bar, choose "Serial Window" and click to "Debug (printf) Viewer".
3. Run line by line to see result in Console Window.

### 10.9.3 Guide SWO for MCUXpresso IDE

**NOTE:** MCUX support SWO for LPC-Link2 debug probe only.

### 10.9.4 Guide SWO for ARMGCC

**NOTE:** ARMGCC has no library support SWO.



## Chapter 11

# DMA: Direct Memory Access Controller Driver

### 11.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Direct Memory Access (DMA) of MCU-Xpresso SDK devices.

### 11.2 Typical use case

#### 11.2.1 DMA Operation

Refer to the driver examples codes located at `<SDK_ROOT>/boards/<BOARD>/driver_examples/dma`

### Files

- file [fsl\\_dma.h](#)

### Data Structures

- struct [dma\\_descriptor\\_t](#)  
*DMA descriptor structure. [More...](#)*
- struct [dma\\_xfercfg\\_t](#)  
*DMA transfer configuration. [More...](#)*
- struct [dma\\_channel\\_trigger\\_t](#)  
*DMA channel trigger. [More...](#)*
- struct [dma\\_channel\\_config\\_t](#)  
*DMA channel trigger. [More...](#)*
- struct [dma\\_transfer\\_config\\_t](#)  
*DMA transfer configuration. [More...](#)*
- struct [dma\\_handle\\_t](#)  
*DMA transfer handle structure. [More...](#)*

### Macros

- #define [DMA\\_MAX\\_TRANSFER\\_COUNT](#) 0x400  
*DMA max transfer size.*
- #define [FSL\\_FEATURE\\_DMA\\_NUMBER\\_OF\\_CHANNELS](#)<sub>n</sub>(x) FSL\_FEATURE\_DMA\_NUMBER\_OF\_CHANNELS  
*DMA channel numbers.*
- #define [FSL\\_FEATURE\\_DMA\\_LINK\\_DESCRIPTOR\\_ALIGN\\_SIZE](#) (16U)  
*DMA head link descriptor table align size.*
- #define [DMA\\_ALLOCATE\\_HEAD\\_DESCRIPTOR](#)(name, number) SDK\_ALIGN([dma\\_descriptor\\_t](#) name[number], FSL\_FEATURE\_DMA\_DESCRIPTOR\_ALIGN\_SIZE)  
*DMA head descriptor table allocate macro To simplify user interface, this macro will help allocate descriptor memory, user just need to provide the name and the number for the allocate descriptor.*

## Typical use case

- `#define DMA_ALLOCATE_HEAD_DESCRIPTOR_AT_NONCACHEABLE(name, number) AT_NONCACHEABLE_SECTION_ALIGN(dma_descriptor_t name[number], FSL_FEATURE_DMA_DESCRIPTOR_ALIGN_SIZE)`  
*DMA head descriptor table allocate macro at noncacheable part To simplify user interface, this macro will help allocate descriptor memory at noncacheable part, user just need to provide the name and the number for the allocate descriptor.*
- `#define DMA_ALLOCATE_LINK_DESCRIPTOR(name, number) SDK_ALIGN(dma_descriptor_t name[number], FSL_FEATURE_DMA_LINK_DESCRIPTOR_ALIGN_SIZE)`  
*DMA link descriptor table allocate macro To simplify user interface, this macro will help allocate descriptor memory, user just need to provide the name and the number for the allocate descriptor.*
- `#define DMA_ALLOCATE_LINK_DESCRIPTOR_AT_NONCACHEABLE(name, number) AT_NONCACHEABLE_SECTION_ALIGN(dma_descriptor_t name[number], FSL_FEATURE_DMA_LINK_DESCRIPTOR_ALIGN_SIZE)`  
*DMA link descriptor table allocate macro at noncacheable part To simplify user interface, this macro will help allocate descriptor memory at noncacheable part, user just need to provide the name and the number for the allocate descriptor.*
- `#define DMA_COMMON_REG_GET(base, channel, reg) (((volatile uint32_t *)&((base)->COMMON[0].reg)))[DMA_CHANNEL_GROUP(channel)]`  
*DMA linked descriptor address algin size.*
- `#define DMA_DESCRIPTOR_END_ADDRESS(start, inc, bytes, width) ((void *)((uint32_t)(start) + inc * bytes - inc * width))`  
*DMA descriptor end address calculate.*
- `#define DMA_CHANNEL_XFER(reload, clrTrig, intA, intB, width, srcInc, dstInc, bytes)`  
*DMA channel transfer configurations macro.*

## Typedefs

- `typedef void(* dma_callback )(struct _dma_handle *handle, void *userData, bool transferDone, uint32_t intmode)`  
*Define Callback function for DMA.*

## Enumerations

- `enum _dma_transfer_status { kStatus_DMA_Busy = MAKE_STATUS(kStatusGroup_DMA, 0) }`  
*DMA transfer status.*
- `enum _dma_addr_interleave_size { kDMA_AddressInterleave0xWidth = 0U, kDMA_AddressInterleave1xWidth = 1U, kDMA_AddressInterleave2xWidth = 2U, kDMA_AddressInterleave4xWidth = 4U }`  
*dma address interleave size*
- `enum _dma_transfer_width { kDMA_Transfer8BitWidth = 1U, kDMA_Transfer16BitWidth = 2U, kDMA_Transfer32BitWidth = 4U }`  
*dma transfer width*
- `enum dma_priority_t {`

```

kDMA_ChannelPriority0 = 0,
kDMA_ChannelPriority1,
kDMA_ChannelPriority2,
kDMA_ChannelPriority3,
kDMA_ChannelPriority4,
kDMA_ChannelPriority5,
kDMA_ChannelPriority6,
kDMA_ChannelPriority7 }

```

*DMA channel priority.*

- enum `dma_irq_t` {  
`kDMA_IntA`,  
`kDMA_IntB`,  
`kDMA_IntError` }

*DMA interrupt flags.*

- enum `dma_trigger_type_t` {  
`kDMA_NoTrigger` = 0,  
`kDMA_LowLevelTrigger` = `DMA_CHANNEL_CFG_HWTRIGEN(1) | DMA_CHANNEL_CFG-_TRIGTYPE(1)`,  
`kDMA_HighLevelTrigger`,  
`kDMA_FallingEdgeTrigger` = `DMA_CHANNEL_CFG_HWTRIGEN(1)`,  
`kDMA_RisingEdgeTrigger` }

*DMA trigger type.*

- enum `_dma_burst_size` {  
`kDMA_BurstSize1` = 0U,  
`kDMA_BurstSize2` = 1U,  
`kDMA_BurstSize4` = 2U,  
`kDMA_BurstSize8` = 3U,  
`kDMA_BurstSize16` = 4U,  
`kDMA_BurstSize32` = 5U,  
`kDMA_BurstSize64` = 6U,  
`kDMA_BurstSize128` = 7U,  
`kDMA_BurstSize256` = 8U,  
`kDMA_BurstSize512` = 9U,  
`kDMA_BurstSize1024` = 10U }

*DMA burst size.*

- enum `dma_trigger_burst_t` {

## Typical use case

```
kDMA_SingleTransfer = 0,
kDMA_LevelBurstTransfer = DMA_CHANNEL_CFG_TRIGBURST(1),
kDMA_EdgeBurstTransfer1 = DMA_CHANNEL_CFG_TRIGBURST(1),
kDMA_EdgeBurstTransfer2,
kDMA_EdgeBurstTransfer4,
kDMA_EdgeBurstTransfer8,
kDMA_EdgeBurstTransfer16,
kDMA_EdgeBurstTransfer32,
kDMA_EdgeBurstTransfer64,
kDMA_EdgeBurstTransfer128,
kDMA_EdgeBurstTransfer256,
kDMA_EdgeBurstTransfer512,
kDMA_EdgeBurstTransfer1024 }
 DMA trigger burst.
• enum dma_burst_wrap_t {
 kDMA_NoWrap = 0,
 kDMA_SrcWrap = DMA_CHANNEL_CFG_SRCBURSTWRAP(1),
 kDMA_DstWrap = DMA_CHANNEL_CFG_DSTBURSTWRAP(1),
 kDMA_SrcAndDstWrap }
 DMA burst wrapping.
• enum dma_transfer_type_t {
 kDMA_MemoryToMemory = 0x0U,
 kDMA_PeripheralToMemory,
 kDMA_MemoryToPeripheral,
 kDMA_StaticToStatic }
 DMA transfer type.
```

## Driver version

- #define **FSL\_DMA\_DRIVER\_VERSION** (MAKE\_VERSION(2, 4, 0))  
 *DMA driver version.*

## DMA initialization and De-initialization

- void **DMA\_Init** (DMA\_Type \*base)  
 *Initializes DMA peripheral.*
- void **DMA\_Deinit** (DMA\_Type \*base)  
 *Deinitializes DMA peripheral.*
- void **DMA\_InstallDescriptorMemory** (DMA\_Type \*base, void \*addr)  
 *Install DMA descriptor memory.*

## DMA Channel Operation

- static bool **DMA\_ChannelIsActive** (DMA\_Type \*base, uint32\_t channel)  
 *Return whether DMA channel is processing transfer.*
- static bool **DMA\_ChannelIsBusy** (DMA\_Type \*base, uint32\_t channel)  
 *Return whether DMA channel is busy.*
- static void **DMA\_EnableChannelInterrupts** (DMA\_Type \*base, uint32\_t channel)



- *Enables the interrupt source for the DMA transfer.*  
static void [DMA\\_DisableChannelInterrupts](#) (DMA\_Type \*base, uint32\_t channel)
- *Disables the interrupt source for the DMA transfer.*  
static void [DMA\\_EnableChannel](#) (DMA\_Type \*base, uint32\_t channel)
- *Enable DMA channel.*  
static void [DMA\\_DisableChannel](#) (DMA\_Type \*base, uint32\_t channel)
- *Disable DMA channel.*  
static void [DMA\\_EnableChannelPeriphRq](#) (DMA\_Type \*base, uint32\_t channel)
- *Set PERIPHREQEN of channel configuration register.*  
static void [DMA\\_DisableChannelPeriphRq](#) (DMA\_Type \*base, uint32\_t channel)
- *Get PERIPHREQEN value of channel configuration register.*  
void [DMA\\_ConfigureChannelTrigger](#) (DMA\_Type \*base, uint32\_t channel, [dma\\_channel\\_trigger\\_t](#) \*trigger)
- *Set trigger settings of DMA channel.*  
void [DMA\\_SetChannelConfig](#) (DMA\_Type \*base, uint32\_t channel, [dma\\_channel\\_trigger\\_t](#) \*trigger, bool isPeriph)
- *set channel config.*  
uint32\_t [DMA\\_GetRemainingBytes](#) (DMA\_Type \*base, uint32\_t channel)
- *Gets the remaining bytes of the current DMA descriptor transfer.*  
static void [DMA\\_SetChannelPriority](#) (DMA\_Type \*base, uint32\_t channel, [dma\\_priority\\_t](#) priority)
- *Set priority of channel configuration register.*  
static [dma\\_priority\\_t](#) [DMA\\_GetChannelPriority](#) (DMA\_Type \*base, uint32\_t channel)
- *Get priority of channel configuration register.*  
static void [DMA\\_SetChannelConfigValid](#) (DMA\_Type \*base, uint32\_t channel)
- *Set channel configuration valid.*  
static void [DMA\\_DoChannelSoftwareTrigger](#) (DMA\_Type \*base, uint32\_t channel)
- *Do software trigger for the channel.*  
static void [DMA\\_LoadChannelTransferConfig](#) (DMA\_Type \*base, uint32\_t channel, uint32\_t xfer)
- *Load channel transfer configurations.*  
void [DMA\\_CreateDescriptor](#) ([dma\\_descriptor\\_t](#) \*desc, [dma\\_xfercfg\\_t](#) \*xfercfg, void \*srcAddr, void \*dstAddr, void \*nextDesc)
- *Create application specific DMA descriptor to be used in a chain in transfer.*  
void [DMA\\_SetupDescriptor](#) ([dma\\_descriptor\\_t](#) \*desc, uint32\_t xfercfg, void \*srcStartAddr, void \*dstStartAddr, void \*nextDesc)
- *setup dma descriptor*  
void [DMA\\_SetupChannelDescriptor](#) ([dma\\_descriptor\\_t](#) \*desc, uint32\_t xfercfg, void \*srcStartAddr, void \*dstStartAddr, void \*nextDesc, [dma\\_burst\\_wrap\\_t](#) wrapType, uint32\_t burstSize)
- *setup dma channel descriptor*  
void [DMA\\_LoadChannelDescriptor](#) (DMA\_Type \*base, uint32\_t channel, [dma\\_descriptor\\_t](#) \*descriptor)
- *load channel transfer decriptor.*

## DMA Transactional Operation

- void [DMA\\_AbortTransfer](#) ([dma\\_handle\\_t](#) \*handle)
- *Abort running transfer by handle.*
- void [DMA\\_CreateHandle](#) ([dma\\_handle\\_t](#) \*handle, DMA\_Type \*base, uint32\_t channel)
- *Creates the DMA handle.*
- void [DMA\\_SetCallback](#) ([dma\\_handle\\_t](#) \*handle, [dma\\_callback](#) callback, void \*userData)
- *Installs a callback function for the DMA transfer.*

## Data Structure Documentation

- void [DMA\\_PrepareTransfer](#) ([dma\\_transfer\\_config\\_t](#) \*config, void \*srcAddr, void \*dstAddr, uint32\_t byteWidth, uint32\_t transferBytes, [dma\\_transfer\\_type\\_t](#) type, void \*nextDesc)  
*Prepares the DMA transfer structure.*
- void [DMA\\_PrepareChannelTransfer](#) ([dma\\_channel\\_config\\_t](#) \*config, void \*srcStartAddr, void \*dstStartAddr, uint32\_t xferCfg, [dma\\_transfer\\_type\\_t](#) type, [dma\\_channel\\_trigger\\_t](#) \*trigger, void \*nextDesc)  
*Prepare channel transfer configurations.*
- [status\\_t](#) [DMA\\_SubmitTransfer](#) ([dma\\_handle\\_t](#) \*handle, [dma\\_transfer\\_config\\_t](#) \*config)  
*Submits the DMA transfer request.*
- void [DMA\\_SubmitChannelTransferParameter](#) ([dma\\_handle\\_t](#) \*handle, uint32\_t xfercfg, void \*srcStartAddr, void \*dstStartAddr, void \*nextDesc)  
*Submit channel transfer paramter directly.*
- void [DMA\\_SubmitChannelDescriptor](#) ([dma\\_handle\\_t](#) \*handle, [dma\\_descriptor\\_t](#) \*descriptor)  
*Submit channel descriptor.*
- [status\\_t](#) [DMA\\_SubmitChannelTransfer](#) ([dma\\_handle\\_t](#) \*handle, [dma\\_channel\\_config\\_t](#) \*config)  
*Submits the DMA channel transfer request.*
- void [DMA\\_StartTransfer](#) ([dma\\_handle\\_t](#) \*handle)  
*DMA start transfer.*
- void [DMA\\_IRQHandle](#) ([DMA\\_Type](#) \*base)  
*DMA IRQ handler for descriptor transfer complete.*

## 11.3 Data Structure Documentation

### 11.3.1 struct dma\_descriptor\_t

#### Data Fields

- volatile uint32\_t [xfercfg](#)  
*Transfer configuration.*
- void \* [srcEndAddr](#)  
*Last source address of DMA transfer.*
- void \* [dstEndAddr](#)  
*Last destination address of DMA transfer.*
- void \* [linkToNextDesc](#)  
*Address of next DMA descriptor in chain.*

### 11.3.2 struct dma\_xfercfg\_t

#### Data Fields

- bool [valid](#)  
*Descriptor is ready to transfer.*
- bool [reload](#)  
*Reload channel configuration register after current descriptor is exhausted.*
- bool [swtrig](#)  
*Perform software trigger.*
- bool [clrtrig](#)

- *Clear trigger.*
- bool [intA](#)  
*Raises IRQ when transfer is done and set IRQA status register flag.*
- bool [intB](#)  
*Raises IRQ when transfer is done and set IRQB status register flag.*
- uint8\_t [byteWidth](#)  
*Byte width of data to transfer.*
- uint8\_t [srcInc](#)  
*Increment source address by 'srcInc' x 'byteWidth'.*
- uint8\_t [dstInc](#)  
*Increment destination address by 'dstInc' x 'byteWidth'.*
- uint16\_t [transferCount](#)  
*Number of transfers.*

#### 11.3.2.0.0.4 Field Documentation

##### 11.3.2.0.0.4.1 bool dma\_xfercfg\_t::swtrig

Transfer if fired when 'valid' is set

### 11.3.3 struct dma\_channel\_trigger\_t

#### Data Fields

- [dma\\_trigger\\_type\\_t](#) type  
*Select hardware trigger as edge triggered or level triggered.*
- [dma\\_trigger\\_burst\\_t](#) burst  
*Select whether hardware triggers cause a single or burst transfer.*
- [dma\\_burst\\_wrap\\_t](#) wrap  
*Select wrap type, source wrap or dest wrap, or both.*

#### 11.3.3.0.0.5 Field Documentation

##### 11.3.3.0.0.5.1 dma\_trigger\_type\_t dma\_channel\_trigger\_t::type

##### 11.3.3.0.0.5.2 dma\_trigger\_burst\_t dma\_channel\_trigger\_t::burst

##### 11.3.3.0.0.5.3 dma\_burst\_wrap\_t dma\_channel\_trigger\_t::wrap

### 11.3.4 struct dma\_channel\_config\_t

#### Data Fields

- void \* [srcStartAddr](#)  
*Source data address.*
- void \* [dstStartAddr](#)  
*Destination data address.*
- void \* [nextDesc](#)

## Macro Definition Documentation

- *Chain custom descriptor.*  
uint32\_t [xferCfg](#)  
*channel transfer configurations*
- [dma\\_channel\\_trigger\\_t](#) \* [trigger](#)  
*DMA trigger type.*
- bool [isPeriph](#)  
*select the request type*

### 11.3.5 struct dma\_transfer\_config\_t

#### Data Fields

- uint8\_t \* [srcAddr](#)  
*Source data address.*
- uint8\_t \* [dstAddr](#)  
*Destination data address.*
- uint8\_t \* [nextDesc](#)  
*Chain custom descriptor.*
- [dma\\_xfercfg\\_t](#) [xfercfg](#)  
*Transfer options.*
- bool [isPeriph](#)  
*DMA transfer is driven by peripheral.*

### 11.3.6 struct dma\_handle\_t

#### Data Fields

- [dma\\_callback](#) [callback](#)  
*Callback function.*
- void \* [userData](#)  
*Callback function parameter.*
- DMA\_Type \* [base](#)  
*DMA peripheral base address.*
- uint8\_t [channel](#)  
*DMA channel number.*

#### 11.3.6.0.0.6 Field Documentation

##### 11.3.6.0.0.6.1 dma\_callback dma\_handle\_t::callback

Invoked when transfer of descriptor with interrupt flag finishes

## 11.4 Macro Definition Documentation

### 11.4.1 #define FSL\_DMA\_DRIVER\_VERSION (MAKE\_VERSION(2, 4, 0))

Version 2.4.0.

**11.4.2** `#define DMA_ALLOCATE_HEAD_DESCRIPTOR( name,  
number ) SDK_ALIGN(dma_descriptor_t name[number],  
FSL_FEATURE_DMA_DESCRIPTOR_ALIGN_SIZE)`

## Macro Definition Documentation

### Parameters

|                      |                                |
|----------------------|--------------------------------|
| <i>name,allocate</i> | decriptor name.                |
| <i>number,number</i> | of descriptor to be allocated. |

**11.4.3 #define DMA\_ALLOCATE\_HEAD\_DESCRIPTOR\_AT\_NONCACHEABLE(  
    *name, number* ) AT\_NONCACHEABLE\_SECTION\_ALIGN(dma\_descriptor\_t  
    name[number], FSL\_FEATURE\_DMA\_DESCRIPTOR\_ALIGN\_SIZE)**

### Parameters

|                      |                                |
|----------------------|--------------------------------|
| <i>name,allocate</i> | decriptor name.                |
| <i>number,number</i> | of descriptor to be allocated. |

**11.4.4 #define DMA\_ALLOCATE\_LINK\_DESCRIPTOR( *name*,  
    *number* ) SDK\_ALIGN(dma\_descriptor\_t name[number],  
    FSL\_FEATURE\_DMA\_LINK\_DESCRIPTOR\_ALIGN\_SIZE)**

### Parameters

|                      |                                |
|----------------------|--------------------------------|
| <i>name,allocate</i> | decriptor name.                |
| <i>number,number</i> | of descriptor to be allocated. |

**11.4.5 #define DMA\_ALLOCATE\_LINK\_DESCRIPTOR\_AT\_NONCACHEABLE(  
    *name, number* ) AT\_NONCACHEABLE\_SECTION\_ALIGN(dma\_descriptor\_t  
    name[number], FSL\_FEATURE\_DMA\_LINK\_DESCRIPTOR\_ALIGN\_SI-  
    ZE)**

### Parameters

|                      |                                |
|----------------------|--------------------------------|
| <i>name,allocate</i> | decriptor name.                |
| <i>number,number</i> | of descriptor to be allocated. |

**11.4.6 #define DMA\_DESCRIPTOR\_END\_ADDRESS( *start, inc, bytes, width*  
    ) ((void \*)((uint32\_t)(start) + inc \* bytes - inc \* width))**

## Parameters

|                        |                 |
|------------------------|-----------------|
| <i>start, start</i>    | address         |
| <i>inc, address</i>    | interleave size |
| <i>bytes, transfer</i> | bytes           |
| <i>width, transfer</i> | width           |

## 11.4.7 #define DMA\_CHANNEL\_XFER( reload, clrTrig, intA, intB, width, srcInc, dstInc, bytes )

### Value:

```

DMA_CHANNEL_XFERCFG_CFGVALID_MASK | DMA_CHANNEL_XFERCFG_RELOAD(reload) | DMA_CHANNEL_XFERCFG_CLRTRIG(
 clrTrig) | \
 DMA_CHANNEL_XFERCFG_SETINTA(intA) | DMA_CHANNEL_XFERCFG_SETINTB(intB) |
 DMA_CHANNEL_XFERCFG_WIDTH(width == 4 ? 2 : (width - 1)) |
 DMA_CHANNEL_XFERCFG_SRCINC(srcInc == 4 ? (srcInc - 1) : srcInc) |
 DMA_CHANNEL_XFERCFG_DSTINC(dstInc == 4 ? (dstInc - 1) : dstInc) |
 DMA_CHANNEL_XFERCFG_XFERCOUNT(bytes / width - 1)

```

## Parameters

|                            |                                                               |
|----------------------------|---------------------------------------------------------------|
| <i>reload, true</i>        | is reload link descriptor after current exhaust, false is not |
| <i>clrTrig, true</i>       | is clear trigger status, wait software trigger, false is not  |
| <i>intA, enable</i>        | interruptA                                                    |
| <i>intB, enable</i>        | interruptB                                                    |
| <i>width, transfer</i>     | width                                                         |
| <i>srcInc, source</i>      | address interleave size                                       |
| <i>dstInc, destination</i> | address interleave size                                       |
| <i>bytes, transfer</i>     | bytes                                                         |

## 11.5 Typedef Documentation

### 11.5.1 typedef void(\* dma\_callback)(struct \_dma\_handle \*handle, void \*userData, bool transferDone, uint32\_t intmode)

### 11.6 Enumeration Type Documentation

#### 11.6.1 enum \_dma\_transfer\_status

Enumerator

*kStatus\_DMA\_Busy* Channel is busy and can't handle the transfer request.

#### 11.6.2 enum \_dma\_addr\_interleave\_size

Enumerator

*kDMA\_AddressInterleave0xWidth* dma source/destination address no interleave  
*kDMA\_AddressInterleave1xWidth* dma source/destination address interleave 1xwidth  
*kDMA\_AddressInterleave2xWidth* dma source/destination address interleave 2xwidth  
*kDMA\_AddressInterleave4xWidth* dma source/destination address interleave 3xwidth

#### 11.6.3 enum \_dma\_transfer\_width

Enumerator

*kDMA\_Transfer8BitWidth* dma channel transfer bit width is 8 bit  
*kDMA\_Transfer16BitWidth* dma channel transfer bit width is 16 bit  
*kDMA\_Transfer32BitWidth* dma channel transfer bit width is 32 bit

#### 11.6.4 enum dma\_priority\_t

Enumerator

*kDMA\_ChannelPriority0* Highest channel priority - priority 0.  
*kDMA\_ChannelPriority1* Channel priority 1.  
*kDMA\_ChannelPriority2* Channel priority 2.  
*kDMA\_ChannelPriority3* Channel priority 3.  
*kDMA\_ChannelPriority4* Channel priority 4.  
*kDMA\_ChannelPriority5* Channel priority 5.  
*kDMA\_ChannelPriority6* Channel priority 6.  
*kDMA\_ChannelPriority7* Lowest channel priority - priority 7.



### 11.6.5 enum dma\_irq\_t

Enumerator

***kDMA\_IntA*** DMA interrupt flag A.  
***kDMA\_IntB*** DMA interrupt flag B.  
***kDMA\_IntError*** DMA interrupt flag error.

### 11.6.6 enum dma\_trigger\_type\_t

Enumerator

***kDMA\_NoTrigger*** Trigger is disabled.  
***kDMA\_LowLevelTrigger*** Low level active trigger.  
***kDMA\_HighLevelTrigger*** High level active trigger.  
***kDMA\_FallingEdgeTrigger*** Falling edge active trigger.  
***kDMA\_RisingEdgeTrigger*** Rising edge active trigger.

### 11.6.7 enum \_dma\_burst\_size

Enumerator

***kDMA\_BurstSize1*** burst size 1 transfer  
***kDMA\_BurstSize2*** burst size 2 transfer  
***kDMA\_BurstSize4*** burst size 4 transfer  
***kDMA\_BurstSize8*** burst size 8 transfer  
***kDMA\_BurstSize16*** burst size 16 transfer  
***kDMA\_BurstSize32*** burst size 32 transfer  
***kDMA\_BurstSize64*** burst size 64 transfer  
***kDMA\_BurstSize128*** burst size 128 transfer  
***kDMA\_BurstSize256*** burst size 256 transfer  
***kDMA\_BurstSize512*** burst size 512 transfer  
***kDMA\_BurstSize1024*** burst size 1024 transfer

### 11.6.8 enum dma\_trigger\_burst\_t

Enumerator

***kDMA\_SingleTransfer*** Single transfer.  
***kDMA\_LevelBurstTransfer*** Burst transfer driven by level trigger.  
***kDMA\_EdgeBurstTransfer1*** Perform 1 transfer by edge trigger.  
***kDMA\_EdgeBurstTransfer2*** Perform 2 transfers by edge trigger.

## Function Documentation

***kDMA\_EdgeBurstTransfer4*** Perform 4 transfers by edge trigger.  
***kDMA\_EdgeBurstTransfer8*** Perform 8 transfers by edge trigger.  
***kDMA\_EdgeBurstTransfer16*** Perform 16 transfers by edge trigger.  
***kDMA\_EdgeBurstTransfer32*** Perform 32 transfers by edge trigger.  
***kDMA\_EdgeBurstTransfer64*** Perform 64 transfers by edge trigger.  
***kDMA\_EdgeBurstTransfer128*** Perform 128 transfers by edge trigger.  
***kDMA\_EdgeBurstTransfer256*** Perform 256 transfers by edge trigger.  
***kDMA\_EdgeBurstTransfer512*** Perform 512 transfers by edge trigger.  
***kDMA\_EdgeBurstTransfer1024*** Perform 1024 transfers by edge trigger.

### 11.6.9 enum dma\_burst\_wrap\_t

Enumerator

***kDMA\_NoWrap*** Wrapping is disabled.  
***kDMA\_SrcWrap*** Wrapping is enabled for source.  
***kDMA\_DstWrap*** Wrapping is enabled for destination.  
***kDMA\_SrcAndDstWrap*** Wrapping is enabled for source and destination.

### 11.6.10 enum dma\_transfer\_type\_t

Enumerator

***kDMA\_MemoryToMemory*** Transfer from memory to memory (increment source and destination)  
***kDMA\_PeripheralToMemory*** Transfer from peripheral to memory (increment only destination)  
***kDMA\_MemoryToPeripheral*** Transfer from memory to peripheral (increment only source)  
***kDMA\_StaticToStatic*** Peripheral to static memory (do not increment source or destination)

## 11.7 Function Documentation

### 11.7.1 void DMA\_Init ( DMA\_Type \* *base* )

This function enable the DMA clock, set descriptor table and enable DMA peripheral.

Parameters

|             |                              |
|-------------|------------------------------|
| <i>base</i> | DMA peripheral base address. |
|-------------|------------------------------|

### 11.7.2 void DMA\_Deinit ( DMA\_Type \* *base* )

This function gates the DMA clock.

## Parameters

|             |                              |
|-------------|------------------------------|
| <i>base</i> | DMA peripheral base address. |
|-------------|------------------------------|

### 11.7.3 void DMA\_InstallDescriptorMemory ( DMA\_Type \* *base*, void \* *addr* )

This function used to register DMA descriptor memory for linked transfer, a typical case is ping pong transfer which will request more than one DMA descriptor memory space, although current DMA driver has a default DMA descriptor buffer, but it support one DMA descriptor for one channel only.

## Parameters

|             |                        |
|-------------|------------------------|
| <i>base</i> | DMA base address.      |
| <i>addr</i> | DMA descriptor address |

### 11.7.4 static bool DMA\_ChannelsActive ( DMA\_Type \* *base*, uint32\_t *channel* ) [inline], [static]

## Parameters

|                |                              |
|----------------|------------------------------|
| <i>base</i>    | DMA peripheral base address. |
| <i>channel</i> | DMA channel number.          |

## Returns

True for active state, false otherwise.

### 11.7.5 static bool DMA\_ChannelsBusy ( DMA\_Type \* *base*, uint32\_t *channel* ) [inline], [static]

## Parameters

|             |                              |
|-------------|------------------------------|
| <i>base</i> | DMA peripheral base address. |
|-------------|------------------------------|

## Function Documentation

|                |                     |
|----------------|---------------------|
| <i>channel</i> | DMA channel number. |
|----------------|---------------------|

Returns

True for busy state, false otherwise.

**11.7.6 static void DMA\_EnableChannelInterrupts ( DMA\_Type \* *base*, uint32\_t *channel* ) [inline], [static]**

Parameters

|                |                              |
|----------------|------------------------------|
| <i>base</i>    | DMA peripheral base address. |
| <i>channel</i> | DMA channel number.          |

**11.7.7 static void DMA\_DisableChannelInterrupts ( DMA\_Type \* *base*, uint32\_t *channel* ) [inline], [static]**

Parameters

|                |                              |
|----------------|------------------------------|
| <i>base</i>    | DMA peripheral base address. |
| <i>channel</i> | DMA channel number.          |

**11.7.8 static void DMA\_EnableChannel ( DMA\_Type \* *base*, uint32\_t *channel* ) [inline], [static]**

Parameters

|                |                              |
|----------------|------------------------------|
| <i>base</i>    | DMA peripheral base address. |
| <i>channel</i> | DMA channel number.          |

**11.7.9 static void DMA\_DisableChannel ( DMA\_Type \* *base*, uint32\_t *channel* ) [inline], [static]**

## Parameters

|                |                              |
|----------------|------------------------------|
| <i>base</i>    | DMA peripheral base address. |
| <i>channel</i> | DMA channel number.          |

**11.7.10 static void DMA\_EnableChannelPeriphRq ( DMA\_Type \* *base*, uint32\_t *channel* ) [inline], [static]**

## Parameters

|                |                              |
|----------------|------------------------------|
| <i>base</i>    | DMA peripheral base address. |
| <i>channel</i> | DMA channel number.          |

**11.7.11 static void DMA\_DisableChannelPeriphRq ( DMA\_Type \* *base*, uint32\_t *channel* ) [inline], [static]**

## Parameters

|                |                              |
|----------------|------------------------------|
| <i>base</i>    | DMA peripheral base address. |
| <i>channel</i> | DMA channel number.          |

## Returns

True for enabled PeriphRq, false for disabled.

**11.7.12 void DMA\_ConfigureChannelTrigger ( DMA\_Type \* *base*, uint32\_t *channel*, dma\_channel\_trigger\_t \* *trigger* )**

## Parameters

|                |                              |
|----------------|------------------------------|
| <i>base</i>    | DMA peripheral base address. |
| <i>channel</i> | DMA channel number.          |
| <i>trigger</i> | trigger configuration.       |

---

## Function Documentation

**11.7.13** void DMA\_SetChannelConfig ( DMA\_Type \* *base*, uint32\_t *channel*, dma\_channel\_trigger\_t \* *trigger*, bool *isPeriph* )

This function provide a interface to configure channel configuration registers.

## Parameters

|                 |                                       |
|-----------------|---------------------------------------|
| <i>base</i>     | DMA base address.                     |
| <i>channel</i>  | DMA channel number.                   |
| <i>trigger</i>  | channel configurations structure.     |
| <i>isPeriph</i> | true is periph request, false is not. |

**11.7.14 uint32\_t DMA\_GetRemainingBytes ( DMA\_Type \* *base*, uint32\_t *channel* )**

## Parameters

|                |                              |
|----------------|------------------------------|
| <i>base</i>    | DMA peripheral base address. |
| <i>channel</i> | DMA channel number.          |

## Returns

The number of bytes which have not been transferred yet.

**11.7.15 static void DMA\_SetChannelPriority ( DMA\_Type \* *base*, uint32\_t *channel*, dma\_priority\_t *priority* ) [inline], [static]**

## Parameters

|                 |                              |
|-----------------|------------------------------|
| <i>base</i>     | DMA peripheral base address. |
| <i>channel</i>  | DMA channel number.          |
| <i>priority</i> | Channel priority value.      |

**11.7.16 static dma\_priority\_t DMA\_GetChannelPriority ( DMA\_Type \* *base*, uint32\_t *channel* ) [inline], [static]**

## Parameters

|                |                              |
|----------------|------------------------------|
| <i>base</i>    | DMA peripheral base address. |
| <i>channel</i> | DMA channel number.          |

## Function Documentation

### Returns

Channel priority value.

**11.7.17 static void DMA\_SetChannelConfigValid ( DMA\_Type \* *base*, uint32\_t *channel* ) [inline], [static]**

### Parameters

|                |                              |
|----------------|------------------------------|
| <i>base</i>    | DMA peripheral base address. |
| <i>channel</i> | DMA channel number.          |

**11.7.18 static void DMA\_DoChannelSoftwareTrigger ( DMA\_Type \* *base*, uint32\_t *channel* ) [inline], [static]**

### Parameters

|                |                              |
|----------------|------------------------------|
| <i>base</i>    | DMA peripheral base address. |
| <i>channel</i> | DMA channel number.          |

**11.7.19 static void DMA\_LoadChannelTransferConfig ( DMA\_Type \* *base*, uint32\_t *channel*, uint32\_t *xfer* ) [inline], [static]**

### Parameters

|                |                              |
|----------------|------------------------------|
| <i>base</i>    | DMA peripheral base address. |
| <i>channel</i> | DMA channel number.          |
| <i>xfer</i>    | transfer configurations.     |

**11.7.20 void DMA\_CreateDescriptor ( dma\_descriptor\_t \* *desc*, dma\_xfercfg\_t \* *xfercfg*, void \* *srcAddr*, void \* *dstAddr*, void \* *nextDesc* )**



## Parameters

|                 |                                            |
|-----------------|--------------------------------------------|
| <i>desc</i>     | DMA descriptor address.                    |
| <i>xfercfg</i>  | Transfer configuration for DMA descriptor. |
| <i>srcAddr</i>  | Address of last item to transmit           |
| <i>dstAddr</i>  | Address of last item to receive.           |
| <i>nextDesc</i> | Address of next descriptor in chain.       |

### 11.7.21 void DMA\_SetupDescriptor ( dma\_descriptor\_t \* *desc*, uint32\_t *xfercfg*, void \* *srcStartAddr*, void \* *dstStartAddr*, void \* *nextDesc* )

Note: This function do not support configure wrap descriptor.

## Parameters

|                     |                                            |
|---------------------|--------------------------------------------|
| <i>desc</i>         | DMA descriptor address.                    |
| <i>xfercfg</i>      | Transfer configuration for DMA descriptor. |
| <i>srcStartAddr</i> | Start address of source address.           |
| <i>dstStartAddr</i> | Start address of destination address.      |
| <i>nextDesc</i>     | Address of next descriptor in chain.       |

### 11.7.22 void DMA\_SetupChannelDescriptor ( dma\_descriptor\_t \* *desc*, uint32\_t *xfercfg*, void \* *srcStartAddr*, void \* *dstStartAddr*, void \* *nextDesc*, dma\_burst\_wrap\_t *wrapType*, uint32\_t *burstSize* )

Note: This function support configure wrap descriptor.

## Parameters

|                     |                                                      |
|---------------------|------------------------------------------------------|
| <i>desc</i>         | DMA descriptor address.                              |
| <i>xfercfg</i>      | Transfer configuration for DMA descriptor.           |
| <i>srcStartAddr</i> | Start address of source address.                     |
| <i>dstStartAddr</i> | Start address of destination address.                |
| <i>nextDesc</i>     | Address of next descriptor in chain.                 |
| <i>wrapType</i>     | burst wrap type.                                     |
| <i>burstSize</i>    | burst size, reference <code>_dma_burst_size</code> . |

## Function Documentation

### 11.7.23 void DMA\_LoadChannelDescriptor ( DMA\_Type \* *base*, uint32\_t *channel*, dma\_descriptor\_t \* *descriptor* )

This function can be used to load descriptor to driver internal channel descriptor that is used to start DMA transfer, the head descriptor table is defined in DMA driver, it is useful for the case:

1. for the polling transfer, application can allocate a local descriptor memory table to prepare a descriptor firstly and then call this api to load the configured descriptor to driver descriptor table.

```
* DMA_Init(DMA0);
* DMA_EnableChannel(DMA0, DEMO_DMA_CHANNEL);
* DMA_SetupDescriptor(desc, xferCfg, s_srcBuffer, &s_destBuffer[0], NULL);
* DMA_LoadChannelDescriptor(DMA0, DEMO_DMA_CHANNEL, (
 dma_descriptor_t *)desc);
* DMA_DoChannelSoftwareTrigger(DMA0, DEMO_DMA_CHANNEL);
* while(DMA_ChannelIsBusy(DMA0, DEMO_DMA_CHANNEL))
* {}
*
```

#### Parameters

|                   |                            |
|-------------------|----------------------------|
| <i>base</i>       | DMA base address.          |
| <i>channel</i>    | DMA channel.               |
| <i>descriptor</i> | configured DMA descriptor. |

### 11.7.24 void DMA\_AbortTransfer ( dma\_handle\_t \* *handle* )

This function aborts DMA transfer specified by handle.

#### Parameters

|               |                     |
|---------------|---------------------|
| <i>handle</i> | DMA handle pointer. |
|---------------|---------------------|

### 11.7.25 void DMA\_CreateHandle ( dma\_handle\_t \* *handle*, DMA\_Type \* *base*, uint32\_t *channel* )

This function is called if using transaction API for DMA. This function initializes the internal state of DMA handle.

#### Parameters

|                |                                                                             |
|----------------|-----------------------------------------------------------------------------|
| <i>handle</i>  | DMA handle pointer. The DMA handle stores callback function and parameters. |
| <i>base</i>    | DMA peripheral base address.                                                |
| <i>channel</i> | DMA channel number.                                                         |

**11.7.26** `void DMA_SetCallback ( dma_handle_t * handle, dma_callback callback,  
void * userData )`

This callback is called in DMA IRQ handler. Use the callback to do something after the current major loop transfer completes.

## Function Documentation

### Parameters

|                 |                                  |
|-----------------|----------------------------------|
| <i>handle</i>   | DMA handle pointer.              |
| <i>callback</i> | DMA callback function pointer.   |
| <i>userData</i> | Parameter for callback function. |

**11.7.27 void DMA\_PrepareTransfer ( dma\_transfer\_config\_t \* *config*, void \* *srcAddr*, void \* *dstAddr*, uint32\_t *byteWidth*, uint32\_t *transferBytes*, dma\_transfer\_type\_t *type*, void \* *nextDesc* )**

### Parameters

|                      |                                                          |
|----------------------|----------------------------------------------------------|
| <i>config</i>        | The user configuration structure of type dma_transfer_t. |
| <i>srcAddr</i>       | DMA transfer source address.                             |
| <i>dstAddr</i>       | DMA transfer destination address.                        |
| <i>byteWidth</i>     | DMA transfer destination address width(bytes).           |
| <i>transferBytes</i> | DMA transfer bytes to be transferred.                    |
| <i>type</i>          | DMA transfer type.                                       |
| <i>nextDesc</i>      | Chain custom descriptor to transfer.                     |

### Note

The data address and the data width must be consistent. For example, if the SRC is 4 bytes, so the source address must be 4 bytes aligned, or it shall result in source address error(SAE).

**11.7.28 void DMA\_PrepareChannelTransfer ( dma\_channel\_config\_t \* *config*, void \* *srcStartAddr*, void \* *dstStartAddr*, uint32\_t *xferCfg*, dma\_transfer\_type\_t *type*, dma\_channel\_trigger\_t \* *trigger*, void \* *nextDesc* )**

This function used to prepare channel transfer configurations.

### Parameters

---

|                     |                                                                                            |
|---------------------|--------------------------------------------------------------------------------------------|
| <i>config</i>       | Pointer to DMA channel transfer configuration structure.                                   |
| <i>srcStartAddr</i> | source start address.                                                                      |
| <i>dstStartAddr</i> | destination start address.                                                                 |
| <i>xferCfg</i>      | xfer configuration, user can reference DMA_CHANNEL_XFER about to how to get xferCfg value. |
| <i>type</i>         | transfer type.                                                                             |
| <i>trigger</i>      | DMA channel trigger configurations.                                                        |
| <i>nextDesc</i>     | address of next descriptor.                                                                |

### 11.7.29 status\_t DMA\_SubmitTransfer ( dma\_handle\_t \* *handle*, dma\_transfer\_config\_t \* *config* )

This function submits the DMA transfer request according to the transfer configuration structure. If the user submits the transfer request repeatedly, this function packs an unprocessed request as a TCD and enables scatter/gather feature to process it in the next time.

Parameters

|               |                                                  |
|---------------|--------------------------------------------------|
| <i>handle</i> | DMA handle pointer.                              |
| <i>config</i> | Pointer to DMA transfer configuration structure. |

Return values

|                              |                                                                     |
|------------------------------|---------------------------------------------------------------------|
| <i>kStatus_DMA_Success</i>   | It means submit transfer request succeed.                           |
| <i>kStatus_DMA_QueueFull</i> | It means TCD queue is full. Submit transfer request is not allowed. |
| <i>kStatus_DMA_Busy</i>      | It means the given channel is busy, need to submit request later.   |

### 11.7.30 void DMA\_SubmitChannelTransferParameter ( dma\_handle\_t \* *handle*, uint32\_t *xfercfg*, void \* *srcStartAddr*, void \* *dstStartAddr*, void \* *nextDesc* )

This function used to configure channel head descriptor that is used to start DMA transfer, the head descriptor table is defined in DMA driver, it is useful for the case:

1. for the single transfer, application doesn't need to allocate descriptor table, the head descriptor can be used for it.

```
DMA_SetChannelConfig(base, channel, trigger, isPeriph);
DMA_CreateHandle(handle, base, channel)
DMA_SubmitChannelTransferParameter(handle,
 DMA_CHANNEL_XFER(reload, clrTrig, intA, intB, width, srcInc, dstInc,
```

## Function Documentation

```
bytes), srcStartAddr, dstStartAddr, NULL);
DMA_StartTransfer(handle)
*
```

- for the linked transfer, application should responsible for link descriptor, for example, if 4 transfer is required, then application should prepare three descriptor table with macro , the head descriptor in driver can be used for the first transfer descriptor.

```
//define link descriptor table in application with macro
DMA_ALLOCATE_LINK_DESCRIPTOR(nextDesc[3]);

DMA_SetupDescriptor(nextDesc0, DMA_CHANNEL_XFER(reload, clrTrig,
 intA, intB, width, srcInc, dstInc, bytes),
srcStartAddr, dstStartAddr, nextDesc1);
DMA_SetupDescriptor(nextDesc1, DMA_CHANNEL_XFER(reload, clrTrig,
 intA, intB, width, srcInc, dstInc, bytes),
srcStartAddr, dstStartAddr, nextDesc2);
DMA_SetupDescriptor(nextDesc2, DMA_CHANNEL_XFER(reload, clrTrig,
 intA, intB, width, srcInc, dstInc, bytes),
srcStartAddr, dstStartAddr, NULL);
DMA_SetChannelConfig(base, channel, trigger, isPeriph);
DMA_CreateHandle(handle, base, channel)
DMA_SubmitChannelTransferParameter(handle,
 DMA_CHANNEL_XFER(reload, clrTrig, intA, intB, width, srcInc, dstInc,
bytes), srcStartAddr, dstStartAddr, nextDesc0);
DMA_StartTransfer(handle);
*
```

### Parameters

|                     |                                                                                            |
|---------------------|--------------------------------------------------------------------------------------------|
| <i>handle</i>       | Pointer to DMA handle.                                                                     |
| <i>xferCfg</i>      | xfer configuration, user can reference DMA_CHANNEL_XFER about to how to get xferCfg value. |
| <i>srcStartAddr</i> | source start address.                                                                      |
| <i>dstStartAddr</i> | destination start address.                                                                 |
| <i>nextDesc</i>     | address of next descriptor.                                                                |

### 11.7.31 void DMA\_SubmitChannelDescriptor ( dma\_handle\_t \* *handle*, dma\_descriptor\_t \* *descriptor* )

This function used to configure channel head descriptor that is used to start DMA transfer, the head descriptor table is defined in DMA driver, this function is typical for the ping pong case:

- for the ping pong case, application should responsible for the descriptor, for example, application should prepare two descriptor table with macro.

```
//define link descriptor table in application with macro
DMA_ALLOCATE_LINK_DESCRIPTOR(nextDesc[2]);

DMA_SetupDescriptor(nextDesc0, DMA_CHANNEL_XFER(reload, clrTrig,
 intA, intB, width, srcInc, dstInc, bytes),
srcStartAddr, dstStartAddr, nextDesc1);
DMA_SetupDescriptor(nextDesc1, DMA_CHANNEL_XFER(reload, clrTrig,
 intA, intB, width, srcInc, dstInc, bytes),
```

```

srcStartAddr, dstStartAddr, nextDesc0);
DMA_SetChannelConfig(base, channel, trigger, isPeriph);
DMA_CreateHandle(handle, base, channel)
DMA_SubmitChannelDescriptor(handle, nextDesc0);
DMA_StartTransfer(handle);

```

\*

#### Parameters

|                   |                        |
|-------------------|------------------------|
| <i>handle</i>     | Pointer to DMA handle. |
| <i>descriptor</i> | descriptor to submit.  |

### 11.7.32 `status_t DMA_SubmitChannelTransfer ( dma_handle_t * handle, dma_channel_config_t * config )`

This function submits the DMA transfer request according to the transfer configuration structure. If the user submits the transfer request repeatedly, this function packs an unprocessed request as a TCD and enables scatter/gather feature to process it in the next time. It is used for the case:

1. for the single transfer, application doesn't need to allocate descriptor table, the head descriptor can be used for it.

```

DMA_CreateHandle(handle, base, channel)
DMA_PrepareChannelTransfer(config, srcStartAddr, dstStartAddr, xferCfg, type,
 trigger, NULL);
DMA_SubmitChannelTransfer(handle, config)
DMA_StartTransfer(handle)

```

\*

2. for the linked transfer, application should responsible for link descriptor, for example, if 4 transfer is required, then application should prepare three descriptor table with macro , the head descriptor in driver can be used for the first transfer descriptor.

```

//define link descriptor table in application with macro
DMA_ALLOCATE_LINK_DESCRIPTOR(nextDesc);
DMA_SetupDescriptor(nextDesc0, DMA_CHANNEL_XFER(reload, clrTrig,
 intA, intB, width, srcInc, dstInc, bytes),
srcStartAddr, dstStartAddr, nextDesc1);
DMA_SetupDescriptor(nextDesc1, DMA_CHANNEL_XFER(reload, clrTrig,
 intA, intB, width, srcInc, dstInc, bytes),
srcStartAddr, dstStartAddr, nextDesc2);
DMA_SetupDescriptor(nextDesc2, DMA_CHANNEL_XFER(reload, clrTrig,
 intA, intB, width, srcInc, dstInc, bytes),
srcStartAddr, dstStartAddr, NULL);
DMA_CreateHandle(handle, base, channel)
DMA_PrepareChannelTransfer(config, srcStartAddr, dstStartAddr, xferCfg, type,
 trigger, nextDesc0);
DMA_SubmitChannelTransfer(handle, config)
DMA_StartTransfer(handle)

```

\*

3. for the ping pong case, application should responsible for link descriptor, for example, application should prepare two descriptor table with macro , the head descriptor in driver can be used for the first transfer descriptor.

## Function Documentation

```
//define link descriptor table in application with macro
DMA_ALLOCATE_LINK_DESCRIPTOR(nextDesc);

DMA_SetupDescriptor(nextDesc0, DMA_CHANNEL_XFER(reload, clrTrig,
 intA, intB, width, srcInc, dstInc, bytes),
srcStartAddr, dstStartAddr, nextDesc1);
DMA_SetupDescriptor(nextDesc1, DMA_CHANNEL_XFER(reload, clrTrig,
 intA, intB, width, srcInc, dstInc, bytes),
srcStartAddr, dstStartAddr, nextDesc0);
DMA_CreateHandle(handle, base, channel)
DMA_PrepareChannelTransfer(config,srcStartAddr,dstStartAddr,xferCfg,type,
 trigger,nextDesc0);
DMA_SubmitChannelTransfer(handle, config)
DMA_StartTransfer(handle)
*
```

### Parameters

|               |                                                  |
|---------------|--------------------------------------------------|
| <i>handle</i> | DMA handle pointer.                              |
| <i>config</i> | Pointer to DMA transfer configuration structure. |

### Return values

|                              |                                                                     |
|------------------------------|---------------------------------------------------------------------|
| <i>kStatus_DMA_Success</i>   | It means submit transfer request succeed.                           |
| <i>kStatus_DMA_QueueFull</i> | It means TCD queue is full. Submit transfer request is not allowed. |
| <i>kStatus_DMA_Busy</i>      | It means the given channel is busy, need to submit request later.   |

### 11.7.33 void DMA\_StartTransfer ( dma\_handle\_t \* *handle* )

This function enables the channel request. User can call this function after submitting the transfer request. It will trigger transfer start with software trigger only when hardware trigger is not used.

#### Parameters

|               |                     |
|---------------|---------------------|
| <i>handle</i> | DMA handle pointer. |
|---------------|---------------------|

### 11.7.34 void DMA\_IRQHandle ( DMA\_Type \* *base* )

This function clears the channel major interrupt flag and call the callback function if it is not NULL.

#### Parameters

|             |                   |
|-------------|-------------------|
| <i>base</i> | DMA base address. |
|-------------|-------------------|



## Chapter 12

### DMIC: Digital Microphone

#### 12.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Digital Microphone (DMIC) module.

The DMIC driver is created to help the user more easily operate the DMIC module. This driver can be used to performed basic and advanced DMIC operations. The driver can be used to transfer data from DMIC to memory using DMA as well as in interrupt mode. The DMIC and DMA transfer in pingpong mode is preferred as DMIC is a streaming device.

#### 12.2 Function groups

##### 12.2.1 Initialization and deinitialization

This function group implements DMIC initialization and deinitialization API. [DMIC\\_Init\(\)](#) function Enables the clock to the DMIC register interface. [DMIC\\_Dinit\(\)](#) function Disables the clock to the DMIC register interface.

##### 12.2.2 Configuration

This function group implements DMIC configuration API. [DMIC\\_ConfigIO\(\)](#)function configures the use of PDM (Pulse Density moulation) pins. [DMIC\\_SetOperationMode\(\)](#)function configures the mode of operation either in DMA or in interrupt. [DMIC\\_ConfigChannel\(\)](#) function configures the various property of a DMIC channel. [DMIC\\_Use2fs\(\)](#)function configures the clock scaling used for PCM data output. [DMIC\\_EnableChannnel\(\)](#) function enables a particualr DMIC channel. [DMIC\\_FifoChannel\(\)](#) function configures FIFO settings for a DMIC channel.

##### 12.2.3 DMIC Data and status

This function group implements the API to get data and status of DMIC FIFO. [DMIC\\_FifoGetStatus\(\)](#) function gives the status of a DMIC FIFO. [DMIC\\_ClearStatus\(\)](#) function clears the status of a DMIC FIFO. [DMIC\\_FifoGetData\(\)](#) function gets data from a DMIC FIFO.

##### 12.2.4 DMIC Interrupt Functions

[DMIC\\_EnablebleIntCallback\(\)](#) enables the interrupt for the selected DMIC peripheral. [DMIC\\_Disable-IntCallback\(\)](#) disables the interrupt for the selected DMIC peripheral.

## Typical use case

### 12.2.5 DMIC HWVAD Functions

This function group implements the API for HWVAD. [DMIC\\_SetGainNoiseEstHwvad\(\)](#) Sets the gain value for the noise estimator. [DMIC\\_SetGainSignalEstHwvad\(\)](#) Sets the gain value for the signal estimator. [DMIC\\_SetFilterCtrlHwvad\(\)](#) Sets the HWVAD filter cutoff frequency parameter. [DMIC\\_SetInputGainHwvad\(\)](#) Sets the input gain of HWVAD. [DMIC\\_CtrlClrIntrHwvad\(\)](#) Clears HWVAD internal interrupt flag. [DMIC\\_FilterResetHwvad\(\)](#) Resets HWVAD filters. [DMIC\\_GetNoiseEnvlpEst\(\)](#) Gets the value from output of the filter z7.

### 12.2.6 DMIC HWVAD Interrupt Functions

[DMIC\\_HwvadEnableIntCallback\(\)](#) enables the HWVAD interrupt for the selected DMIC peripheral. [DMIC\\_HwvadDisableIntCallback\(\)](#) disables the HWVAD interrupt for the selected DMIC peripheral.

## 12.3 Typical use case

### 12.3.1 DMIC DMA Configuration

Refer to the driver examples codes located at <SDK\_ROOT>/boards/<BOARD>/driver\_examples/dmic

### 12.3.2 DMIC use case

Refer to the driver examples codes located at <SDK\_ROOT>/boards/<BOARD>/driver\_examples/dmic

## Modules

- [DMIC DMA Driver](#)
- [DMIC Driver](#)

## 12.4 DMIC Driver

### 12.4.1 Overview

#### Files

- file [fsl\\_dmic.h](#)

#### Data Structures

- struct [dmic\\_channel\\_config\\_t](#)  
*DMIC Channel configuration structure. [More...](#)*

#### Typedefs

- typedef void(\* [dmic\\_callback\\_t](#))(void)  
*DMIC Callback function.*
- typedef void(\* [dmic\\_hwvad\\_callback\\_t](#))(void)  
*HWVAD Callback function.*

#### Enumerations

- enum [\\_dmic\\_status](#) {  
    [kStatus\\_DMIC\\_Busy](#) = MAKE\_STATUS(kStatusGroup\_DMIC, 0),  
    [kStatus\\_DMIC\\_Idle](#) = MAKE\_STATUS(kStatusGroup\_DMIC, 1),  
    [kStatus\\_DMIC\\_OverRunError](#) = MAKE\_STATUS(kStatusGroup\_DMIC, 2),  
    [kStatus\\_DMIC\\_UnderRunError](#) = MAKE\_STATUS(kStatusGroup\_DMIC, 3) }  
*DMIC transfer status.*
- enum [operation\\_mode\\_t](#) {  
    [kDMIC\\_OperationModeInterrupt](#) = 1U,  
    [kDMIC\\_OperationModeDma](#) = 2U }  
*DMIC different operation modes.*
- enum [stereo\\_side\\_t](#) {  
    [kDMIC\\_Left](#) = 0U,  
    [kDMIC\\_Right](#) = 1U }  
*DMIC left/right values.*
- enum [pdm\\_div\\_t](#) {

## DMIC Driver

```
kDMIC_PdmDiv1 = 0U,
kDMIC_PdmDiv2 = 1U,
kDMIC_PdmDiv3 = 2U,
kDMIC_PdmDiv4 = 3U,
kDMIC_PdmDiv6 = 4U,
kDMIC_PdmDiv8 = 5U,
kDMIC_PdmDiv12 = 6U,
kDMIC_PdmDiv16 = 7U,
kDMIC_PdmDiv24 = 8U,
kDMIC_PdmDiv32 = 9U,
kDMIC_PdmDiv48 = 10U,
kDMIC_PdmDiv64 = 11U,
kDMIC_PdmDiv96 = 12U,
kDMIC_PdmDiv128 = 13U }
```

*DMIC Clock pre-divider values.*

- enum `compensation_t` {  
    kDMIC\_CompValueZero = 0U,  
    kDMIC\_CompValueNegativePoint16 = 1U,  
    kDMIC\_CompValueNegativePoint15 = 2U,  
    kDMIC\_CompValueNegativePoint13 = 3U }

*Pre-emphasis Filter coefficient value for 2FS and 4FS modes.*

- enum `dc_removal_t` {  
    kDMIC\_DcNoRemove = 0U,  
    kDMIC\_DcCut155 = 1U,  
    kDMIC\_DcCut78 = 2U,  
    kDMIC\_DcCut39 = 3U }

*DMIC DC filter control values.*

- enum `dmic_io_t` {  
    kDMIC\_PdmDual = 0,  
    kDMIC\_PdmStereo = 4 }

*DMIC IO configuration.*

- enum `dmic_channel_t` {  
    kDMIC\_Channel0 = 0U,  
    kDMIC\_Channel1 = 1U }

*DMIC Channel number.*

- enum `_dmic_channel_mask` {  
    kDMIC\_EnableChannel0 = 1 << 0U,  
    kDMIC\_EnableChannel1 = 1 << 1U }

*DMIC Channel mask.*

- enum `dmic_phy_sample_rate_t` {  
    kDMIC\_PhyFullSpeed = 0U,  
    kDMIC\_PhyHalfSpeed = 1U }

*DMIC and decimator sample rates.*

## DMIC version

- #define [FSL\\_DMIC\\_DRIVER\\_VERSION](#) ([MAKE\\_VERSION](#)(2, 1, 1))  
*DMIC driver version 2.1.1.*

## Initialization and deinitialization

- uint32\_t [DMIC\\_GetInstance](#) (DMIC\_Type \*base)  
*Get the DMIC instance from peripheral base address.*
- void [DMIC\\_Init](#) (DMIC\_Type \*base)  
*Turns DMIC Clock on.*
- void [DMIC\\_DeInit](#) (DMIC\_Type \*base)  
*Turns DMIC Clock off.*
- void [DMIC\\_ConfigIO](#) (DMIC\_Type \*base, [dmic\\_io\\_t](#) config)  
*Configure DMIC io.*
- static void [DMIC\\_SetIOCFG](#) (DMIC\_Type \*base, uint32\_t sel)  
*Stereo PDM select.*
- void [DMIC\\_SetOperationMode](#) (DMIC\_Type \*base, [operation\\_mode\\_t](#) mode)  
*Set DMIC operating mode.*
- void [DMIC\\_Use2fs](#) (DMIC\_Type \*base, bool use2fs)  
*Configure Clock scaling.*

## Channel configuration

- void [DMIC\\_CfgChannelDc](#) (DMIC\_Type \*base, [dmic\\_channel\\_t](#) channel, [dc\\_removal\\_t](#) dc\_cut\_level, uint32\_t post\_dc\_gain\_reduce, bool saturate16bit)  
*Configure DMIC channel.*
- void [DMIC\\_ConfigChannel](#) (DMIC\_Type \*base, [dmic\\_channel\\_t](#) channel, [stereo\\_side\\_t](#) side, [dmic\\_channel\\_config\\_t](#) \*channel\_config)  
*Configure DMIC channel.*
- void [DMIC\\_EnableChannel](#) (DMIC\_Type \*base, uint32\_t channelmask)  
*Enable a particular channel.*
- void [DMIC\\_FifoChannel](#) (DMIC\_Type \*base, uint32\_t channel, uint32\_t trig\_level, uint32\_t enable, uint32\_t resetn)  
*Configure fifo settings for DMIC channel.*
- static void [DMIC\\_EnableChannelInterrupt](#) (DMIC\_Type \*base, [dmic\\_channel\\_t](#) channel, bool enable)  
*Enable a particular channel interrupt request.*
- static void [DMIC\\_EnableChannelDma](#) (DMIC\_Type \*base, [dmic\\_channel\\_t](#) channel, bool enable)  
*Enable a particular channel dma request.*
- static void [DMIC\\_EnableChannelFifo](#) (DMIC\_Type \*base, [dmic\\_channel\\_t](#) channel, bool enable)  
*Enable a particular channel fifo.*
- static void [DMIC\\_DoFifoReset](#) (DMIC\_Type \*base, [dmic\\_channel\\_t](#) channel)  
*Channel fifo reset.*
- static uint32\_t [DMIC\\_FifoGetStatus](#) (DMIC\_Type \*base, uint32\_t channel)  
*Get FIFO status.*
- static void [DMIC\\_FifoClearStatus](#) (DMIC\_Type \*base, uint32\_t channel, uint32\_t mask)  
*Clear FIFO status.*

## DMIC Driver

- static uint32\_t [DMIC\\_FifoGetData](#) (DMIC\_Type \*base, uint32\_t channel)  
*Get FIFO data.*
- static uint32\_t [DMIC\\_FifoGetAddress](#) (DMIC\_Type \*base, uint32\_t channel)  
*Get FIFO address.*

### Register callback.

- void [DMIC\\_EnableIntCallback](#) (DMIC\_Type \*base, [dmic\\_callback\\_t](#) cb)  
*Enable callback.*
- void [DMIC\\_DisableIntCallback](#) (DMIC\_Type \*base, [dmic\\_callback\\_t](#) cb)  
*Disable callback.*

## HWVAD

- static void [DMIC\\_SetGainNoiseEstHwvad](#) (DMIC\_Type \*base, uint32\_t value)  
*Sets the gain value for the noise estimator.*
- static void [DMIC\\_SetGainSignalEstHwvad](#) (DMIC\_Type \*base, uint32\_t value)  
*Sets the gain value for the signal estimator.*
- static void [DMIC\\_SetFilterCtrlHwvad](#) (DMIC\_Type \*base, uint32\_t value)  
*Sets the hwvad filter cutoff frequency parameter.*
- static void [DMIC\\_SetInputGainHwvad](#) (DMIC\_Type \*base, uint32\_t value)  
*Sets the input gain of hwvad.*
- static void [DMIC\\_CtrlClrIntrHwvad](#) (DMIC\_Type \*base, bool st10)  
*Clears hwvad internal interrupt flag.*
- static void [DMIC\\_FilterResetHwvad](#) (DMIC\_Type \*base, bool rstt)  
*Resets hwvad filters.*
- static uint16\_t [DMIC\\_GetNoiseEnvlpEst](#) (DMIC\_Type \*base)  
*Gets the value from output of the filter z7.*
- void [DMIC\\_HwvadEnableIntCallback](#) (DMIC\_Type \*base, [dmic\\_hwvad\\_callback\\_t](#) vadcb)  
*Enable hwvad callback.*
- void [DMIC\\_HwvadDisableIntCallback](#) (DMIC\_Type \*base, [dmic\\_hwvad\\_callback\\_t](#) vadcb)  
*Disable callback.*

## 12.4.2 Data Structure Documentation

### 12.4.2.1 struct dmic\_channel\_config\_t

#### Data Fields

- [pdm\\_div\\_t](#) [divhfclk](#)  
*DMIC Clock pre-divider values.*
- uint32\_t [osr](#)  
*oversampling rate(CIC decimation rate) for PCM*
- int32\_t [gainshft](#)  
*4FS PCM data gain control*
- [compensation\\_t](#) [preac2coef](#)

- *Pre-emphasis Filter coefficient value for 2FS.*  
`compensation_t preac4coef`
- *Pre-emphasis Filter coefficient value for 4FS.*  
`dc_removal_t dc_cut_level`
- *DMIC DC filter control values.*  
`uint32_t post_dc_gain_reduce`
- *Fine gain adjustment in the form of a number of bits to downshift.*  
`dmic_phy_sample_rate_t sample_rate`
- *DMIC and decimator sample rates.*  
`bool saturate16bit`
- *Selects 16-bit saturation.*

#### 12.4.2.1.0.7 Field Documentation

##### 12.4.2.1.0.7.1 `dc_removal_t dmic_channel_config_t::dc_cut_level`

##### 12.4.2.1.0.7.2 `bool dmic_channel_config_t::saturate16bit`

0 means results roll over if out range and do not saturate. 1 means if the result overflows, it saturates at 0xFFFF for positive overflow and 0x8000 for negative overflow.

### 12.4.3 Macro Definition Documentation

#### 12.4.3.1 `#define FSL_DMIC_DRIVER_VERSION (MAKE_VERSION(2, 1, 1))`

### 12.4.4 Typedef Documentation

#### 12.4.4.1 `typedef void(* dmic_callback_t)(void)`

#### 12.4.4.2 `typedef void(* dmic_hwvad_callback_t)(void)`

### 12.4.5 Enumeration Type Documentation

#### 12.4.5.1 `enum _dmic_status`

Enumerator

- `kStatus_DMIC_Busy`* DMIC is busy.
- `kStatus_DMIC_Idle`* DMIC is idle.
- `kStatus_DMIC_OverRunError`* DMIC over run Error.
- `kStatus_DMIC_UnderRunError`* DMIC under run Error.

## DMIC Driver

### 12.4.5.2 enum operation\_mode\_t

Enumerator

*kDMIC\_OperationModeInterrupt* Interrupt mode.

*kDMIC\_OperationModeDma* DMA mode.

### 12.4.5.3 enum stereo\_side\_t

Enumerator

*kDMIC\_Left* Left Stereo channel.

*kDMIC\_Right* Right Stereo channel.

### 12.4.5.4 enum pdm\_div\_t

Enumerator

*kDMIC\_PdmDiv1* DMIC pre-divider set in divide by 1.

*kDMIC\_PdmDiv2* DMIC pre-divider set in divide by 2.

*kDMIC\_PdmDiv3* DMIC pre-divider set in divide by 3.

*kDMIC\_PdmDiv4* DMIC pre-divider set in divide by 4.

*kDMIC\_PdmDiv6* DMIC pre-divider set in divide by 6.

*kDMIC\_PdmDiv8* DMIC pre-divider set in divide by 8.

*kDMIC\_PdmDiv12* DMIC pre-divider set in divide by 12.

*kDMIC\_PdmDiv16* DMIC pre-divider set in divide by 16.

*kDMIC\_PdmDiv24* DMIC pre-divider set in divide by 24.

*kDMIC\_PdmDiv32* DMIC pre-divider set in divide by 32.

*kDMIC\_PdmDiv48* DMIC pre-divider set in divide by 48.

*kDMIC\_PdmDiv64* DMIC pre-divider set in divide by 64.

*kDMIC\_PdmDiv96* DMIC pre-divider set in divide by 96.

*kDMIC\_PdmDiv128* DMIC pre-divider set in divide by 128.

### 12.4.5.5 enum compensation\_t

Enumerator

*kDMIC\_CompValueZero* Compensation 0.

*kDMIC\_CompValueNegativePoint16* Compensation -0.16.

*kDMIC\_CompValueNegativePoint15* Compensation -0.15.

*kDMIC\_CompValueNegativePoint13* Compensation -0.13.



#### 12.4.5.6 enum dc\_removal\_t

Enumerator

*kDMIC\_DcNoRemove* Flat response no filter.  
*kDMIC\_DcCut155* Cut off Frequency is 155 Hz.  
*kDMIC\_DcCut78* Cut off Frequency is 78 Hz.  
*kDMIC\_DcCut39* Cut off Frequency is 39 Hz.

#### 12.4.5.7 enum dmic\_io\_t

Enumerator

*kDMIC\_PdmDual* Two separate pairs of PDM wires.  
*kDMIC\_PdmStereo* Stereo data0.

#### 12.4.5.8 enum dmic\_channel\_t

Enumerator

*kDMIC\_Channel0* DMIC channel 0.  
*kDMIC\_Channel1* DMIC channel 1.

#### 12.4.5.9 enum \_dmic\_channel\_mask

Enumerator

*kDMIC\_EnableChannel0* DMIC channel 0 mask.  
*kDMIC\_EnableChannel1* DMIC channel 1 mask.

#### 12.4.5.10 enum dmic\_phy\_sample\_rate\_t

Enumerator

*kDMIC\_PhyFullSpeed* Decimator gets one sample per each chosen clock edge of PDM interface.  
*kDMIC\_PhyHalfSpeed* PDM clock to Microphone is halved, decimator receives each sample twice.

### 12.4.6 Function Documentation

#### 12.4.6.1 uint32\_t DMIC\_GetInstance ( DMIC\_Type \* base )

## DMIC Driver

### Parameters

|             |                               |
|-------------|-------------------------------|
| <i>base</i> | DMIC peripheral base address. |
|-------------|-------------------------------|

### Returns

DMIC instance.

#### 12.4.6.2 void DMIC\_Init ( DMIC\_Type \* *base* )

### Parameters

|             |             |
|-------------|-------------|
| <i>base</i> | : DMIC base |
|-------------|-------------|

### Returns

Nothing

#### 12.4.6.3 void DMIC\_DeInit ( DMIC\_Type \* *base* )

### Parameters

|             |             |
|-------------|-------------|
| <i>base</i> | : DMIC base |
|-------------|-------------|

### Returns

Nothing

#### 12.4.6.4 void DMIC\_ConfigIO ( DMIC\_Type \* *base*, dmic\_io\_t *config* )

### Parameters

|               |                                      |
|---------------|--------------------------------------|
| <i>base</i>   | : The base address of DMIC interface |
| <i>config</i> | : DMIC io configuration              |

### Returns

Nothing

#### 12.4.6.5 static void DMIC\_SetIOCFG ( DMIC\_Type \* *base*, uint32\_t *sel* ) [inline], [static]

## Parameters

|             |                                                                                                       |
|-------------|-------------------------------------------------------------------------------------------------------|
| <i>base</i> | : The base address of DMIC interface                                                                  |
| <i>sel</i>  | : Reference <code>dmic_io_t</code> , can be a single or combination value of <code>dmic_io_t</code> . |

## Returns

Nothing

**12.4.6.6 void DMIC\_SetOperationMode ( DMIC\_Type \* *base*, operation\_mode\_t *mode* )**

## Parameters

|             |                                      |
|-------------|--------------------------------------|
| <i>base</i> | : The base address of DMIC interface |
| <i>mode</i> | : DMIC mode                          |

## Returns

Nothing

**12.4.6.7 void DMIC\_Use2fs ( DMIC\_Type \* *base*, bool *use2fs* )**

## Parameters

|               |                                      |
|---------------|--------------------------------------|
| <i>base</i>   | : The base address of DMIC interface |
| <i>use2fs</i> | : clock scaling                      |

## Returns

Nothing

**12.4.6.8 void DMIC\_CfgChannelDc ( DMIC\_Type \* *base*, `dmic_channel_t` *channel*, `dc_removal_t` *dc\_cut\_level*, `uint32_t` *post\_dc\_gain\_reduce*, bool *saturate16bit* )**

## DMIC Driver

### Parameters

|                            |                                                                      |
|----------------------------|----------------------------------------------------------------------|
| <i>base</i>                | : The base address of DMIC interface                                 |
| <i>channel</i>             | : DMIC channel                                                       |
| <i>dc_cut_level</i>        | : dc_removal_t, Cut off Frequency                                    |
| <i>post_dc_gain_reduce</i> | : Fine gain adjustment in the form of a number of bits to downshift. |
| <i>saturate16bit</i>       | : If selects 16-bit saturation.                                      |

**12.4.6.9 void DMIC\_ConfigChannel ( DMIC\_Type \* *base*, dmic\_channel\_t *channel*, stereo\_side\_t *side*, dmic\_channel\_config\_t \* *channel\_config* )**

### Parameters

|                       |                                          |
|-----------------------|------------------------------------------|
| <i>base</i>           | : The base address of DMIC interface     |
| <i>channel</i>        | : DMIC channel                           |
| <i>side</i>           | : stereo_side_t, choice of left or right |
| <i>channel_config</i> | : Channel configuration                  |

### Returns

Nothing

**12.4.6.10 void DMIC\_EnableChannel ( DMIC\_Type \* *base*, uint32\_t *channelmask* )**

### Parameters

|                               |                                      |
|-------------------------------|--------------------------------------|
| <i>base</i>                   | : The base address of DMIC interface |
| <i>channel-mask,reference</i> | _dmic_channel_mask                   |

### Returns

Nothing

**12.4.6.11 void DMIC\_FifoChannel ( DMIC\_Type \* *base*, uint32\_t *channel*, uint32\_t *trig\_level*, uint32\_t *enable*, uint32\_t *resetn* )**

## Parameters

|                   |                                      |
|-------------------|--------------------------------------|
| <i>base</i>       | : The base address of DMIC interface |
| <i>channel</i>    | : DMIC channel                       |
| <i>trig_level</i> | : FIFO trigger level                 |
| <i>enable</i>     | : FIFO level                         |
| <i>resetrn</i>    | : FIFO reset                         |

## Returns

Nothing

**12.4.6.12** `static void DMIC_EnableChannelInterrupt ( DMIC_Type * base, dmic_channel_t channel, bool enable ) [inline], [static]`

## Parameters

|                |                                      |
|----------------|--------------------------------------|
| <i>base</i>    | : The base address of DMIC interface |
| <i>channel</i> | : Channel selection                  |

**12.4.6.13** `static void DMIC_EnableChannelDma ( DMIC_Type * base, dmic_channel_t channel, bool enable ) [inline], [static]`

## Parameters

|                |                                      |
|----------------|--------------------------------------|
| <i>base</i>    | : The base address of DMIC interface |
| <i>channel</i> | : Channel selection                  |
| <i>enable</i>  | : true is enable, false is disable   |

**12.4.6.14** `static void DMIC_EnableChannelFifo ( DMIC_Type * base, dmic_channel_t channel, bool enable ) [inline], [static]`

## Parameters

---

## DMIC Driver

|                |                                      |
|----------------|--------------------------------------|
| <i>base</i>    | : The base address of DMIC interface |
| <i>channel</i> | : Channel selection                  |
| <i>enable</i>  | : true is enable, false is disable   |

**12.4.6.15** `static void DMIC_DoFifoReset ( DMIC_Type * base, dmic_channel_t channel )  
[inline], [static]`

Parameters

|                |                                      |
|----------------|--------------------------------------|
| <i>base</i>    | : The base address of DMIC interface |
| <i>channel</i> | : Channel selection                  |

**12.4.6.16** `static uint32_t DMIC_FifoGetStatus ( DMIC_Type * base, uint32_t channel )  
[inline], [static]`

Parameters

|                |                                      |
|----------------|--------------------------------------|
| <i>base</i>    | : The base address of DMIC interface |
| <i>channel</i> | : DMIC channel                       |

Returns

FIFO status

**12.4.6.17** `static void DMIC_FifoClearStatus ( DMIC_Type * base, uint32_t channel,  
uint32_t mask ) [inline], [static]`

Parameters

|                |                                      |
|----------------|--------------------------------------|
| <i>base</i>    | : The base address of DMIC interface |
| <i>channel</i> | : DMIC channel                       |
| <i>mask</i>    | : Bits to be cleared                 |

Returns

FIFO status

**12.4.6.18** `static uint32_t DMIC_FifoGetData ( DMIC_Type * base, uint32_t channel )`  
`[inline], [static]`

## DMIC Driver

### Parameters

|                |                                      |
|----------------|--------------------------------------|
| <i>base</i>    | : The base address of DMIC interface |
| <i>channel</i> | : DMIC channel                       |

### Returns

FIFO data

**12.4.6.19 static uint32\_t DMIC\_FifoGetAddress ( DMIC\_Type \* *base*, uint32\_t *channel* )**  
**[inline], [static]**

### Parameters

|                |                                      |
|----------------|--------------------------------------|
| <i>base</i>    | : The base address of DMIC interface |
| <i>channel</i> | : DMIC channel                       |

### Returns

FIFO data

**12.4.6.20 void DMIC\_EnableIntCallback ( DMIC\_Type \* *base*, dmic\_callback\_t *cb* )**

This function enables the interrupt for the selected DMIC peripheral. The callback function is not enabled until this function is called.

### Parameters

|             |                                              |
|-------------|----------------------------------------------|
| <i>base</i> | Base address of the DMIC peripheral.         |
| <i>cb</i>   | callback Pointer to store callback function. |

### Return values

|              |  |
|--------------|--|
| <i>None.</i> |  |
|--------------|--|

**12.4.6.21 void DMIC\_DisableIntCallback ( DMIC\_Type \* *base*, dmic\_callback\_t *cb* )**

This function disables the interrupt for the selected DMIC peripheral.



## Parameters

|             |                                               |
|-------------|-----------------------------------------------|
| <i>base</i> | Base address of the DMIC peripheral.          |
| <i>cb</i>   | callback Pointer to store callback function.. |

## Return values

|              |  |
|--------------|--|
| <i>None.</i> |  |
|--------------|--|

**12.4.6.22 static void DMIC\_SetGainNoiseEstHwvad ( DMIC\_Type \* *base*, uint32\_t *value* )**  
**[inline], [static]**

## Parameters

|              |                                     |
|--------------|-------------------------------------|
| <i>base</i>  | DMIC base pointer                   |
| <i>value</i> | gain value for the noise estimator. |

## Return values

|              |  |
|--------------|--|
| <i>None.</i> |  |
|--------------|--|

**12.4.6.23 static void DMIC\_SetGainSignalEstHwvad ( DMIC\_Type \* *base*, uint32\_t *value* )**  
**[inline], [static]**

## Parameters

|              |                                      |
|--------------|--------------------------------------|
| <i>base</i>  | DMIC base pointer                    |
| <i>value</i> | gain value for the signal estimator. |

## Return values

|              |  |
|--------------|--|
| <i>None.</i> |  |
|--------------|--|

**12.4.6.24 static void DMIC\_SetFilterCtrlHwvad ( DMIC\_Type \* *base*, uint32\_t *value* )**  
**[inline], [static]**

## DMIC Driver

### Parameters

|              |                          |
|--------------|--------------------------|
| <i>base</i>  | DMIC base pointer        |
| <i>value</i> | cut off frequency value. |

### Return values

|              |  |
|--------------|--|
| <i>None.</i> |  |
|--------------|--|

**12.4.6.25 static void DMIC\_SetInputGainHwvad ( DMIC\_Type \* *base*, uint32\_t *value* )**  
**[inline], [static]**

### Parameters

|              |                             |
|--------------|-----------------------------|
| <i>base</i>  | DMIC base pointer           |
| <i>value</i> | input gain value for hwvad. |

### Return values

|              |  |
|--------------|--|
| <i>None.</i> |  |
|--------------|--|

**12.4.6.26 static void DMIC\_CtrlClrIntrHwvad ( DMIC\_Type \* *base*, bool *st10* )**  
**[inline], [static]**

### Parameters

|             |                   |
|-------------|-------------------|
| <i>base</i> | DMIC base pointer |
| <i>st10</i> | bit value.        |

### Return values

|              |  |
|--------------|--|
| <i>None.</i> |  |
|--------------|--|

**12.4.6.27 static void DMIC\_FilterResethwvad ( DMIC\_Type \* *base*, bool *rstf* )**  
**[inline], [static]**

## Parameters

|             |                   |
|-------------|-------------------|
| <i>base</i> | DMIC base pointer |
| <i>rstt</i> | Reset bit value.  |

## Return values

|              |  |
|--------------|--|
| <i>None.</i> |  |
|--------------|--|

#### 12.4.6.28 static uint16\_t DMIC\_GetNoiseEnvlpEst ( DMIC\_Type \* *base* ) [inline], [static]

## Parameters

|             |                   |
|-------------|-------------------|
| <i>base</i> | DMIC base pointer |
|-------------|-------------------|

## Return values

|               |               |
|---------------|---------------|
| <i>output</i> | of filter z7. |
|---------------|---------------|

#### 12.4.6.29 void DMIC\_HwvadEnableIntCallback ( DMIC\_Type \* *base*, dmic\_hwvad\_callback\_t *vadcb* )

This function enables the hwvad interrupt for the selected DMIC peripheral. The callback function is not enabled until this function is called.

## Parameters

|              |                                              |
|--------------|----------------------------------------------|
| <i>base</i>  | Base address of the DMIC peripheral.         |
| <i>vadcb</i> | callback Pointer to store callback function. |

## Return values

|              |  |
|--------------|--|
| <i>None.</i> |  |
|--------------|--|

#### 12.4.6.30 void DMIC\_HwvadDisableIntCallback ( DMIC\_Type \* *base*, dmic\_hwvad\_callback\_t *vadcb* )

This function disables the hwvad interrupt for the selected DMIC peripheral.

## DMIC Driver

### Parameters

|              |                                               |
|--------------|-----------------------------------------------|
| <i>base</i>  | Base address of the DMIC peripheral.          |
| <i>vadcb</i> | callback Pointer to store callback function.. |

### Return values

|              |  |
|--------------|--|
| <i>None.</i> |  |
|--------------|--|

## 12.5 DMIC DMA Driver

### 12.5.1 Overview

#### Files

- file [fsl\\_dmic\\_dma.h](#)

#### Data Structures

- struct [dmic\\_transfer\\_t](#)  
*DMIC transfer structure. [More...](#)*
- struct [dmic\\_dma\\_handle\\_t](#)  
*DMIC DMA handle. [More...](#)*

#### Typedefs

- typedef void(\* [dmic\\_dma\\_transfer\\_callback\\_t](#) )(DMIC\_Type \*base, dmic\_dma\_handle\_t \*handle, [status\\_t](#) status, void \*userData)  
*DMIC transfer callback function.*

#### DMIC DMA version

- #define [FSL\\_DMIC\\_DMA\\_DRIVER\\_VERSION](#) ([MAKE\\_VERSION](#)(2, 1, 1))  
*DMIC DMA driver version 2.1.1.*

#### DMA transactional

- [status\\_t DMIC\\_TransferCreateHandleDMA](#) (DMIC\_Type \*base, dmic\_dma\_handle\_t \*handle, [dmic\\_dma\\_transfer\\_callback\\_t](#) callback, void \*userData, [dma\\_handle\\_t](#) \*rxDmaHandle)  
*Initializes the DMIC handle which is used in transactional functions.*
- [status\\_t DMIC\\_TransferReceiveDMA](#) (DMIC\_Type \*base, dmic\_dma\_handle\_t \*handle, [dmic\\_transfer\\_t](#) \*xfer, uint32\_t dmic\_channel)  
*Receives data using DMA.*
- void [DMIC\\_TransferAbortReceiveDMA](#) (DMIC\_Type \*base, dmic\_dma\_handle\_t \*handle)  
*Aborts the received data using DMA.*
- [status\\_t DMIC\\_TransferGetReceiveCountDMA](#) (DMIC\_Type \*base, dmic\_dma\_handle\_t \*handle, uint32\_t \*count)  
*Get the number of bytes that have been received.*
- void [DMIC\\_InstallDMADescriptorMemory](#) (dmic\_dma\_handle\_t \*handle, void \*linkAddr, size\_t linkNum)  
*Install DMA descriptor memory.*

### 12.5.2 Data Structure Documentation

#### 12.5.2.1 struct dmic\_transfer\_t

##### Data Fields

- void \* [data](#)  
*The buffer of data to be transfer.*
- uint8\_t [dataWidth](#)  
*DMIC support 16bit/32bit.*
- size\_t [dataSize](#)  
*The byte count to be transfer.*
- uint8\_t [dataAddrInterleaveSize](#)  
*destination address interleave size*
- struct \_dmic\_transfer \* [linkTransfer](#)  
*use to support link transfer*

##### 12.5.2.1.0.8 Field Documentation

###### 12.5.2.1.0.8.1 void\* dmic\_transfer\_t::data

###### 12.5.2.1.0.8.2 size\_t dmic\_transfer\_t::dataSize

#### 12.5.2.2 struct \_dmic\_dma\_handle

##### Data Fields

- DMIC\_Type \* [base](#)  
*DMIC peripheral base address.*
- [dma\\_handle\\_t](#) \* [rxDmaHandle](#)  
*The DMA RX channel used.*
- [dmic\\_dma\\_transfer\\_callback\\_t](#) [callback](#)  
*Callback function.*
- void \* [userData](#)  
*DMIC callback function parameter.*
- size\_t [transferSize](#)  
*Size of the data to receive.*
- volatile uint8\_t [state](#)  
*Internal state of DMIC DMA transfer.*
- [dma\\_descriptor\\_t](#) \* [desLink](#)  
*descriptor pool pointer*
- size\_t [linkNum](#)  
*number of descriptor in descriptors pool*

### 12.5.2.2.0.9 Field Documentation

12.5.2.2.0.9.1 **DMIC\_Type\*** **dmic\_dma\_handle\_t::base**

12.5.2.2.0.9.2 **dma\_handle\_t\*** **dmic\_dma\_handle\_t::rxDmaHandle**

12.5.2.2.0.9.3 **dmic\_dma\_transfer\_callback\_t** **dmic\_dma\_handle\_t::callback**

12.5.2.2.0.9.4 **void\*** **dmic\_dma\_handle\_t::userData**

12.5.2.2.0.9.5 **size\_t** **dmic\_dma\_handle\_t::transferSize**

### 12.5.3 Macro Definition Documentation

12.5.3.1 **#define FSL\_DMIC\_DMA\_DRIVER\_VERSION (MAKE\_VERSION(2, 1, 1))**

### 12.5.4 Typedef Documentation

12.5.4.1 **typedef void**(**\* dmic\_dma\_transfer\_callback\_t**)(**DMIC\_Type \*base**,  
**dmic\_dma\_handle\_t \*handle**, **status\_t status**, **void \*userData**)

### 12.5.5 Function Documentation

12.5.5.1 **status\_t DMIC\_TransferCreateHandleDMA ( DMIC\_Type \* *base*,**  
**dmic\_dma\_handle\_t \* *handle*, dmic\_dma\_transfer\_callback\_t *callback*, void \***  
***userData*, dma\_handle\_t \* *rxDmaHandle* )**

## DMIC DMA Driver

### Parameters

|                    |                                                      |
|--------------------|------------------------------------------------------|
| <i>base</i>        | DMIC peripheral base address.                        |
| <i>handle</i>      | Pointer to <code>dmic_dma_handle_t</code> structure. |
| <i>callback</i>    | Callback function.                                   |
| <i>userData</i>    | User data.                                           |
| <i>rxDmaHandle</i> | User-requested DMA handle for RX DMA transfer.       |

### 12.5.5.2 `status_t DMIC_TransferReceiveDMA ( DMIC_Type * base, dmic_dma_handle_t * handle, dmic_transfer_t * xfer, uint32_t dmic_channel )`

This function receives data using DMA. This is a non-blocking function, which returns right away. When all data is received, the receive callback function is called.

### Parameters

|                     |                                                                    |
|---------------------|--------------------------------------------------------------------|
| <i>base</i>         | USART peripheral base address.                                     |
| <i>handle</i>       | Pointer to <code>usart_dma_handle_t</code> structure.              |
| <i>xfer</i>         | DMIC DMA transfer structure. See <a href="#">dmic_transfer_t</a> . |
| <i>dmic_channel</i> | DMIC start channel number.                                         |

### Return values

|                        |  |
|------------------------|--|
| <i>kStatus_Success</i> |  |
|------------------------|--|

### 12.5.5.3 `void DMIC_TransferAbortReceiveDMA ( DMIC_Type * base, dmic_dma_handle_t * handle )`

This function aborts the received data using DMA.

### Parameters

|               |                                                     |
|---------------|-----------------------------------------------------|
| <i>base</i>   | DMIC peripheral base address                        |
| <i>handle</i> | Pointer to <code>dmic_dma_handle_t</code> structure |

### 12.5.5.4 `status_t DMIC_TransferGetReceiveCountDMA ( DMIC_Type * base, dmic_dma_handle_t * handle, uint32_t * count )`

This function gets the number of bytes that have been received.



## Parameters

|               |                               |
|---------------|-------------------------------|
| <i>base</i>   | DMIC peripheral base address. |
| <i>handle</i> | DMIC handle pointer.          |
| <i>count</i>  | Receive bytes count.          |

## Return values

|                                     |                                               |
|-------------------------------------|-----------------------------------------------|
| <i>kStatus_NoTransferInProgress</i> | No receive in progress.                       |
| <i>kStatus_InvalidArgument</i>      | Parameter is invalid.                         |
| <i>kStatus_Success</i>              | Get successfully through the parameter count; |

#### 12.5.5.5 void DMIC\_InstallDMADescriptorMemory ( dmic\_dma\_handle\_t \* *handle*, void \* *linkAddr*, size\_t *linkNum* )

This function used to register DMA descriptor memory for linked transfer, a typical case is ping pong transfer which will request more than one DMA descriptor memory space, it should be called after DMIC\_TransferCreateHandleDMA. User should be take care about the address of DMA descriptor pool which required align with 16BYTE at least.

## Parameters

|                 |                                         |
|-----------------|-----------------------------------------|
| <i>handle</i>   | Pointer to DMA channel transfer handle. |
| <i>linkAddr</i> | DMA link descriptor address.            |
| <i>num</i>      | DMA link descriptor number.             |



## Chapter 13

### FLASH: Flash driver

#### 13.1 Overview

The MCUXpresso SDK provides a peripheral driver for the flash driver module of MCUXpresso SDK devices.

#### Files

- file [flash\\_header.h](#)
- file [fsl\\_flash.h](#)

#### Data Structures

- struct [IMG\\_HEADER\\_T](#)  
*Image header. [More...](#)*
- struct [BOOT\\_BLOCK\\_T](#)  
*Boot block. [More...](#)*
- struct [flash\\_config\\_t](#)  
*Flash configuration information. [More...](#)*

#### Macros

- #define [FLASH\\_FAIL](#) (1 << 0)  
*FLASH INT\_STATUS register definitions.*
- #define [FLASH\\_FAIL](#) (1 << 0)  
*FLASH INT\_STATUS register definitions.*
- #define [FLASH\\_ERR](#) (1 << 1)  
*Illegal command.*
- #define [FLASH\\_ERR](#) (1 << 1)  
*Illegal command.*
- #define [FLASH\\_DONE](#) (1 << 2)  
*Command complete.*
- #define [FLASH\\_DONE](#) (1 << 2)  
*Command complete.*
- #define [FLASH\\_ECC\\_ERR](#) (1 << 3)  
*ECC error detected.*
- #define [FLASH\\_ECC\\_ERR](#) (1 << 3)  
*ECC error detected.*

## Enumerations

- enum `flash_status_t` {  
`kStatus_FLASH_Success` = `FLASH_DONE`,  
`kStatus_FLASH_Fail` = `FLASH_DONE` | `FLASH_FAIL`,  
`kStatus_FLASH_InvalidArgument` = `MAKE_STATUS(kStatusGroup_Generic, 4)`,  
`kStatus_FLASH_AlignmentError` = `MAKE_STATUS(kStatusGroup_FLASH, 6)`,  
`kStatus_FLASH_EccError` = `FLASH_DONE` | `FLASH_ECC_ERR`,  
`kStatus_FLASH_Error` = `FLASH_DONE` | `FLASH_ERR` }
- enum `flash_read_mode_t` { , `FLASH_ReadModeNormalEccOff` = (`FLASH_READ_MODE_NORMAL` << `FLASH_READ_MODE_SHIFT`)|(1<<`FLASH_READ_MODE_ECC_OFF_SHIFT`) }

## Functions

- void `FLASH_Init` (`FLASH_Type` \*pFLASH)  
*Enable the FLASH.*
- void `FLASH_Powerdown` (`FLASH_Type` \*pFLASH)  
*Power down the FLASH.*
- int `FLASH_Wait` (`FLASH_Type` \*pFLASH)  
*Wait for FLASH command to complete.*
- int `FLASH_Erase` (`FLASH_Type` \*pFLASH, uint8\_t \*pu8Start, uint8\_t \*pu8End)  
*Erase page.*
- int `FLASH_ErasePages` (`FLASH_Type` \*pFLASH, uint32\_t u32StartPage, uint32\_t u32PageCount)  
*Erase multiple pages.*
- int `FLASH_BlankCheck` (`FLASH_Type` \*pFLASH, uint8\_t \*pu8Start, uint8\_t \*pu8End)  
*Page Blank check.*
- int `FLASH_MarginCheck` (`FLASH_Type` \*pFLASH, uint8\_t \*pu8Start, uint8\_t \*pu8End)  
*Margin Check.*
- int `FLASH_Program` (`FLASH_Type` \*pFLASH, uint32\_t \*pu32Start, uint32\_t \*pu32Data, uint32\_t u32Length)  
*Program page.*
- int `FLASH_Checksum` (`FLASH_Type` \*pFLASH, uint8\_t \*pu8Start, uint8\_t \*pu8End, uint32\_t au32Checksum[4])  
*Page Checksum.*
- int `FLASH_Read` (`FLASH_Type` \*pFLASH, uint8\_t \*pu8Start, uint32\_t u32ReadMode, uint32\_t au32Data[4])  
*Read flash word (16 byte worth of data)*
- void `FLASH_SetReadMode` (`FLASH_Type` \*pFLASH, bool freq\_48M\_not\_32M)  
*Configure the flash wait state depending of the elwe mode and CPU frequency. When the CPU clock frequency is decreased, the Set Read command shall be called after the frequency change. When the CPU clock frequency is increased, the Set Read command shall be called before the frequency change.*
- void `FLASH_CalculateChecksum` (const uint32\_t \*input, size\_t nb\_128b\_words, uint32\_t \*misr, int init)  
*Calculate checksum using the same checksum algorithm as the CMD\_CHECKSUM implementation of the Flash controller. When executed over a 512 byte page (page size) must return the same value as FLASH\_Checksum.*
- int `FLASH_ConfigPageVerifyPageChecksum` (const uint32\_t \*page\_buffer, uint32\_t \*misr)  
*Calculate checksum over page (N-2) aka CONFIG page and check it matches the expected value.*
- int `FLASH_ConfigPageVerifyGpoChecksum` (const uint32\_t \*page\_buffer, uint32\_t \*misr)  
*Calculate checksum over GPO array of CONFIG page and check it matches the expected value.*

- void [FLASH\\_ConfigPageUpdate](#) (uint32\_t \*page\_ram\_buffer, uint32\_t \*gpo\_chksum, uint32\_t \*page\_chksum)  
*Configure the flash wait state depending of the elwe mode and CPU frequency. When the CPU clock frequency is decreased, the Set Read command shall be called after the frequency change. When the CPU clock frequency is increased, the Set Read command shall be called before the frequency change.*
- int [FLASH\\_GetStatus](#) (FLASH\_Type \*pFLASH)  
*Return unfiltered FLASH\_INT\_STATUS. In normal operation FLASH\_DONE rises systematically but other status bits may rise at the same time or have risen before to notify of an error. Usually testing the value returned by FLASH\_Wait is sufficient but in some special cases the raw value may be needed.*

## Variables

- uint32\_t [IMG\\_HEADER\\_T::vectors](#) [NUMBER\_CCSUM\_VECTORS]  
*critical vectors protected by csum*
- uint32\_t [IMG\\_HEADER\\_T::vectorCsum](#)  
*csum of vectors 0-7*
- uint32\_t [IMG\\_HEADER\\_T::imageSignature](#)  
*image signature*
- uint32\_t [IMG\\_HEADER\\_T::bootBlockOffset](#)  
*offset of boot block structure*
- uint32\_t [IMG\\_HEADER\\_T::header\\_crc](#)  
*the CRC of header*
- uint32\_t [BOOT\\_BLOCK\\_T::header\\_marker](#)  
*Image header marker should always be set to 0xbb0110bb+/-2.*
- uint32\_t [BOOT\\_BLOCK\\_T::img\\_type](#)  
*Image check type, with or without optional CRC.*
- uint32\_t [BOOT\\_BLOCK\\_T::target\\_addr](#)  
*Target address.*
- uint32\_t [BOOT\\_BLOCK\\_T::img\\_len](#)  
*Image length or the length of image CRC check should be done.*
- uint32\_t [BOOT\\_BLOCK\\_T::stated\\_size](#)  
*max size of any subsequent image : AppSize0 = 2 x stated\_size*
- uint32\_t [BOOT\\_BLOCK\\_T::certificate\\_offset](#)  
*Offset of the certificate list.*
- uint32\_t [BOOT\\_BLOCK\\_T::compatibility\\_offset](#)  
*Offset of the compatibility list.*
- uint32\_t [BOOT\\_BLOCK\\_T::version](#)  
*Image version for multi-image support.*

## 13.2 Data Structure Documentation

### 13.2.1 struct IMG\_HEADER\_T

Be very cautious when modifying the [IMG\\_HEADER\\_T](#) and the [BOOT\\_BLOCK\\_T](#) structures (alignment) as these structures are used in the image\_tool.py (which does not take care of alignment).

## Data Fields

- uint32\_t [vectors](#) [NUMBER\_CCSUM\_VECTORS]

## Data Structure Documentation

- `uint32_t` [vectorCsum](#)  
*critical vectors protected by csum*
- `uint32_t` [imageSignature](#)  
*csum of vectors 0-7*
- `uint32_t` [bootBlockOffset](#)  
*image signature*
- `uint32_t` [header\\_crc](#)  
*offset of boot block structure*
- `uint32_t` [header\\_crc](#)  
*the CRC of header*

### 13.2.2 struct BOOT\_BLOCK\_T

For some ADC16 channels, there are two pin selections in channel multiplexer. For example, ADC0\_SE4a and ADC0\_SE4b are the different channels that share the same channel number.

#### Data Fields

- `uint32_t` [header\\_marker](#)  
*Image header marker should always be set to 0xbb0110bb+/-2.*
- `uint32_t` [img\\_type](#)  
*Image check type, with or without optional CRC.*
- `uint32_t` [target\\_addr](#)  
*Target address.*
- `uint32_t` [img\\_len](#)  
*Image length or the length of image CRC check should be done.*
- `uint32_t` [stated\\_size](#)  
*max size of any subsequent image : AppSize0 = 2 x stated\_size*
- `uint32_t` [certificate\\_offset](#)  
*Offset of the certificate list.*
- `uint32_t` [compatibility\\_offset](#)  
*Offset of the compatibility list.*
- `uint32_t` [version](#)  
*Image version for multi-image support.*

### 13.2.3 struct flash\_config\_t

An instance of this structure is allocated by the user of the flash driver and at initialization.

#### Data Fields

- `uint32_t` [PFlashBlockBase](#)  
*A base address of the first PFlash block.*
- `uint32_t` [PFlashTotalSize](#)  
*The size of the combined PFlash block.*
- `uint32_t` [PFlashSectorSize](#)

*The size in bytes of a sector of PFlash.*

#### 13.2.3.0.0.10 Field Documentation

13.2.3.0.0.10.1 uint32\_t flash\_config\_t::PFlashTotalSize

13.2.3.0.0.10.2 uint32\_t flash\_config\_t::PFlashSectorSize

### 13.3 Macro Definition Documentation

#### 13.3.1 #define FLASH\_FAIL (1 << 0)

FLASH\_INT\_ENABLE register definitions.

Command failed

#### 13.3.2 #define FLASH\_FAIL (1 << 0)

FLASH\_INT\_ENABLE register definitions.

Command failed

### 13.4 Enumeration Type Documentation

#### 13.4.1 enum flash\_status\_t

Enumerator

*kStatus\_FLASH\_Success* flash operation is successful

*kStatus\_FLASH\_Fail* flash operation is not successful

*kStatus\_FLASH\_InvalidArgument* Invalid argument.

*kStatus\_FLASH\_AlignmentError* Alignment Error.

*kStatus\_FLASH\_EccError* ECC error detected.

*kStatus\_FLASH\_Error* Illegal command.

#### 13.4.2 enum flash\_read\_mode\_t

Enumerator

*FLASH\_ReadModeNormalEccOff* flash operation is not successful

### 13.5 Function Documentation

#### 13.5.1 void FLASH\_Init ( FLASH\_Type \* pFLASH )

## Function Documentation

### Parameters

|               |                                       |
|---------------|---------------------------------------|
| <i>pFLASH</i> | Pointer to selected FLASHx peripheral |
|---------------|---------------------------------------|

### Returns

Nothing

### 13.5.2 void FLASH\_Powerdown ( FLASH\_Type \* *pFLASH* )

### Parameters

|               |                                       |
|---------------|---------------------------------------|
| <i>pFLASH</i> | Pointer to selected FLASHx peripheral |
|---------------|---------------------------------------|

### Returns

Nothing

### 13.5.3 int FLASH\_Wait ( FLASH\_Type \* *pFLASH* )

### Parameters

|               |                                       |
|---------------|---------------------------------------|
| <i>pFLASH</i> | Pointer to selected FLASHx peripheral |
|---------------|---------------------------------------|

### Returns

INT\_STATUS with ECC\_ERR bit masked out

### 13.5.4 int FLASH\_Erase ( FLASH\_Type \* *pFLASH*, uint8\_t \* *pu8Start*, uint8\_t \* *pu8End* )

### Parameters

---



|    |                 |                                      |
|----|-----------------|--------------------------------------|
|    | <i>pFLASH</i>   | Pointer to selected FLASH peripheral |
| in | <i>pu8Start</i> | Start address with page to inspect   |
| in | <i>pu8End</i>   | End address (included in check)      |

Returns

INT\_STATUS with ECC\_ERR bit masked out

See Also

[flash\\_status\\_t](#)

### 13.5.5 int FLASH\_ErasePages ( FLASH\_Type \* *pFLASH*, uint32\_t *u32StartPage*, uint32\_t *u32PageCount* )

Parameters

|    |                     |                                      |
|----|---------------------|--------------------------------------|
|    | <i>pFLASH</i>       | Pointer to selected FLASH peripheral |
| in | <i>u32StartPage</i> | Index of page to start erasing from  |
| in | <i>u32PageCount</i> | Number of pages to erase             |

Returns

INT\_STATUS with ECC\_ERR bit masked out

See Also

[flash\\_status\\_t](#)

### 13.5.6 int FLASH\_BlankCheck ( FLASH\_Type \* *pFLASH*, uint8\_t \* *pu8Start*, uint8\_t \* *pu8End* )

Parameters

---

## Function Documentation

|    |                 |                                      |
|----|-----------------|--------------------------------------|
|    | <i>pFLASH</i>   | Pointer to selected FLASH peripheral |
| in | <i>pu8Start</i> | Start address with page to inspect   |
| in | <i>pu8End</i>   | End address (included in check)      |

Returns

INT\_STATUS with ECC\_ERR bit masked out

See Also

[flash\\_status\\_t](#)

### 13.5.7 int FLASH\_MarginCheck ( FLASH\_Type \* *pFLASH*, uint8\_t \* *pu8Start*, uint8\_t \* *pu8End* )

Parameters

|    |                 |                                      |
|----|-----------------|--------------------------------------|
|    | <i>pFLASH</i>   | Pointer to selected FLASH peripheral |
| in | <i>pu8Start</i> | Start address with page to inspect   |
| in | <i>pu8End</i>   | End address (included in check)      |

Returns

INT\_STATUS with ECC\_ERR bit masked out

See Also

[flash\\_status\\_t](#)

### 13.5.8 int FLASH\_Program ( FLASH\_Type \* *pFLASH*, uint32\_t \* *pu32Start*, uint32\_t \* *pu32Data*, uint32\_t *u32Length* )

Parameters

|     |                  |                                                   |
|-----|------------------|---------------------------------------------------|
| in  | <i>pFLASH</i>    | Pointer to selected FLASH peripheral              |
| out | <i>pu32Start</i> | Pointer location that must be programmed in flash |
| in  | <i>pu32Data</i>  | Pointer to source buffer being written to flash   |
| in  | <i>u32Length</i> | Number of bytes to be programmed                  |

Returns

INT\_STATUS with ECC\_ERR bit masked out

See Also

[flash\\_status\\_t](#)

**13.5.9 int FLASH\_Checksum ( FLASH\_Type \* *pFLASH*, uint8\_t \* *pu8Start*, uint8\_t \* *pu8End*, uint32\_t *au32Checksum*[4] )**

Parameters

|     |                     |                                                                     |
|-----|---------------------|---------------------------------------------------------------------|
|     | <i>pFLASH</i>       | Pointer to selected FLASH peripheral                                |
| in  | <i>pu8Start</i>     | Pointer to data within starting page page checksum must be computed |
| in  | <i>pu8End</i>       | Pointer to data whose page is the last of the checksum calculation  |
| out | <i>au32Checksum</i> | Four 32bit word array to store checksum calculation result          |

Returns

INT\_STATUS with ECC\_ERR bit masked out

See Also

[flash\\_status\\_t](#)

**13.5.10 int FLASH\_Read ( FLASH\_Type \* *pFLASH*, uint8\_t \* *pu8Start*, uint32\_t *u32ReadMode*, uint32\_t *au32Data*[4] )**

## Function Documentation

### Parameters

|     |                    |                                            |
|-----|--------------------|--------------------------------------------|
|     | <i>pFLASH</i>      | Pointer to selected FLASH peripheral       |
| in  | <i>pu8Start</i>    | Pointer to data to be read                 |
| in  | <i>u32ReadMode</i> | Read mode see also flash_read_mode_t       |
| out | <i>au32Data</i>    | Four 32bit word array to store read result |

### Returns

INT\_STATUS with ECC\_ERR bit masked out

### See Also

[flash\\_status\\_t](#)

### 13.5.11 void FLASH\_SetReadMode ( FLASH\_Type \* *pFLASH*, bool *freq\_48M\_not\_32M* )

### Parameters

|                         |                                                           |
|-------------------------|-----------------------------------------------------------|
| <i>pFLASH</i>           | Pointer to selected FLASHx peripheral                     |
| <i>freq_48M_not_32M</i> | CPU clock frequency @48MHz - lower or equal to 32Mhz if 0 |

### Returns

Nothing

### 13.5.12 void FLASH\_CalculateChecksum ( const uint32\_t \* *input*, size\_t *nb\_128b\_words*, uint32\_t \* *misr*, int *init* )

### Parameters

|     |                      |                                                                   |
|-----|----------------------|-------------------------------------------------------------------|
| in  | <i>input</i>         | Pointer to data over which checksum calculation must be executed. |
| in  | <i>nb_128b_words</i> | Number of 16 byte words on flash.                                 |
| out | <i>misr</i>          | Pointer on a four 32bit word array to store calculated checksum.  |
| in  | <i>init</i>          | Set to true to clear the misr buffer.                             |

Returns

Nothing

### 13.5.13 int FLASH\_ConfigPageVerifyPageChecksum ( const uint32\_t \* *page\_buffer*, uint32\_t \* *misr* )

Parameters

|     |                    |                                                                                                                           |
|-----|--------------------|---------------------------------------------------------------------------------------------------------------------------|
| in  | <i>page_buffer</i> | Pointer to data over which checksum calculation must be executed.                                                         |
| out | <i>misr</i>        | Pointer on a four 32bit word array to store calculated checksum. Note: this buffer is only useful for debugging purposes. |

Returns

Result of the page checksum verification:

- 0: Verify successfully.
- -1: Verification failed.

### 13.5.14 int FLASH\_ConfigPageVerifyGpoChecksum ( const uint32\_t \* *page\_buffer*, uint32\_t \* *misr* )

Parameters

|     |                    |                                                                                                                        |
|-----|--------------------|------------------------------------------------------------------------------------------------------------------------|
| in  | <i>page_buffer</i> | Pointer to data over which checksum calculation must be executed.                                                      |
| out | <i>misr</i>        | Pointer on a 4 32bit word array to store calculated checksum. Note: this buffer is only useful for debugging purposes. |

Returns

Result of the GPO array checksum verification:

- 0: Verify successfully.
- -1: Verification failed.

**13.5.15** void FLASH\_ConfigPageUpdate ( uint32\_t \* *page\_ram\_buffer*, uint32\_t \* *gpo\_chksum*, uint32\_t \* *page\_chksum* )

## Parameters

|                        |                                                                                      |
|------------------------|--------------------------------------------------------------------------------------|
| <i>page_ram_buffer</i> | Pointer to RAM page buffer in which the read-modify-write of page (N-2) is performed |
| <i>gpo_chksum</i>      | Pointer on a four 32bit word array to store calculated checksum.                     |
| <i>page_chksum</i>     | Pointer on a four 32bit word array to store calculated checksum.                     |

## Returns

Nothing

### 13.5.16 int FLASH\_GetStatus ( FLASH\_Type \* *pFLASH* )

## Parameters

|               |                                        |
|---------------|----------------------------------------|
| <i>pFLASH</i> | Pointer to selected FLASHx peripheral. |
|---------------|----------------------------------------|

## Returns

INT\_STATUS raw value.

## See Also

[flash\\_status\\_t](#)

## 13.6 Variable Documentation

### 13.6.1 uint32\_t BOOT\_BLOCK\_T::img\_len

For faster boot application could set a smaller length than actual image. For Secure boot images, this MUST be the entire image length







## Chapter 14

### FLEXCOMM: FLEXCOMM Driver

#### 14.1 Overview

The MCUXpresso SDK provides a generic driver and multiple protocol-specific FLEXCOMM drivers for the FLEXCOMM module of MCUXpresso SDK devices.

#### Modules

- [FLEXCOMM Driver](#)

### 14.2 FLEXCOMM Driver

#### 14.2.1 Overview

##### Typedefs

- typedef void(\* [flexcomm\\_irq\\_handler\\_t](#))(void \*base, void \*handle)  
*Typedef for interrupt handler.*

##### Enumerations

- enum [FLEXCOMM\\_PERIPH\\_T](#) {  
    [FLEXCOMM\\_PERIPH\\_NONE](#),  
    [FLEXCOMM\\_PERIPH\\_USART](#),  
    [FLEXCOMM\\_PERIPH\\_SPI](#),  
    [FLEXCOMM\\_PERIPH\\_I2C](#),  
    [FLEXCOMM\\_PERIPH\\_I2S\\_TX](#),  
    [FLEXCOMM\\_PERIPH\\_I2S\\_RX](#) }  
*FLEXCOMM peripheral modes.*

##### Functions

- uint32\_t [FLEXCOMM\\_GetInstance](#) (void \*base)  
*Returns instance number for FLEXCOMM module with given base address.*
- [status\\_t](#) [FLEXCOMM\\_Init](#) (void \*base, [FLEXCOMM\\_PERIPH\\_T](#) periph)  
*Initializes FLEXCOMM and selects peripheral mode according to the second parameter.*
- void [FLEXCOMM\\_SetIRQHandler](#) (void \*base, [flexcomm\\_irq\\_handler\\_t](#) handler, void \*handle)  
*Sets IRQ handler for given FLEXCOMM module.*

##### Variables

- IRQn\_Type const [kFlexcommIrqs](#) []  
*Array with IRQ number for each FLEXCOMM module.*

##### Driver version

- #define [FSL\\_FLEXCOMM\\_DRIVER\\_VERSION](#) ([MAKE\\_VERSION](#)(2, 0, 2))  
*FlexCOMM driver version 2.0.2.*

## 14.2.2 Macro Definition Documentation

14.2.2.1 `#define FSL_FLEXCOMM_DRIVER_VERSION (MAKE_VERSION(2, 0, 2))`

## 14.2.3 Typedef Documentation

14.2.3.1 `typedef void(* flexcomm_irq_handler_t)(void *base, void *handle)`

## 14.2.4 Enumeration Type Documentation

14.2.4.1 `enum FLEXCOMM_PERIPH_T`

Enumerator

***FLEXCOMM\_PERIPH\_NONE*** No peripheral.  
***FLEXCOMM\_PERIPH\_USART*** USART peripheral.  
***FLEXCOMM\_PERIPH\_SPI*** SPI Peripheral.  
***FLEXCOMM\_PERIPH\_I2C*** I2C Peripheral.  
***FLEXCOMM\_PERIPH\_I2S\_TX*** I2S TX Peripheral.  
***FLEXCOMM\_PERIPH\_I2S\_RX*** I2S RX Peripheral.

## 14.2.5 Function Documentation

14.2.5.1 `uint32_t FLEXCOMM_GetInstance ( void * base )`

14.2.5.2 `status_t FLEXCOMM_Init ( void * base, FLEXCOMM_PERIPH_T periph )`

14.2.5.3 `void FLEXCOMM_SetIRQHandler ( void * base, flexcomm_irq_handler_t handler, void * handle )`

It is used by drivers register IRQ handler according to FLEXCOMM mode

## 14.2.6 Variable Documentation

14.2.6.1 `IRQn_Type const kFlexcommIrqs[]`



## Chapter 15

# I2C: Inter-Integrated Circuit Driver

### 15.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Inter-Integrated Circuit (I2C) module of MCUXpresso SDK devices.

The I2C driver includes functional APIs and transactional APIs.

Functional APIs are feature/property target low-level APIs. Functional APIs can be used for the I2C master/slave initialization/configuration/operation for optimization/customization purpose. Using the functional APIs requires the knowledge of the I2C master peripheral and how to organize functional APIs to meet the application requirements. The I2C functional operation groups provide the functional APIs set.

Transactional APIs are transaction target high-level APIs. The transactional APIs can be used to enable the peripheral quickly and also in the application if the code size and performance of transactional APIs satisfy the requirements. If the code size and performance are critical requirements, see the transactional API implementation and write custom code using the functional APIs or accessing the hardware registers.

Transactional APIs support asynchronous transfer. This means that the functions [I2C\\_MasterTransferNonBlocking\(\)](#) set up the interrupt non-blocking transfer. When the transfer completes, the upper layer is notified through a callback function with the status.

### 15.2 Typical use case

#### 15.2.1 Master Operation in functional method

```
i2c_master_config_t masterConfig;
uint8_t status;
status_t result = kStatus_Success;
uint8_t txBuff[BUFFER_SIZE];

/* Get default configuration for master. */
I2C_MasterGetDefaultConfig(&masterConfig);

/* Init I2C master. */
I2C_MasterInit(EXAMPLE_I2C_MASTER_BASEADDR, &masterConfig, I2C_MASTER_CLK);

/* Send start and slave address. */
I2C_MasterStart(EXAMPLE_I2C_MASTER_BASEADDR, 7-bit slave address,
 kI2C_Write/kI2C_Read);

/* Wait address sent out. */
while(!((status = I2C_GetStatusFlag(EXAMPLE_I2C_MASTER_BASEADDR)) & kI2C_IntPendingFlag))
{
}

if(status & kI2C_ReceiveNakFlag)
{
 return kStatus_I2C_Nak;
}
```

## Typical use case

```
result = I2C_MasterWriteBlocking(EXAMPLE_I2C_MASTER_BASEADDR, txBuff, BUFFER_SIZE);

if(result)
{
 /* If error occurs, send STOP. */
 I2C_MasterStop(EXAMPLE_I2C_MASTER_BASEADDR, kI2CStop);
 return result;
}

while(!(I2C_GetStatusFlag(EXAMPLE_I2C_MASTER_BASEADDR) & kI2C_IntPendingFlag))
{

}

/* Wait all data sent out, send STOP. */
I2C_MasterStop(EXAMPLE_I2C_MASTER_BASEADDR, kI2CStop);
```

### 15.2.2 Master Operation in interrupt transactional method

```
i2c_master_handle_t g_m_handle;
volatile bool g_MasterCompletionFlag = false;
i2c_master_config_t masterConfig;
uint8_t status;
status_t result = kStatus_Success;
uint8_t txBuff[BUFFER_SIZE];
i2c_master_transfer_t masterXfer;

static void i2c_master_callback(I2C_Type *base, i2c_master_handle_t *handle,
 status_t status, void *userData)
{
 /* Signal transfer success when received success status. */
 if (status == kStatus_Success)
 {
 g_MasterCompletionFlag = true;
 }
}

/* Get default configuration for master. */
I2C_MasterGetDefaultConfig(&masterConfig);

/* Init I2C master. */
I2C_MasterInit(EXAMPLE_I2C_MASTER_BASEADDR, &masterConfig, I2C_MASTER_CLK);

masterXfer.slaveAddress = I2C_MASTER_SLAVE_ADDR_7BIT;
masterXfer.direction = kI2C_Write;
masterXfer.subaddress = NULL;
masterXfer.subaddressSize = 0;
masterXfer.data = txBuff;
masterXfer.dataSize = BUFFER_SIZE;
masterXfer.flags = kI2C_TransferDefaultFlag;

I2C_MasterTransferCreateHandle(EXAMPLE_I2C_MASTER_BASEADDR, &g_m_handle,
 i2c_master_callback, NULL);
I2C_MasterTransferNonBlocking(EXAMPLE_I2C_MASTER_BASEADDR, &g_m_handle, &
 masterXfer);

/* Wait for transfer completed. */
while (!g_MasterCompletionFlag)
{
}
g_MasterCompletionFlag = false;
```

### 15.2.3 Master Operation in DMA transactional method

```

i2c_master_dma_handle_t g_m_dma_handle;
dma_handle_t dmaHandle;
volatile bool g_MasterCompletionFlag = false;
i2c_master_config_t masterConfig;
uint8_t txBuff[BUFFER_SIZE];
i2c_master_transfer_t masterXfer;

static void i2c_master_callback(I2C_Type *base, i2c_master_dma_handle_t *handle,
 status_t status, void *userData)
{
 /* Signal transfer success when received success status. */
 if (status == kStatus_Success)
 {
 g_MasterCompletionFlag = true;
 }
}

/* Get default configuration for master. */
I2C_MasterGetDefaultConfig(&masterConfig);

/* Init I2C master. */
I2C_MasterInit(EXAMPLE_I2C_MASTER_BASEADDR, &masterConfig, I2C_MASTER_CLK);

masterXfer.slaveAddress = I2C_MASTER_SLAVE_ADDR_7BIT;
masterXfer.direction = kI2C_Write;
masterXfer.subaddress = NULL;
masterXfer.subaddressSize = 0;
masterXfer.data = txBuff;
masterXfer.dataSize = BUFFER_SIZE;
masterXfer.flags = kI2C_TransferDefaultFlag;

DMA_EnableChannel(EXAMPLE_DMA, EXAMPLE_I2C_MASTER_CHANNEL);
DMA_CreateHandle(&dmaHandle, EXAMPLE_DMA, EXAMPLE_I2C_MASTER_CHANNEL);

I2C_MasterTransferCreateHandleDMA(EXAMPLE_I2C_MASTER_BASEADDR, &
 g_m_dma_handle, i2c_master_callback, NULL, &dmaHandle);
I2C_MasterTransferDMA(EXAMPLE_I2C_MASTER_BASEADDR, &g_m_dma_handle, &masterXfer);

/* Wait for transfer completed. */
while (!g_MasterCompletionFlag)
{
}
g_MasterCompletionFlag = false;

```

### 15.2.4 Slave Operation in functional method

```

i2c_slave_config_t slaveConfig;
uint8_t status;
status_t result = kStatus_Success;

I2C_SlaveGetDefaultConfig(&slaveConfig); /*default configuration 7-bit addressing
 mode*/
slaveConfig.slaveAddr = 7-bit address
slaveConfig.addressingMode = kI2C_Address7bit/kI2C_RangeMatch;
I2C_SlaveInit(EXAMPLE_I2C_SLAVE_BASEADDR, &slaveConfig);

/* Wait address match. */
while(!((status = I2C_GetStatusFlag(EXAMPLE_I2C_SLAVE_BASEADDR)) & kI2C_AddressMatchFlag))
{
}

```

## Typical use case

```
/* Slave transmit, master reading from slave. */
if (status & kI2C_TransferDirectionFlag)
{
 result = I2C_SlaveWriteBlocking(EXAMPLE_I2C_SLAVE_BASEADDR);
}
else
{
 I2C_SlaveReadBlocking(EXAMPLE_I2C_SLAVE_BASEADDR);
}

return result;
```

### 15.2.5 Slave Operation in interrupt transactional method

```
i2c_slave_config_t slaveConfig;
i2c_slave_handle_t g_s_handle;
volatile bool g_SlaveCompletionFlag = false;

static void i2c_slave_callback(I2C_Type *base, i2c_slave_transfer_t *xfer, void *
 userData)
{
 switch (xfer->event)
 {
 /* Transmit request */
 case kI2C_SlaveTransmitEvent:
 /* Update information for transmit process */
 xfer->data = g_slave_buff;
 xfer->dataSize = I2C_DATA_LENGTH;
 break;

 /* Receive request */
 case kI2C_SlaveReceiveEvent:
 /* Update information for received process */
 xfer->data = g_slave_buff;
 xfer->dataSize = I2C_DATA_LENGTH;
 break;

 /* Transfer done */
 case kI2C_SlaveCompletionEvent:
 g_SlaveCompletionFlag = true;
 break;

 default:
 g_SlaveCompletionFlag = true;
 break;
 }
}

I2C_SlaveGetDefaultConfig(&slaveConfig); /*default configuration 7-bit addressing
 mode*/
slaveConfig.slaveAddr = 7-bit address
slaveConfig.addressingMode = kI2C_Address7bit/kI2C_RangeMatch;

I2C_SlaveInit(EXAMPLE_I2C_SLAVE_BASEADDR, &slaveConfig);

I2C_SlaveTransferCreateHandle(EXAMPLE_I2C_SLAVE_BASEADDR, &g_s_handle,
 i2c_slave_callback, NULL);

I2C_SlaveTransferNonBlocking(EXAMPLE_I2C_SLAVE_BASEADDR, &g_s_handle,
 kI2C_SlaveCompletionEvent);

/* Wait for transfer completed. */
while (!g_SlaveCompletionFlag)
{
}
```



```
g_SlaveCompletionFlag = false;
```

## Modules

- [I2C DMA Driver](#)
- [I2C Driver](#)
- [I2C FreeRTOS Driver](#)
- [I2C Master Driver](#)
- [I2C Slave Driver](#)

## I2C Driver

### 15.3 I2C Driver

#### 15.3.1 Overview

##### Files

- file [fsl\\_i2c.h](#)

##### Macros

- `#define I2C_RETRY_TIMES 0U` /\* Define to zero means keep waiting until the flag is asserted/deassert. \*/  
*Retry times for waiting flag.*
- `#define I2C_STAT_MSTCODE_IDLE (0)`  
*Master Idle State Code.*
- `#define I2C_STAT_MSTCODE_RXREADY (1)`  
*Master Receive Ready State Code.*
- `#define I2C_STAT_MSTCODE_TXREADY (2)`  
*Master Transmit Ready State Code.*
- `#define I2C_STAT_MSTCODE_NACKADR (3)`  
*Master NACK by slave on address State Code.*
- `#define I2C_STAT_MSTCODE_NACKDAT (4)`  
*Master NACK by slave on data State Code.*

##### Enumerations

- `enum _i2c_status {`  
`kStatus_I2C_Busy = MAKE_STATUS(kStatusGroup_FLEXCOMM_I2C, 0),`  
`kStatus_I2C_Idle = MAKE_STATUS(kStatusGroup_FLEXCOMM_I2C, 1),`  
`kStatus_I2C_Nak,`  
`kStatus_I2C_InvalidParameter,`  
`kStatus_I2C_BitError = MAKE_STATUS(kStatusGroup_FLEXCOMM_I2C, 4),`  
`kStatus_I2C_ArbitrationLost = MAKE_STATUS(kStatusGroup_FLEXCOMM_I2C, 5),`  
`kStatus_I2C_NoTransferInProgress,`  
`kStatus_I2C_DmaRequestFail = MAKE_STATUS(kStatusGroup_FLEXCOMM_I2C, 7),`  
`kStatus_I2C_Timeout = MAKE_STATUS(kStatusGroup_FLEXCOMM_I2C, 10),`  
`kStatus_I2C_Addr_Nak = MAKE_STATUS(kStatusGroup_FLEXCOMM_I2C, 11) }`  
*I2C status return codes.*

##### Driver version

- `#define FSL_I2C_DRIVER_VERSION (MAKE_VERSION(2, 0, 6))`  
*I2C driver version 2.0.6.*

## 15.3.2 Macro Definition Documentation

15.3.2.1 **#define FSL\_I2C\_DRIVER\_VERSION (MAKE\_VERSION(2, 0, 6))**

15.3.2.2 **#define I2C\_RETRY\_TIMES 0U /\* Define to zero means keep waiting until the flag is assert/deassert. \*/**

## 15.3.3 Enumeration Type Documentation

### 15.3.3.1 enum \_i2c\_status

Enumerator

***kStatus\_I2C\_Busy*** The master is already performing a transfer.

***kStatus\_I2C\_Idle*** The slave driver is idle.

***kStatus\_I2C\_Nak*** The slave device sent a NAK in response to a byte.

***kStatus\_I2C\_InvalidParameter*** Unable to proceed due to invalid parameter.

***kStatus\_I2C\_BitError*** Transferred bit was not seen on the bus.

***kStatus\_I2C\_ArbitrationLost*** Arbitration lost error.

***kStatus\_I2C\_NoTransferInProgress*** Attempt to abort a transfer when one is not in progress.

***kStatus\_I2C\_DmaRequestFail*** DMA request failed.

***kStatus\_I2C\_Timeout*** Timeout polling status flags.

***kStatus\_I2C\_Addr\_Nak*** NAK received for Address.

## I2C Master Driver

### 15.4 I2C Master Driver

#### 15.4.1 Overview

##### Data Structures

- struct [i2c\\_master\\_config\\_t](#)  
*Structure with settings to initialize the I2C master module. [More...](#)*
- struct [i2c\\_master\\_transfer\\_t](#)  
*Non-blocking transfer descriptor structure. [More...](#)*
- struct [i2c\\_master\\_handle\\_t](#)  
*Driver handle for master non-blocking APIs. [More...](#)*

##### Typedefs

- typedef void(\* [i2c\\_master\\_transfer\\_callback\\_t](#) )(I2C\_Type \*base, i2c\_master\_handle\_t \*handle, [status\\_t](#) completionStatus, void \*userData)  
*Master completion callback function pointer type.*

##### Enumerations

- enum [\\_i2c\\_master\\_flags](#) {  
    [kI2C\\_MasterPendingFlag](#) = I2C\_STAT\_MSTPENDING\_MASK,  
    [kI2C\\_MasterArbitrationLostFlag](#),  
    [kI2C\\_MasterStartStopErrorFlag](#) }  
*I2C master peripheral flags.*
- enum [i2c\\_direction\\_t](#) {  
    [kI2C\\_Write](#) = 0U,  
    [kI2C\\_Read](#) = 1U }  
*Direction of master and slave transfers.*
- enum [\\_i2c\\_master\\_transfer\\_flags](#) {  
    [kI2C\\_TransferDefaultFlag](#) = 0x00U,  
    [kI2C\\_TransferNoStartFlag](#) = 0x01U,  
    [kI2C\\_TransferRepeatedStartFlag](#) = 0x02U,  
    [kI2C\\_TransferNoStopFlag](#) = 0x04U }  
*Transfer option flags.*
- enum [\\_i2c\\_transfer\\_states](#)  
*States for the state machine used by transactional APIs.*

##### Initialization and deinitialization

- void [I2C\\_MasterGetDefaultConfig](#) ([i2c\\_master\\_config\\_t](#) \*masterConfig)  
*Provides a default configuration for the I2C master peripheral.*
- void [I2C\\_MasterInit](#) (I2C\_Type \*base, const [i2c\\_master\\_config\\_t](#) \*masterConfig, uint32\_t src-Clock\_Hz)

- *Initializes the I2C master peripheral.*
- void [I2C\\_MasterDeinit](#) (I2C\_Type \*base)
- *Deinitializes the I2C master peripheral.*
- uint32\_t [I2C\\_GetInstance](#) (I2C\_Type \*base)
- *Returns an instance number given a base address.*
- static void [I2C\\_MasterReset](#) (I2C\_Type \*base)
- *Performs a software reset.*
- static void [I2C\\_MasterEnable](#) (I2C\_Type \*base, bool enable)
- *Enables or disables the I2C module as master.*

## Status

- static uint32\_t [I2C\\_GetStatusFlags](#) (I2C\_Type \*base)
- *Gets the I2C status flags.*
- static void [I2C\\_MasterClearStatusFlags](#) (I2C\_Type \*base, uint32\_t statusMask)
- *Clears the I2C master status flag state.*

## Interrupts

- static void [I2C\\_EnableInterrupts](#) (I2C\_Type \*base, uint32\_t interruptMask)
- *Enables the I2C master interrupt requests.*
- static void [I2C\\_DisableInterrupts](#) (I2C\_Type \*base, uint32\_t interruptMask)
- *Disables the I2C master interrupt requests.*
- static uint32\_t [I2C\\_GetEnabledInterrupts](#) (I2C\_Type \*base)
- *Returns the set of currently enabled I2C master interrupt requests.*

## Bus operations

- void [I2C\\_MasterSetBaudRate](#) (I2C\_Type \*base, uint32\_t baudRate\_Bps, uint32\_t srcClock\_Hz)
- *Sets the I2C bus frequency for master transactions.*
- static bool [I2C\\_MasterGetBusIdleState](#) (I2C\_Type \*base)
- *Returns whether the bus is idle.*
- status\_t [I2C\\_MasterStart](#) (I2C\_Type \*base, uint8\_t address, i2c\_direction\_t direction)
- *Sends a START on the I2C bus.*
- status\_t [I2C\\_MasterStop](#) (I2C\_Type \*base)
- *Sends a STOP signal on the I2C bus.*
- static status\_t [I2C\\_MasterRepeatedStart](#) (I2C\_Type \*base, uint8\_t address, i2c\_direction\_t direction)
- *Sends a REPEATED START on the I2C bus.*
- status\_t [I2C\\_MasterWriteBlocking](#) (I2C\_Type \*base, const void \*txBuff, size\_t txSize, uint32\_t flags)
- *Performs a polling send transfer on the I2C bus.*
- status\_t [I2C\\_MasterReadBlocking](#) (I2C\_Type \*base, void \*rxBuff, size\_t rxSize, uint32\_t flags)
- *Performs a polling receive transfer on the I2C bus.*
- status\_t [I2C\\_MasterTransferBlocking](#) (I2C\_Type \*base, i2c\_master\_transfer\_t \*xfer)
- *Performs a master polling transfer on the I2C bus.*

## I2C Master Driver

### Non-blocking

- void [I2C\\_MasterTransferCreateHandle](#) (I2C\_Type \*base, i2c\_master\_handle\_t \*handle, [i2c\\_master\\_transfer\\_callback\\_t](#) callback, void \*userData)  
*Creates a new handle for the I2C master non-blocking APIs.*
- [status\\_t I2C\\_MasterTransferNonBlocking](#) (I2C\_Type \*base, i2c\_master\_handle\_t \*handle, [i2c\\_master\\_transfer\\_t](#) \*xfer)  
*Performs a non-blocking transaction on the I2C bus.*
- [status\\_t I2C\\_MasterTransferGetCount](#) (I2C\_Type \*base, i2c\_master\_handle\_t \*handle, size\_t \*count)  
*Returns number of bytes transferred so far.*
- [status\\_t I2C\\_MasterTransferAbort](#) (I2C\_Type \*base, i2c\_master\_handle\_t \*handle)  
*Terminates a non-blocking I2C master transmission early.*

### IRQ handler

- void [I2C\\_MasterTransferHandleIRQ](#) (I2C\_Type \*base, i2c\_master\_handle\_t \*handle)  
*Reusable routine to handle master interrupts.*

## 15.4.2 Data Structure Documentation

### 15.4.2.1 struct i2c\_master\_config\_t

This structure holds configuration settings for the I2C peripheral. To initialize this structure to reasonable defaults, call the [I2C\\_MasterGetDefaultConfig\(\)](#) function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.

### Data Fields

- bool [enableMaster](#)  
*Whether to enable master mode.*
- uint32\_t [baudRate\\_Bps](#)  
*Desired baud rate in bits per second.*
- bool [enableTimeout](#)  
*Enable internal timeout function.*

**15.4.2.1.0.11 Field Documentation****15.4.2.1.0.11.1** `bool i2c_master_config_t::enableMaster`**15.4.2.1.0.11.2** `uint32_t i2c_master_config_t::baudRate_Bps`**15.4.2.1.0.11.3** `bool i2c_master_config_t::enableTimeout`**15.4.2.2 struct \_i2c\_master\_transfer**

I2C master transfer typedef.

This structure is used to pass transaction parameters to the [I2C\\_MasterTransferNonBlocking\(\)](#) API.**Data Fields**

- `uint32_t flags`  
*Bit mask of options for the transfer.*
- `uint16_t slaveAddress`  
*The 7-bit slave address.*
- `i2c_direction_t direction`  
*Either [kI2C\\_Read](#) or [kI2C\\_Write](#).*
- `uint32_t subaddress`  
*Sub address.*
- `size_t subaddressSize`  
*Length of sub address to send in bytes.*
- `void * data`  
*Pointer to data to transfer.*
- `size_t dataSize`  
*Number of bytes to transfer.*

**15.4.2.2.0.12 Field Documentation****15.4.2.2.0.12.1** `uint32_t i2c_master_transfer_t::flags`See enumeration [\\_i2c\\_master\\_transfer\\_flags](#) for available options. Set to 0 or [kI2C\\_TransferDefaultFlag](#) for normal transfers.**15.4.2.2.0.12.2** `uint16_t i2c_master_transfer_t::slaveAddress`**15.4.2.2.0.12.3** `i2c_direction_t i2c_master_transfer_t::direction`**15.4.2.2.0.12.4** `uint32_t i2c_master_transfer_t::subaddress`

Transferred MSB first.

**15.4.2.2.0.12.5** `size_t i2c_master_transfer_t::subaddressSize`

Maximum size is 4 bytes.

## I2C Master Driver

15.4.2.2.0.12.6 void\* i2c\_master\_transfer\_t::data

15.4.2.2.0.12.7 size\_t i2c\_master\_transfer\_t::dataSize

### 15.4.2.3 struct \_i2c\_master\_handle

I2C master handle typedef.

Note

The contents of this structure are private and subject to change.

#### Data Fields

- uint8\_t [state](#)  
*Transfer state machine current state.*
- uint32\_t [transferCount](#)  
*Indicates progress of the transfer.*
- uint32\_t [remainingBytes](#)  
*Remaining byte count in current state.*
- uint8\_t \* [buf](#)  
*Buffer pointer for current state.*
- i2c\_master\_transfer\_t [transfer](#)  
*Copy of the current transfer info.*
- [i2c\\_master\\_transfer\\_callback\\_t](#) [completionCallback](#)  
*Callback function pointer.*
- void \* [userData](#)  
*Application data passed to callback.*



#### 15.4.2.3.0.13 Field Documentation

15.4.2.3.0.13.1 `uint8_t i2c_master_handle_t::state`

15.4.2.3.0.13.2 `uint32_t i2c_master_handle_t::remainingBytes`

15.4.2.3.0.13.3 `uint8_t* i2c_master_handle_t::buf`

15.4.2.3.0.13.4 `i2c_master_transfer_t i2c_master_handle_t::transfer`

15.4.2.3.0.13.5 `i2c_master_transfer_callback_t i2c_master_handle_t::completionCallback`

15.4.2.3.0.13.6 `void* i2c_master_handle_t::userData`

#### 15.4.3 Typedef Documentation

15.4.3.1 `typedef void(* i2c_master_transfer_callback_t)(I2C_Type *base,  
i2c_master_handle_t *handle, status_t completionStatus, void *userData)`

This callback is used only for the non-blocking master transfer API. Specify the callback you wish to use in the call to [I2C\\_MasterTransferCreateHandle\(\)](#).

## I2C Master Driver

### Parameters

|                          |                                                                                |
|--------------------------|--------------------------------------------------------------------------------|
| <i>base</i>              | The I2C peripheral base address.                                               |
| <i>completion-Status</i> | Either kStatus_Success or an error code describing how the transfer completed. |
| <i>userData</i>          | Arbitrary pointer-sized value passed from the application.                     |

## 15.4.4 Enumeration Type Documentation

### 15.4.4.1 enum \_i2c\_master\_flags

#### Note

These enums are meant to be OR'd together to form a bit mask.

#### Enumerator

***kI2C\_MasterPendingFlag*** The I2C module is waiting for software interaction.

***kI2C\_MasterArbitrationLostFlag*** The arbitration of the bus was lost. There was collision on the bus

***kI2C\_MasterStartStopErrorFlag*** There was an error during start or stop phase of the transaction.

### 15.4.4.2 enum i2c\_direction\_t

#### Enumerator

***kI2C\_Write*** Master transmit.

***kI2C\_Read*** Master receive.

### 15.4.4.3 enum \_i2c\_master\_transfer\_flags

#### Note

These enumerations are intended to be OR'd together to form a bit mask of options for the [\\_i2c\\_master\\_transfer::flags](#) field.

#### Enumerator

***kI2C\_TransferDefaultFlag*** Transfer starts with a start signal, stops with a stop signal.

***kI2C\_TransferNoStartFlag*** Don't send a start condition, address, and sub address.

***kI2C\_TransferRepeatedStartFlag*** Send a repeated start condition.

***kI2C\_TransferNoStopFlag*** Don't send a stop condition.

#### 15.4.4.4 enum \_i2c\_transfer\_states

### 15.4.5 Function Documentation

#### 15.4.5.1 void I2C\_MasterGetDefaultConfig ( i2c\_master\_config\_t \* *masterConfig* )

This function provides the following default configuration for the I2C master peripheral:

```
* masterConfig->enableMaster = true;
* masterConfig->baudRate_Bps = 100000U;
* masterConfig->enableTimeout = false;
*
```

After calling this function, you can override any settings in order to customize the configuration, prior to initializing the master driver with [I2C\\_MasterInit\(\)](#).

Parameters

|     |                     |                                                                                                          |
|-----|---------------------|----------------------------------------------------------------------------------------------------------|
| out | <i>masterConfig</i> | User provided configuration structure for default values. Refer to <a href="#">i2c_master_config_t</a> . |
|-----|---------------------|----------------------------------------------------------------------------------------------------------|

#### 15.4.5.2 void I2C\_MasterInit ( I2C\_Type \* *base*, const i2c\_master\_config\_t \* *masterConfig*, uint32\_t *srcClock\_Hz* )

This function enables the peripheral clock and initializes the I2C master peripheral as described by the user provided configuration. A software reset is performed prior to configuration.

Parameters

|                     |                                                                                                                                          |
|---------------------|------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>         | The I2C peripheral base address.                                                                                                         |
| <i>masterConfig</i> | User provided peripheral configuration. Use <a href="#">I2C_MasterGetDefaultConfig()</a> to get a set of defaults that you can override. |
| <i>srcClock_Hz</i>  | Frequency in Hertz of the I2C functional clock. Used to calculate the baud rate divisors, filter widths, and timeout periods.            |

#### 15.4.5.3 void I2C\_MasterDeinit ( I2C\_Type \* *base* )

This function disables the I2C master peripheral and gates the clock. It also performs a software reset to restore the peripheral to reset conditions.

## I2C Master Driver

### Parameters

|             |                                  |
|-------------|----------------------------------|
| <i>base</i> | The I2C peripheral base address. |
|-------------|----------------------------------|

#### 15.4.5.4 `uint32_t I2C_GetInstance ( I2C_Type * base )`

If an invalid base address is passed, debug builds will assert. Release builds will just return instance number 0.

### Parameters

|             |                                  |
|-------------|----------------------------------|
| <i>base</i> | The I2C peripheral base address. |
|-------------|----------------------------------|

### Returns

I2C instance number starting from 0.

#### 15.4.5.5 `static void I2C_MasterReset ( I2C_Type * base ) [inline], [static]`

Restores the I2C master peripheral to reset conditions.

### Parameters

|             |                                  |
|-------------|----------------------------------|
| <i>base</i> | The I2C peripheral base address. |
|-------------|----------------------------------|

#### 15.4.5.6 `static void I2C_MasterEnable ( I2C_Type * base, bool enable ) [inline], [static]`

### Parameters

|               |                                                                      |
|---------------|----------------------------------------------------------------------|
| <i>base</i>   | The I2C peripheral base address.                                     |
| <i>enable</i> | Pass true to enable or false to disable the specified I2C as master. |

#### 15.4.5.7 `static uint32_t I2C_GetStatusFlags ( I2C_Type * base ) [inline], [static]`

A bit mask with the state of all I2C status flags is returned. For each flag, the corresponding bit in the return value is set if the flag is asserted.

## Parameters

|             |                                  |
|-------------|----------------------------------|
| <i>base</i> | The I2C peripheral base address. |
|-------------|----------------------------------|

## Returns

State of the status flags:

- 1: related status flag is set.
- 0: related status flag is not set.

## See Also

[\\_i2c\\_master\\_flags](#)

**15.4.5.8** `static void I2C_MasterClearStatusFlags ( I2C_Type * base, uint32_t statusMask ) [inline], [static]`

The following status register flags can be cleared:

- [kI2C\\_MasterArbitrationLostFlag](#)
- [kI2C\\_MasterStartStopErrorFlag](#)

Attempts to clear other flags has no effect.

## Parameters

|                   |                                                                                                                                                                                                                             |
|-------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>       | The I2C peripheral base address.                                                                                                                                                                                            |
| <i>statusMask</i> | A bitmask of status flags that are to be cleared. The mask is composed of <a href="#">_i2c_master_flags</a> enumerators OR'd together. You may pass the result of a previous call to <a href="#">I2C_GetStatusFlags()</a> . |

## See Also

[\\_i2c\\_master\\_flags](#).

**15.4.5.9** `static void I2C_EnableInterrupts ( I2C_Type * base, uint32_t interruptMask ) [inline], [static]`

## I2C Master Driver

### Parameters

|                      |                                                                                                                                                     |
|----------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>          | The I2C peripheral base address.                                                                                                                    |
| <i>interruptMask</i> | Bit mask of interrupts to enable. See <a href="#">_i2c_master_flags</a> for the set of constants that should be OR'd together to form the bit mask. |

**15.4.5.10** `static void I2C_DisableInterrupts ( I2C_Type * base, uint32_t interruptMask )`  
`[inline], [static]`

### Parameters

|                      |                                                                                                                                                      |
|----------------------|------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>          | The I2C peripheral base address.                                                                                                                     |
| <i>interruptMask</i> | Bit mask of interrupts to disable. See <a href="#">_i2c_master_flags</a> for the set of constants that should be OR'd together to form the bit mask. |

**15.4.5.11** `static uint32_t I2C_GetEnabledInterrupts ( I2C_Type * base )` `[inline],`  
`[static]`

### Parameters

|             |                                  |
|-------------|----------------------------------|
| <i>base</i> | The I2C peripheral base address. |
|-------------|----------------------------------|

### Returns

A bitmask composed of [\\_i2c\\_master\\_flags](#) enumerators OR'd together to indicate the set of enabled interrupts.

**15.4.5.12** `void I2C_MasterSetBaudRate ( I2C_Type * base, uint32_t baudRate_Bps,  
uint32_t srcClock_Hz )`

The I2C master is automatically disabled and re-enabled as necessary to configure the baud rate. Do not call this function during a transfer, or the transfer is aborted.

### Parameters

---

|                     |                                             |
|---------------------|---------------------------------------------|
| <i>base</i>         | The I2C peripheral base address.            |
| <i>srcClock_Hz</i>  | I2C functional clock frequency in Hertz.    |
| <i>baudRate_Bps</i> | Requested bus frequency in bits per second. |

#### 15.4.5.13 **static bool I2C\_MasterGetBusIdleState ( I2C\_Type \* *base* ) [inline], [static]**

Requires the master mode to be enabled.

Parameters

|             |                                  |
|-------------|----------------------------------|
| <i>base</i> | The I2C peripheral base address. |
|-------------|----------------------------------|

Return values

|              |              |
|--------------|--------------|
| <i>true</i>  | Bus is busy. |
| <i>false</i> | Bus is idle. |

#### 15.4.5.14 **status\_t I2C\_MasterStart ( I2C\_Type \* *base*, uint8\_t *address*, i2c\_direction\_t *direction* )**

This function is used to initiate a new master mode transfer by sending the START signal. The slave address is sent following the I2C START signal.

Parameters

|                  |                                               |
|------------------|-----------------------------------------------|
| <i>base</i>      | I2C peripheral base pointer                   |
| <i>address</i>   | 7-bit slave device address.                   |
| <i>direction</i> | Master transfer directions(transmit/receive). |

Return values

|                         |                                     |
|-------------------------|-------------------------------------|
| <i>kStatus_Success</i>  | Successfully send the start signal. |
| <i>kStatus_I2C_Busy</i> | Current bus is busy.                |

#### 15.4.5.15 **status\_t I2C\_MasterStop ( I2C\_Type \* *base* )**

## I2C Master Driver

Return values

|                            |                                    |
|----------------------------|------------------------------------|
| <i>kStatus_Success</i>     | Successfully send the stop signal. |
| <i>kStatus_I2C_Timeout</i> | Send stop signal failed, timeout.  |

**15.4.5.16** `static status_t I2C_MasterRepeatedStart ( I2C_Type * base, uint8_t address, i2c_direction_t direction ) [inline], [static]`

Parameters

|                  |                                               |
|------------------|-----------------------------------------------|
| <i>base</i>      | I2C peripheral base pointer                   |
| <i>address</i>   | 7-bit slave device address.                   |
| <i>direction</i> | Master transfer directions(transmit/receive). |

Return values

|                         |                                                             |
|-------------------------|-------------------------------------------------------------|
| <i>kStatus_Success</i>  | Successfully send the start signal.                         |
| <i>kStatus_I2C_Busy</i> | Current bus is busy but not occupied by current I2C master. |

**15.4.5.17** `status_t I2C_MasterWriteBlocking ( I2C_Type * base, const void * txBuff, size_t txSize, uint32_t flags )`

Sends up to *txSize* number of bytes to the previously addressed slave device. The slave may reply with a NAK to any byte in order to terminate the transfer early. If this happens, this function returns [kStatus\\_I2C\\_Nak](#).

Parameters

|               |                                                                                                                                     |
|---------------|-------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>   | The I2C peripheral base address.                                                                                                    |
| <i>txBuff</i> | The pointer to the data to be transferred.                                                                                          |
| <i>txSize</i> | The length in bytes of the data to be transferred.                                                                                  |
| <i>flags</i>  | Transfer control flag to control special behavior like suppressing start or stop, for normal transfers use kI2C_TransferDefaultFlag |

Return values

---



|                                     |                                                    |
|-------------------------------------|----------------------------------------------------|
| <i>kStatus_Success</i>              | Data was sent successfully.                        |
| <i>kStatus_I2C_Busy</i>             | Another master is currently utilizing the bus.     |
| <i>kStatus_I2C_Nak</i>              | The slave device sent a NAK in response to a byte. |
| <i>kStatus_I2C_Arbitration-Lost</i> | Arbitration lost error.                            |

#### 15.4.5.18 **status\_t I2C\_MasterReadBlocking ( I2C\_Type \* *base*, void \* *rxBuff*, size\_t *rxSize*, uint32\_t *flags* )**

Parameters

|               |                                                                                                                                     |
|---------------|-------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>   | The I2C peripheral base address.                                                                                                    |
| <i>rxBuff</i> | The pointer to the data to be transferred.                                                                                          |
| <i>rxSize</i> | The length in bytes of the data to be transferred.                                                                                  |
| <i>flags</i>  | Transfer control flag to control special behavior like suppressing start or stop, for normal transfers use kI2C_TransferDefaultFlag |

Return values

|                                     |                                                    |
|-------------------------------------|----------------------------------------------------|
| <i>kStatus_Success</i>              | Data was received successfully.                    |
| <i>kStatus_I2C_Busy</i>             | Another master is currently utilizing the bus.     |
| <i>kStatus_I2C_Nak</i>              | The slave device sent a NAK in response to a byte. |
| <i>kStatus_I2C_Arbitration-Lost</i> | Arbitration lost error.                            |

#### 15.4.5.19 **status\_t I2C\_MasterTransferBlocking ( I2C\_Type \* *base*, i2c\_master\_transfer\_t \* *xfer* )**

Note

The API does not return until the transfer succeeds or fails due to arbitration lost or receiving a NAK.

Parameters

|             |                                    |
|-------------|------------------------------------|
| <i>base</i> | I2C peripheral base address.       |
| <i>xfer</i> | Pointer to the transfer structure. |

## I2C Master Driver

### Return values

|                                     |                                              |
|-------------------------------------|----------------------------------------------|
| <i>kStatus_Success</i>              | Successfully complete the data transmission. |
| <i>kStatus_I2C_Busy</i>             | Previous transmission still not finished.    |
| <i>kStatus_I2C_Timeout</i>          | Transfer error, wait signal timeout.         |
| <i>kStatus_I2C_Arbitration-Lost</i> | Transfer error, arbitration lost.            |
| <i>kStatus_I2C_Nak</i>              | Transfer error, receive NAK during transfer. |

#### 15.4.5.20 void I2C\_MasterTransferCreateHandle ( I2C\_Type \* *base*, i2c\_master\_handle\_t \* *handle*, i2c\_master\_transfer\_callback\_t *callback*, void \* *userData* )

The creation of a handle is for use with the non-blocking APIs. Once a handle is created, there is not a corresponding destroy handle. If the user wants to terminate a transfer, the [I2C\\_MasterTransferAbort\(\)](#) API shall be called.

### Parameters

|     |                 |                                                              |
|-----|-----------------|--------------------------------------------------------------|
|     | <i>base</i>     | The I2C peripheral base address.                             |
| out | <i>handle</i>   | Pointer to the I2C master driver handle.                     |
|     | <i>callback</i> | User provided pointer to the asynchronous callback function. |
|     | <i>userData</i> | User provided pointer to the application callback data.      |

#### 15.4.5.21 status\_t I2C\_MasterTransferNonBlocking ( I2C\_Type \* *base*, i2c\_master\_handle\_t \* *handle*, i2c\_master\_transfer\_t \* *xfer* )

### Parameters

|               |                                          |
|---------------|------------------------------------------|
| <i>base</i>   | The I2C peripheral base address.         |
| <i>handle</i> | Pointer to the I2C master driver handle. |
| <i>xfer</i>   | The pointer to the transfer descriptor.  |

### Return values

|                         |                                                                                                             |
|-------------------------|-------------------------------------------------------------------------------------------------------------|
| <i>kStatus_Success</i>  | The transaction was started successfully.                                                                   |
| <i>kStatus_I2C_Busy</i> | Either another master is currently utilizing the bus, or a non-blocking transaction is already in progress. |

**15.4.5.22** `status_t I2C_MasterTransferGetCount ( I2C_Type * base, i2c_master_handle_t * handle, size_t * count )`

## I2C Master Driver

### Parameters

|     |               |                                                                     |
|-----|---------------|---------------------------------------------------------------------|
|     | <i>base</i>   | The I2C peripheral base address.                                    |
|     | <i>handle</i> | Pointer to the I2C master driver handle.                            |
| out | <i>count</i>  | Number of bytes transferred so far by the non-blocking transaction. |

### Return values

|                         |  |
|-------------------------|--|
| <i>kStatus_Success</i>  |  |
| <i>kStatus_I2C_Busy</i> |  |

#### 15.4.5.23 **status\_t I2C\_MasterTransferAbort ( I2C\_Type \* *base*, i2c\_master\_handle\_t \* *handle* )**

### Note

It is not safe to call this function from an IRQ handler that has a higher priority than the I2C peripheral's IRQ priority.

### Parameters

|               |                                          |
|---------------|------------------------------------------|
| <i>base</i>   | The I2C peripheral base address.         |
| <i>handle</i> | Pointer to the I2C master driver handle. |

### Return values

|                            |                                         |
|----------------------------|-----------------------------------------|
| <i>kStatus_Success</i>     | A transaction was successfully aborted. |
| <i>kStatus_I2C_Timeout</i> | Timeout during polling for flags.       |

#### 15.4.5.24 **void I2C\_MasterTransferHandleIRQ ( I2C\_Type \* *base*, i2c\_master\_handle\_t \* *handle* )**

### Note

This function does not need to be called unless you are reimplementing the nonblocking API's interrupt handler routines to add special functionality.

## Parameters

|               |                                          |
|---------------|------------------------------------------|
| <i>base</i>   | The I2C peripheral base address.         |
| <i>handle</i> | Pointer to the I2C master driver handle. |

## I2C Slave Driver

### 15.5 I2C Slave Driver

#### 15.5.1 Overview

#### Data Structures

- struct [i2c\\_slave\\_address\\_t](#)  
*Data structure with 7-bit Slave address and Slave address disable. [More...](#)*
- struct [i2c\\_slave\\_config\\_t](#)  
*Structure with settings to initialize the I2C slave module. [More...](#)*
- struct [i2c\\_slave\\_transfer\\_t](#)  
*I2C slave transfer structure. [More...](#)*
- struct [i2c\\_slave\\_handle\\_t](#)  
*I2C slave handle structure. [More...](#)*

#### Typedefs

- typedef void(\* [i2c\\_slave\\_transfer\\_callback\\_t](#) )(I2C\_Type \*base, volatile [i2c\\_slave\\_transfer\\_t](#) \*transfer, void \*userData)  
*Slave event callback function pointer type.*

#### Enumerations

- enum [\\_i2c\\_slave\\_flags](#) {  
    [kI2C\\_SlavePendingFlag](#) = I2C\_STAT\_SLVPENDING\_MASK,  
    [kI2C\\_SlaveNotStretching](#),  
    [kI2C\\_SlaveSelected](#) = I2C\_STAT\_SLVSEL\_MASK,  
    [kI2C\\_SaveDeselected](#) }  
*I2C slave peripheral flags.*
- enum [i2c\\_slave\\_address\\_register\\_t](#) {  
    [kI2C\\_SlaveAddressRegister0](#) = 0U,  
    [kI2C\\_SlaveAddressRegister1](#) = 1U,  
    [kI2C\\_SlaveAddressRegister2](#) = 2U,  
    [kI2C\\_SlaveAddressRegister3](#) = 3U }  
*I2C slave address register.*
- enum [i2c\\_slave\\_address\\_qual\\_mode\\_t](#) {  
    [kI2C\\_QualModeMask](#) = 0U,  
    [kI2C\\_QualModeExtend](#) }  
*I2C slave address match options.*
- enum [i2c\\_slave\\_bus\\_speed\\_t](#)  
*I2C slave bus speed options.*
- enum [i2c\\_slave\\_transfer\\_event\\_t](#) {

```

kI2C_SlaveAddressMatchEvent = 0x01U,
kI2C_SlaveTransmitEvent = 0x02U,
kI2C_SlaveReceiveEvent = 0x04U,
kI2C_SlaveCompletionEvent = 0x20U,
kI2C_SlaveDeselectedEvent,
kI2C_SlaveAllEvents }

```

*Set of events sent to the callback for non blocking slave transfers.*

- enum `i2c_slave_fsm_t`

*I2C slave software finite state machine states.*

## Slave initialization and deinitialization

- void `I2C_SlaveGetDefaultConfig` (`i2c_slave_config_t` \*slaveConfig)  
*Provides a default configuration for the I2C slave peripheral.*
- `status_t I2C_SlaveInit` (`I2C_Type` \*base, const `i2c_slave_config_t` \*slaveConfig, `uint32_t` srcClock-  
\_Hz)  
*Initializes the I2C slave peripheral.*
- void `I2C_SlaveSetAddress` (`I2C_Type` \*base, `i2c_slave_address_register_t` addressRegister, `uint8_t`  
address, bool addressDisable)  
*Configures Slave Address n register.*
- void `I2C_SlaveDeinit` (`I2C_Type` \*base)  
*Deinitializes the I2C slave peripheral.*
- static void `I2C_SlaveEnable` (`I2C_Type` \*base, bool enable)  
*Enables or disables the I2C module as slave.*

## Slave status

- static void `I2C_SlaveClearStatusFlags` (`I2C_Type` \*base, `uint32_t` statusMask)  
*Clears the I2C status flag state.*

## Slave bus operations

- `status_t I2C_SlaveWriteBlocking` (`I2C_Type` \*base, const `uint8_t` \*txBuff, `size_t` txSize)  
*Performs a polling send transfer on the I2C bus.*
- `status_t I2C_SlaveReadBlocking` (`I2C_Type` \*base, `uint8_t` \*rxBuff, `size_t` rxSize)  
*Performs a polling receive transfer on the I2C bus.*

## Slave non-blocking

- void `I2C_SlaveTransferCreateHandle` (`I2C_Type` \*base, `i2c_slave_handle_t` \*handle, `i2c_slave_-`  
`transfer_callback_t` callback, void \*userData)  
*Creates a new handle for the I2C slave non-blocking APIs.*
- `status_t I2C_SlaveTransferNonBlocking` (`I2C_Type` \*base, `i2c_slave_handle_t` \*handle, `uint32_t`  
eventMask)

## I2C Slave Driver

- Starts accepting slave transfers.*
- [status\\_t I2C\\_SlaveSetSendBuffer](#) (I2C\_Type \*base, volatile [i2c\\_slave\\_transfer\\_t](#) \*transfer, const void \*txData, size\_t txSize, uint32\_t eventMask)  
*Starts accepting master read from slave requests.*
- [status\\_t I2C\\_SlaveSetReceiveBuffer](#) (I2C\_Type \*base, volatile [i2c\\_slave\\_transfer\\_t](#) \*transfer, void \*rxData, size\_t rxSize, uint32\_t eventMask)  
*Starts accepting master write to slave requests.*
- static uint32\_t [I2C\\_SlaveGetReceivedAddress](#) (I2C\_Type \*base, volatile [i2c\\_slave\\_transfer\\_t](#) \*transfer)  
*Returns the slave address sent by the I2C master.*
- void [I2C\\_SlaveTransferAbort](#) (I2C\_Type \*base, [i2c\\_slave\\_handle\\_t](#) \*handle)  
*Aborts the slave non-blocking transfers.*
- [status\\_t I2C\\_SlaveTransferGetCount](#) (I2C\_Type \*base, [i2c\\_slave\\_handle\\_t](#) \*handle, size\_t \*count)  
*Gets the slave transfer remaining bytes during a interrupt non-blocking transfer.*

## Slave IRQ handler

- void [I2C\\_SlaveTransferHandleIRQ](#) (I2C\_Type \*base, [i2c\\_slave\\_handle\\_t](#) \*handle)  
*Reusable routine to handle slave interrupts.*

## 15.5.2 Data Structure Documentation

### 15.5.2.1 struct i2c\_slave\_address\_t

#### Data Fields

- uint8\_t [address](#)  
*7-bit Slave address SLVADR.*
- bool [addressDisable](#)  
*Slave address disable SADISABLE.*

#### 15.5.2.1.0.14 Field Documentation

##### 15.5.2.1.0.14.1 uint8\_t i2c\_slave\_address\_t::address

##### 15.5.2.1.0.14.2 bool i2c\_slave\_address\_t::addressDisable

### 15.5.2.2 struct i2c\_slave\_config\_t

This structure holds configuration settings for the I2C slave peripheral. To initialize this structure to reasonable defaults, call the [I2C\\_SlaveGetDefaultConfig\(\)](#) function and pass a pointer to your configuration structure instance.

The configuration structure can be made constant so it resides in flash.



## Data Fields

- [i2c\\_slave\\_address\\_t address0](#)  
*Slave's 7-bit address and disable.*
- [i2c\\_slave\\_address\\_t address1](#)  
*Alternate slave 7-bit address and disable.*
- [i2c\\_slave\\_address\\_t address2](#)  
*Alternate slave 7-bit address and disable.*
- [i2c\\_slave\\_address\\_t address3](#)  
*Alternate slave 7-bit address and disable.*
- [i2c\\_slave\\_address\\_qual\\_mode\\_t qualMode](#)  
*Qualify mode for slave address 0.*
- [uint8\\_t qualAddress](#)  
*Slave address qualifier for address 0.*
- [i2c\\_slave\\_bus\\_speed\\_t busSpeed](#)  
*Slave bus speed mode.*
- [bool enableSlave](#)  
*Enable slave mode.*

### 15.5.2.2.0.15 Field Documentation

**15.5.2.2.0.15.1** [i2c\\_slave\\_address\\_t i2c\\_slave\\_config\\_t::address0](#)

**15.5.2.2.0.15.2** [i2c\\_slave\\_address\\_t i2c\\_slave\\_config\\_t::address1](#)

**15.5.2.2.0.15.3** [i2c\\_slave\\_address\\_t i2c\\_slave\\_config\\_t::address2](#)

**15.5.2.2.0.15.4** [i2c\\_slave\\_address\\_t i2c\\_slave\\_config\\_t::address3](#)

**15.5.2.2.0.15.5** [i2c\\_slave\\_address\\_qual\\_mode\\_t i2c\\_slave\\_config\\_t::qualMode](#)

**15.5.2.2.0.15.6** [uint8\\_t i2c\\_slave\\_config\\_t::qualAddress](#)

**15.5.2.2.0.15.7** [i2c\\_slave\\_bus\\_speed\\_t i2c\\_slave\\_config\\_t::busSpeed](#)

If the slave function stretches SCL to allow for software response, it must provide sufficient data setup time to the master before releasing the stretched clock. This is accomplished by inserting one clock time of CLKDIV at that point. The [busSpeed](#) value is used to configure CLKDIV such that one clock time is greater than the tSU;DAT value noted in the I2C bus specification for the I2C mode that is being used. If the [busSpeed](#) mode is unknown at compile time, use the longest data setup time `kI2C_SlaveStandardMode` (250 ns)

**15.5.2.2.0.15.8** [bool i2c\\_slave\\_config\\_t::enableSlave](#)

### 15.5.2.3 struct [i2c\\_slave\\_transfer\\_t](#)

## Data Fields

- [i2c\\_slave\\_handle\\_t \\* handle](#)  
*Pointer to handle that contains this transfer.*
- [i2c\\_slave\\_transfer\\_event\\_t event](#)

## I2C Slave Driver

- *Reason the callback is being invoked.*
- uint8\_t [receivedAddress](#)  
*Matching address send by master.*
- uint32\_t [eventMask](#)  
*Mask of enabled events.*
- uint8\_t \* [rxData](#)  
*Transfer buffer for receive data.*
- const uint8\_t \* [txData](#)  
*Transfer buffer for transmit data.*
- size\_t [txSize](#)  
*Transfer size.*
- size\_t [rxSize](#)  
*Transfer size.*
- size\_t [transferredCount](#)  
*Number of bytes transferred during this transfer.*
- [status\\_t](#) [completionStatus](#)  
*Success or error code describing how the transfer completed.*

### 15.5.2.3.0.16 Field Documentation

**15.5.2.3.0.16.1 i2c\_slave\_handle\_t\* i2c\_slave\_transfer\_t::handle**

**15.5.2.3.0.16.2 i2c\_slave\_transfer\_event\_t i2c\_slave\_transfer\_t::event**

**15.5.2.3.0.16.3 uint8\_t i2c\_slave\_transfer\_t::receivedAddress**

7-bits plus R/nW bit0

**15.5.2.3.0.16.4 uint32\_t i2c\_slave\_transfer\_t::eventMask**

**15.5.2.3.0.16.5 size\_t i2c\_slave\_transfer\_t::transferredCount**

**15.5.2.3.0.16.6 status\_t i2c\_slave\_transfer\_t::completionStatus**

Only applies for [kI2C\\_SlaveCompletionEvent](#).

### 15.5.2.4 struct i2c\_slave\_handle

I2C slave handle typedef.

Note

The contents of this structure are private and subject to change.

### Data Fields

- volatile [i2c\\_slave\\_transfer\\_t](#) [transfer](#)  
*I2C slave transfer.*
- volatile bool [isBusy](#)

- *Whether transfer is busy.*  
volatile `i2c_slave_fsm_t` `slaveFsm`  
*slave transfer state machine.*
- `i2c_slave_transfer_callback_t` `callback`  
*Callback function called at transfer event.*
- void \* `userData`  
*Callback parameter passed to callback.*

#### 15.5.2.4.0.17 Field Documentation

15.5.2.4.0.17.1 volatile `i2c_slave_transfer_t` `i2c_slave_handle_t::transfer`

15.5.2.4.0.17.2 volatile bool `i2c_slave_handle_t::isBusy`

15.5.2.4.0.17.3 volatile `i2c_slave_fsm_t` `i2c_slave_handle_t::slaveFsm`

15.5.2.4.0.17.4 `i2c_slave_transfer_callback_t` `i2c_slave_handle_t::callback`

15.5.2.4.0.17.5 void\* `i2c_slave_handle_t::userData`

### 15.5.3 Typedef Documentation

15.5.3.1 typedef void(\* `i2c_slave_transfer_callback_t`)(`I2C_Type` \*`base`, volatile `i2c_slave_transfer_t` \*`transfer`, void \*`userData`)

This callback is used only for the slave non-blocking transfer API. To install a callback, use the `I2C_SlaveSetCallback()` function after you have created a handle.

Parameters

|                 |                                                                                      |
|-----------------|--------------------------------------------------------------------------------------|
| <i>base</i>     | Base address for the I2C instance on which the event occurred.                       |
| <i>transfer</i> | Pointer to transfer descriptor containing values passed to and/or from the callback. |
| <i>userData</i> | Arbitrary pointer-sized value passed from the application.                           |

### 15.5.4 Enumeration Type Documentation

15.5.4.1 enum `_i2c_slave_flags`

Note

These enums are meant to be OR'd together to form a bit mask.

Enumerator

***kI2C\_SlavePendingFlag*** The I2C module is waiting for software interaction.

***kI2C\_SlaveNotStretching*** Indicates whether the slave is currently stretching clock (0 = yes, 1 = no).

## I2C Slave Driver

***kI2C\_SlaveSelected*** Indicates whether the slave is selected by an address match.

***kI2C\_SaveDeselected*** Indicates that slave was previously deselected (deselect event took place, w1c).

### 15.5.4.2 enum i2c\_slave\_address\_register\_t

Enumerator

***kI2C\_SlaveAddressRegister0*** Slave Address 0 register.

***kI2C\_SlaveAddressRegister1*** Slave Address 1 register.

***kI2C\_SlaveAddressRegister2*** Slave Address 2 register.

***kI2C\_SlaveAddressRegister3*** Slave Address 3 register.

### 15.5.4.3 enum i2c\_slave\_address\_qual\_mode\_t

Enumerator

***kI2C\_QualModeMask*** The SLVQUAL0 field (qualAddress) is used as a logical mask for matching address0.

***kI2C\_QualModeExtend*** The SLVQUAL0 (qualAddress) field is used to extend address 0 matching in a range of addresses.

### 15.5.4.4 enum i2c\_slave\_bus\_speed\_t

### 15.5.4.5 enum i2c\_slave\_transfer\_event\_t

These event enumerations are used for two related purposes. First, a bit mask created by OR'ing together events is passed to [I2C\\_SlaveTransferNonBlocking\(\)](#) in order to specify which events to enable. Then, when the slave callback is invoked, it is passed the current event through its *transfer* parameter.

Note

These enumerations are meant to be OR'd together to form a bit mask of events.

Enumerator

***kI2C\_SlaveAddressMatchEvent*** Received the slave address after a start or repeated start.

***kI2C\_SlaveTransmitEvent*** Callback is requested to provide data to transmit (slave-transmitter role).

***kI2C\_SlaveReceiveEvent*** Callback is requested to provide a buffer in which to place received data (slave-receiver role).

***kI2C\_SlaveCompletionEvent*** All data in the active transfer have been consumed.

***kI2C\_SlaveDeselectedEvent*** The slave function has become deselected (SLVSEL flag changing from 1 to 0).

***kI2C\_SlaveAllEvents*** Bit mask of all available events.

## 15.5.5 Function Documentation

### 15.5.5.1 void I2C\_SlaveGetDefaultConfig ( i2c\_slave\_config\_t \* *slaveConfig* )

This function provides the following default configuration for the I2C slave peripheral:

```
* slaveConfig->enableSlave = true;
* slaveConfig->address0.disable = false;
* slaveConfig->address0.address = 0u;
* slaveConfig->address1.disable = true;
* slaveConfig->address2.disable = true;
* slaveConfig->address3.disable = true;
* slaveConfig->busSpeed = kI2C_SlaveStandardMode;
*
```

After calling this function, override any settings to customize the configuration, prior to initializing the master driver with [I2C\\_SlaveInit\(\)](#). Be sure to override at least the *address0.address* member of the configuration structure with the desired slave address.

Parameters

|     |                    |                                                                                                                    |
|-----|--------------------|--------------------------------------------------------------------------------------------------------------------|
| out | <i>slaveConfig</i> | User provided configuration structure that is set to default values. Refer to <a href="#">i2c_slave_config_t</a> . |
|-----|--------------------|--------------------------------------------------------------------------------------------------------------------|

### 15.5.5.2 status\_t I2C\_SlaveInit ( I2C\_Type \* *base*, const i2c\_slave\_config\_t \* *slaveConfig*, uint32\_t *srcClock\_Hz* )

This function enables the peripheral clock and initializes the I2C slave peripheral as described by the user provided configuration.

Parameters

|                    |                                                                                                                                                             |
|--------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>        | The I2C peripheral base address.                                                                                                                            |
| <i>slaveConfig</i> | User provided peripheral configuration. Use <a href="#">I2C_SlaveGetDefaultConfig()</a> to get a set of defaults that you can override.                     |
| <i>srcClock_Hz</i> | Frequency in Hertz of the I2C functional clock. Used to calculate CLKDIV value to provide enough data setup time for master when slave stretches the clock. |

### 15.5.5.3 void I2C\_SlaveSetAddress ( I2C\_Type \* *base*, i2c\_slave\_address\_register\_t *addressRegister*, uint8\_t *address*, bool *addressDisable* )

This function writes new value to Slave Address register.

## I2C Slave Driver

### Parameters

|                         |                                                                                                      |
|-------------------------|------------------------------------------------------------------------------------------------------|
| <i>base</i>             | The I2C peripheral base address.                                                                     |
| <i>address-Register</i> | The module supports multiple address registers. The parameter determines which one shall be changed. |
| <i>address</i>          | The slave address to be stored to the address register for matching.                                 |
| <i>addressDisable</i>   | Disable matching of the specified address register.                                                  |

#### 15.5.5.4 void I2C\_SlaveDeinit ( I2C\_Type \* *base* )

This function disables the I2C slave peripheral and gates the clock. It also performs a software reset to restore the peripheral to reset conditions.

### Parameters

|             |                                  |
|-------------|----------------------------------|
| <i>base</i> | The I2C peripheral base address. |
|-------------|----------------------------------|

#### 15.5.5.5 static void I2C\_SlaveEnable ( I2C\_Type \* *base*, bool *enable* ) [inline], [static]

### Parameters

|               |                                     |
|---------------|-------------------------------------|
| <i>base</i>   | The I2C peripheral base address.    |
| <i>enable</i> | True to enable or false to disable. |

#### 15.5.5.6 static void I2C\_SlaveClearStatusFlags ( I2C\_Type \* *base*, uint32\_t *statusMask* ) [inline], [static]

The following status register flags can be cleared:

- slave deselected flag

Attempts to clear other flags has no effect.

### Parameters

|                   |                                                                                                                                                                                                                              |
|-------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>       | The I2C peripheral base address.                                                                                                                                                                                             |
| <i>statusMask</i> | A bitmask of status flags that are to be cleared. The mask is composed of <a href="#">_i2c_slave_flags</a> enumerators OR'd together. You may pass the result of a previous call to <code>I2C_SlaveGetStatusFlags()</code> . |

See Also

[\\_i2c\\_slave\\_flags](#).

#### 15.5.5.7 `status_t I2C_SlaveWriteBlocking ( I2C_Type * base, const uint8_t * txBuff, size_t txSize )`

The function executes blocking address phase and blocking data phase.

Parameters

|               |                                                    |
|---------------|----------------------------------------------------|
| <i>base</i>   | The I2C peripheral base address.                   |
| <i>txBuff</i> | The pointer to the data to be transferred.         |
| <i>txSize</i> | The length in bytes of the data to be transferred. |

Returns

`kStatus_Success` Data has been sent.

`kStatus_Fail` Unexpected slave state (master data write while master read from slave is expected).

#### 15.5.5.8 `status_t I2C_SlaveReadBlocking ( I2C_Type * base, uint8_t * rxBuff, size_t rxSize )`

The function executes blocking address phase and blocking data phase.

Parameters

|               |                                                    |
|---------------|----------------------------------------------------|
| <i>base</i>   | The I2C peripheral base address.                   |
| <i>rxBuff</i> | The pointer to the data to be transferred.         |
| <i>rxSize</i> | The length in bytes of the data to be transferred. |

Returns

`kStatus_Success` Data has been received.

`kStatus_Fail` Unexpected slave state (master data read while master write to slave is expected).

## I2C Slave Driver

### 15.5.5.9 void I2C\_SlaveTransferCreateHandle ( I2C\_Type \* *base*, i2c\_slave\_handle\_t \* *handle*, i2c\_slave\_transfer\_callback\_t *callback*, void \* *userData* )

The creation of a handle is for use with the non-blocking APIs. Once a handle is created, there is not a corresponding destroy handle. If the user wants to terminate a transfer, the [I2C\\_SlaveTransferAbort\(\)](#) API shall be called.

#### Parameters

|     |                 |                                                              |
|-----|-----------------|--------------------------------------------------------------|
|     | <i>base</i>     | The I2C peripheral base address.                             |
| out | <i>handle</i>   | Pointer to the I2C slave driver handle.                      |
|     | <i>callback</i> | User provided pointer to the asynchronous callback function. |
|     | <i>userData</i> | User provided pointer to the application callback data.      |

### 15.5.5.10 status\_t I2C\_SlaveTransferNonBlocking ( I2C\_Type \* *base*, i2c\_slave\_handle\_t \* *handle*, uint32\_t *eventMask* )

Call this API after calling [I2C\\_SlaveInit\(\)](#) and [I2C\\_SlaveTransferCreateHandle\(\)](#) to start processing transactions driven by an I2C master. The slave monitors the I2C bus and pass events to the callback that was passed into the call to [I2C\\_SlaveTransferCreateHandle\(\)](#). The callback is always invoked from the interrupt context.

If no slave Tx transfer is busy, a master read from slave request invokes [kI2C\\_SlaveTransmitEvent](#) callback. If no slave Rx transfer is busy, a master write to slave request invokes [kI2C\\_SlaveReceiveEvent](#) callback.

The set of events received by the callback is customizable. To do so, set the *eventMask* parameter to the OR'd combination of [i2c\\_slave\\_transfer\\_event\\_t](#) enumerators for the events you wish to receive. The [kI2C\\_SlaveTransmitEvent](#) and [kI2C\\_SlaveReceiveEvent](#) events are always enabled and do not need to be included in the mask. Alternatively, you can pass 0 to get a default set of only the transmit and receive events that are always enabled. In addition, the [kI2C\\_SlaveAllEvents](#) constant is provided as a convenient way to enable all events.

#### Parameters

|                  |                                                                                                                                                                                                                                                                                                    |
|------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>      | The I2C peripheral base address.                                                                                                                                                                                                                                                                   |
| <i>handle</i>    | Pointer to <a href="#">i2c_slave_handle_t</a> structure which stores the transfer state.                                                                                                                                                                                                           |
| <i>eventMask</i> | Bit mask formed by OR'ing together <a href="#">i2c_slave_transfer_event_t</a> enumerators to specify which events to send to the callback. Other accepted values are 0 to get a default set of only the transmit and receive events, and <a href="#">kI2C_SlaveAllEvents</a> to enable all events. |



## Return values

|                         |                                                           |
|-------------------------|-----------------------------------------------------------|
| <i>kStatus_Success</i>  | Slave transfers were successfully started.                |
| <i>kStatus_I2C_Busy</i> | Slave transfers have already been started on this handle. |

#### 15.5.5.11 **status\_t I2C\_SlaveSetSendBuffer ( I2C\_Type \* *base*, volatile i2c\_slave\_transfer\_t \* *transfer*, const void \* *txData*, size\_t *txSize*, uint32\_t *eventMask* )**

The function can be called in response to [kI2C\\_SlaveTransmitEvent](#) callback to start a new slave Tx transfer from within the transfer callback.

The set of events received by the callback is customizable. To do so, set the *eventMask* parameter to the OR'd combination of [i2c\\_slave\\_transfer\\_event\\_t](#) enumerators for the events you wish to receive. The [kI2C\\_SlaveTransmitEvent](#) and [kI2C\\_SlaveReceiveEvent](#) events are always enabled and do not need to be included in the mask. Alternatively, you can pass 0 to get a default set of only the transmit and receive events that are always enabled. In addition, the [kI2C\\_SlaveAllEvents](#) constant is provided as a convenient way to enable all events.

## Parameters

|                  |                                                                                                                                                                                                                                                                                                    |
|------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>      | The I2C peripheral base address.                                                                                                                                                                                                                                                                   |
| <i>transfer</i>  | Pointer to <a href="#">i2c_slave_transfer_t</a> structure.                                                                                                                                                                                                                                         |
| <i>txData</i>    | Pointer to data to send to master.                                                                                                                                                                                                                                                                 |
| <i>txSize</i>    | Size of <i>txData</i> in bytes.                                                                                                                                                                                                                                                                    |
| <i>eventMask</i> | Bit mask formed by OR'ing together <a href="#">i2c_slave_transfer_event_t</a> enumerators to specify which events to send to the callback. Other accepted values are 0 to get a default set of only the transmit and receive events, and <a href="#">kI2C_SlaveAllEvents</a> to enable all events. |

## Return values

|                         |                                                           |
|-------------------------|-----------------------------------------------------------|
| <i>kStatus_Success</i>  | Slave transfers were successfully started.                |
| <i>kStatus_I2C_Busy</i> | Slave transfers have already been started on this handle. |

#### 15.5.5.12 **status\_t I2C\_SlaveSetReceiveBuffer ( I2C\_Type \* *base*, volatile i2c\_slave\_transfer\_t \* *transfer*, void \* *rxData*, size\_t *rxSize*, uint32\_t *eventMask* )**

The function can be called in response to [kI2C\\_SlaveReceiveEvent](#) callback to start a new slave Rx transfer from within the transfer callback.

## I2C Slave Driver

The set of events received by the callback is customizable. To do so, set the *eventMask* parameter to the OR'd combination of *i2c\_slave\_transfer\_event\_t* enumerators for the events you wish to receive. The *kI2C\_SlaveTransmitEvent* and *kI2C\_SlaveReceiveEvent* events are always enabled and do not need to be included in the mask. Alternatively, you can pass 0 to get a default set of only the transmit and receive events that are always enabled. In addition, the *kI2C\_SlaveAllEvents* constant is provided as a convenient way to enable all events.

### Parameters

|                  |                                                                                                                                                                                                                                                                                  |
|------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>      | The I2C peripheral base address.                                                                                                                                                                                                                                                 |
| <i>transfer</i>  | Pointer to <i>i2c_slave_transfer_t</i> structure.                                                                                                                                                                                                                                |
| <i>rxData</i>    | Pointer to data to store data from master.                                                                                                                                                                                                                                       |
| <i>rxSize</i>    | Size of <i>rxData</i> in bytes.                                                                                                                                                                                                                                                  |
| <i>eventMask</i> | Bit mask formed by OR'ing together <i>i2c_slave_transfer_event_t</i> enumerators to specify which events to send to the callback. Other accepted values are 0 to get a default set of only the transmit and receive events, and <i>kI2C_SlaveAllEvents</i> to enable all events. |

### Return values

|                         |                                                           |
|-------------------------|-----------------------------------------------------------|
| <i>kStatus_Success</i>  | Slave transfers were successfully started.                |
| <i>kStatus_I2C_Busy</i> | Slave transfers have already been started on this handle. |

#### 15.5.5.13 **static uint32\_t I2C\_SlaveGetReceivedAddress ( I2C\_Type \* *base*, volatile i2c\_slave\_transfer\_t \* *transfer* ) [inline], [static]**

This function should only be called from the address match event callback *kI2C\_SlaveAddressMatchEvent*.

### Parameters

|                 |                                  |
|-----------------|----------------------------------|
| <i>base</i>     | The I2C peripheral base address. |
| <i>transfer</i> | The I2C slave transfer.          |

### Returns

The 8-bit address matched by the I2C slave. Bit 0 contains the R/w direction bit, and the 7-bit slave address is in the upper 7 bits.

#### 15.5.5.14 **void I2C\_SlaveTransferAbort ( I2C\_Type \* *base*, i2c\_slave\_handle\_t \* *handle* )**

## Note

This API could be called at any time to stop slave for handling the bus events.

## Parameters

|               |                                                                          |
|---------------|--------------------------------------------------------------------------|
| <i>base</i>   | The I2C peripheral base address.                                         |
| <i>handle</i> | Pointer to i2c_slave_handle_t structure which stores the transfer state. |

## Return values

|                         |  |
|-------------------------|--|
| <i>kStatus_Success</i>  |  |
| <i>kStatus_I2C_Idle</i> |  |

#### 15.5.5.15 status\_t I2C\_SlaveTransferGetCount ( I2C\_Type \* *base*, i2c\_slave\_handle\_t \* *handle*, size\_t \* *count* )

## Parameters

|               |                                                                     |
|---------------|---------------------------------------------------------------------|
| <i>base</i>   | I2C base pointer.                                                   |
| <i>handle</i> | pointer to i2c_slave_handle_t structure.                            |
| <i>count</i>  | Number of bytes transferred so far by the non-blocking transaction. |

## Return values

|                                |                                |
|--------------------------------|--------------------------------|
| <i>kStatus_InvalidArgument</i> | count is Invalid.              |
| <i>kStatus_Success</i>         | Successfully return the count. |

#### 15.5.5.16 void I2C\_SlaveTransferHandleIRQ ( I2C\_Type \* *base*, i2c\_slave\_handle\_t \* *handle* )

## Note

This function does not need to be called unless you are reimplementing the non blocking API's interrupt handler routines to add special functionality.

## I2C Slave Driver

### Parameters

|               |                                                                          |
|---------------|--------------------------------------------------------------------------|
| <i>base</i>   | The I2C peripheral base address.                                         |
| <i>handle</i> | Pointer to i2c_slave_handle_t structure which stores the transfer state. |

## 15.6 I2C DMA Driver

### 15.6.1 Overview

#### Data Structures

- struct [i2c\\_master\\_dma\\_handle\\_t](#)  
*I2C master dma transfer structure. [More...](#)*

#### Macros

- #define [I2C\\_MAX\\_DMA\\_TRANSFER\\_COUNT](#) 1024  
*Maximum length of single DMA transfer (determined by capability of the DMA engine)*

#### Typedefs

- typedef void(\* [i2c\\_master\\_dma\\_transfer\\_callback\\_t](#) )(I2C\_Type \*base, i2c\_master\_dma\_handle\_t \*handle, [status\\_t](#) status, void \*userData)  
*I2C master dma transfer callback typedef.*

#### Driver version

- #define [FSL\\_I2C\\_DMA\\_DRIVER\\_VERSION](#) ([MAKE\\_VERSION](#)(2, 0, 5))  
*I2C DMA driver version 2.0.5.*

### I2C Block DMA Transfer Operation

- void [I2C\\_MasterTransferCreateHandleDMA](#) (I2C\_Type \*base, i2c\_master\_dma\_handle\_t \*handle, [i2c\\_master\\_dma\\_transfer\\_callback\\_t](#) callback, void \*userData, [dma\\_handle\\_t](#) \*dmaHandle)  
*Init the I2C handle which is used in transactional functions.*
- [status\\_t](#) [I2C\\_MasterTransferDMA](#) (I2C\_Type \*base, i2c\_master\_dma\_handle\_t \*handle, i2c\_master\_transfer\_t \*xfer)  
*Performs a master dma non-blocking transfer on the I2C bus.*
- [status\\_t](#) [I2C\\_MasterTransferGetCountDMA](#) (I2C\_Type \*base, i2c\_master\_dma\_handle\_t \*handle, size\_t \*count)  
*Get master transfer status during a dma non-blocking transfer.*
- void [I2C\\_MasterTransferAbortDMA](#) (I2C\_Type \*base, i2c\_master\_dma\_handle\_t \*handle)  
*Abort a master dma non-blocking transfer in a early time.*

### 15.6.2 Data Structure Documentation

#### 15.6.2.1 struct `_i2c_master_dma_handle`

I2C master dma handle typedef.

##### Data Fields

- `uint8_t state`  
*Transfer state machine current state.*
- `uint32_t transferCount`  
*Indicates progress of the transfer.*
- `uint32_t remainingBytesDMA`  
*Remaining byte count to be transferred using DMA.*
- `uint8_t * buf`  
*Buffer pointer for current state.*
- `dma_handle_t * dmaHandle`  
*The DMA handler used.*
- `i2c_master_transfer_t transfer`  
*Copy of the current transfer info.*
- `i2c_master_dma_transfer_callback_t completionCallback`  
*Callback function called after dma transfer finished.*
- `void * userData`  
*Callback parameter passed to callback function.*

### 15.6.2.1.0.18 Field Documentation

15.6.2.1.0.18.1 `uint8_t i2c_master_dma_handle_t::state`

15.6.2.1.0.18.2 `uint32_t i2c_master_dma_handle_t::remainingBytesDMA`

15.6.2.1.0.18.3 `uint8_t* i2c_master_dma_handle_t::buf`

15.6.2.1.0.18.4 `dma_handle_t* i2c_master_dma_handle_t::dmaHandle`

15.6.2.1.0.18.5 `i2c_master_transfer_t i2c_master_dma_handle_t::transfer`

15.6.2.1.0.18.6 `i2c_master_dma_transfer_callback_t i2c_master_dma_handle_t::completion-Callback`

15.6.2.1.0.18.7 `void* i2c_master_dma_handle_t::userData`

### 15.6.3 Macro Definition Documentation

15.6.3.1 `#define FSL_I2C_DMA_DRIVER_VERSION (MAKE_VERSION(2, 0, 5))`

### 15.6.4 Typedef Documentation

15.6.4.1 `typedef void(* i2c_master_dma_transfer_callback_t)(I2C_Type *base, i2c_master_dma_handle_t *handle, status_t status, void *userData)`

### 15.6.5 Function Documentation

15.6.5.1 `void I2C_MasterTransferCreateHandleDMA ( I2C_Type * base, i2c_master_dma_handle_t * handle, i2c_master_dma_transfer_callback_t callback, void * userData, dma_handle_t * dmaHandle )`

## I2C DMA Driver

### Parameters

|                  |                                              |
|------------------|----------------------------------------------|
| <i>base</i>      | I2C peripheral base address                  |
| <i>handle</i>    | pointer to i2c_master_dma_handle_t structure |
| <i>callback</i>  | pointer to user callback function            |
| <i>userData</i>  | user param passed to the callback function   |
| <i>dmaHandle</i> | DMA handle pointer                           |

### 15.6.5.2 status\_t I2C\_MasterTransferDMA ( I2C\_Type \* *base*, i2c\_master\_dma\_handle\_t \* *handle*, i2c\_master\_transfer\_t \* *xfer* )

### Parameters

|               |                                                        |
|---------------|--------------------------------------------------------|
| <i>base</i>   | I2C peripheral base address                            |
| <i>handle</i> | pointer to i2c_master_dma_handle_t structure           |
| <i>xfer</i>   | pointer to transfer structure of i2c_master_transfer_t |

### Return values

|                                     |                                              |
|-------------------------------------|----------------------------------------------|
| <i>kStatus_Success</i>              | Successfully complete the data transmission. |
| <i>kStatus_I2C_Busy</i>             | Previous transmission still not finished.    |
| <i>kStatus_I2C_Timeout</i>          | Transfer error, wait signal timeout.         |
| <i>kStatus_I2C_Arbitration-Lost</i> | Transfer error, arbitration lost.            |
| <i>kStatus_I2C_Nak</i>              | Transfer error, receive Nak during transfer. |

### 15.6.5.3 status\_t I2C\_MasterTransferGetCountDMA ( I2C\_Type \* *base*, i2c\_master\_dma\_handle\_t \* *handle*, size\_t \* *count* )

### Parameters

|               |                                              |
|---------------|----------------------------------------------|
| <i>base</i>   | I2C peripheral base address                  |
| <i>handle</i> | pointer to i2c_master_dma_handle_t structure |



|              |                                                                     |
|--------------|---------------------------------------------------------------------|
| <i>count</i> | Number of bytes transferred so far by the non-blocking transaction. |
|--------------|---------------------------------------------------------------------|

**15.6.5.4** void I2C\_MasterTransferAbortDMA ( I2C\_Type \* *base*, i2c\_master\_dma\_handle\_t \* *handle* )

Parameters

|               |                                              |
|---------------|----------------------------------------------|
| <i>base</i>   | I2C peripheral base address                  |
| <i>handle</i> | pointer to i2c_master_dma_handle_t structure |

## I2C FreeRTOS Driver

### 15.7 I2C FreeRTOS Driver

#### 15.7.1 Overview

##### Data Structures

- struct [i2c\\_rtos\\_handle\\_t](#)  
*I2C FreeRTOS handle. [More...](#)*

##### Driver version

- #define [FSL\\_I2C\\_FREERTOS\\_DRIVER\\_VERSION](#) ([MAKE\\_VERSION](#)(2, 0, 5))  
*I2C freertos driver version 2.0.5.*

##### I2C RTOS Operation

- [status\\_t I2C\\_RTOS\\_Init](#) ([i2c\\_rtos\\_handle\\_t](#) \*handle, [I2C\\_Type](#) \*base, const [i2c\\_master\\_config\\_t](#) \*masterConfig, [uint32\\_t](#) srcClock\_Hz)  
*Initializes I2C.*
- [status\\_t I2C\\_RTOS\\_Deinit](#) ([i2c\\_rtos\\_handle\\_t](#) \*handle)  
*Deinitializes the I2C.*
- [status\\_t I2C\\_RTOS\\_Transfer](#) ([i2c\\_rtos\\_handle\\_t](#) \*handle, [i2c\\_master\\_transfer\\_t](#) \*transfer)  
*Performs I2C transfer.*

#### 15.7.2 Data Structure Documentation

##### 15.7.2.1 struct [i2c\\_rtos\\_handle\\_t](#)

##### Data Fields

- [I2C\\_Type](#) \* [base](#)  
*I2C base address.*
- [i2c\\_master\\_handle\\_t](#) [drv\\_handle](#)  
*A handle of the underlying driver, treated as opaque by the RTOS layer.*
- [status\\_t](#) [async\\_status](#)  
*Transactional state of the underlying driver.*
- [SemaphoreHandle\\_t](#) [mutex](#)  
*A mutex to lock the handle during a transfer.*
- [SemaphoreHandle\\_t](#) [semaphore](#)  
*A semaphore to notify and unblock task when the transfer ends.*

### 15.7.3 Macro Definition Documentation

15.7.3.1 **#define FSL\_I2C\_FREERTOS\_DRIVER\_VERSION (MAKE\_VERSION(2, 0, 5))**

### 15.7.4 Function Documentation

15.7.4.1 **status\_t I2C\_RTOS\_Init ( i2c\_rtos\_handle\_t \* *handle*, I2C\_Type \* *base*, const i2c\_master\_config\_t \* *masterConfig*, uint32\_t *srcClock\_Hz* )**

This function initializes the I2C module and the related RTOS context.

## I2C FreeRTOS Driver

### Parameters

|                     |                                                                          |
|---------------------|--------------------------------------------------------------------------|
| <i>handle</i>       | The RTOS I2C handle, the pointer to an allocated space for RTOS context. |
| <i>base</i>         | The pointer base address of the I2C instance to initialize.              |
| <i>masterConfig</i> | Configuration structure to set-up I2C in master mode.                    |
| <i>srcClock_Hz</i>  | Frequency of input clock of the I2C module.                              |

### Returns

status of the operation.

#### 15.7.4.2 **status\_t I2C\_RTOS\_Deinit ( i2c\_rtos\_handle\_t \* *handle* )**

This function deinitializes the I2C module and the related RTOS context.

### Parameters

|               |                      |
|---------------|----------------------|
| <i>handle</i> | The RTOS I2C handle. |
|---------------|----------------------|

#### 15.7.4.3 **status\_t I2C\_RTOS\_Transfer ( i2c\_rtos\_handle\_t \* *handle*, i2c\_master\_transfer\_t \* *transfer* )**

This function performs an I2C transfer according to data given in the transfer structure.

### Parameters

|                 |                                               |
|-----------------|-----------------------------------------------|
| <i>handle</i>   | The RTOS I2C handle.                          |
| <i>transfer</i> | Structure specifying the transfer parameters. |

### Returns

status of the operation.

## Chapter 16

# SPI: Serial Peripheral Interface Driver

### 16.1 Overview

SPI driver includes functional APIs and transactional APIs.

Functional APIs are feature/property target low level APIs. Functional APIs can be used for SPI initialization/configuration/operation for optimization/customization purpose. Using the functional API requires the knowledge of the SPI peripheral and how to organize functional APIs to meet the application requirements. All functional API use the peripheral base address as the first parameter. SPI functional operation groups provide the functional API set.

Transactional APIs are transaction target high level APIs. Transactional APIs can be used to enable the peripheral and in the application if the code size and performance of transactional APIs satisfy the requirements. If the code size and performance are a critical requirement, see the transactional API implementation and write a custom code. All transactional APIs use the `spi_handle_t` as the first parameter. Initialize the handle by calling the [SPI\\_MasterTransferCreateHandle\(\)](#) or [SPI\\_SlaveTransferCreateHandle\(\)](#) API.

Transactional APIs support asynchronous transfer. This means that the functions [SPI\\_MasterTransferNonBlocking\(\)](#) and [SPI\\_SlaveTransferNonBlocking\(\)](#) set up the interrupt for data transfer. When the transfer completes, the upper layer is notified through a callback function with the `kStatus_SPI_Idle` status.

### 16.2 Typical use case

#### 16.2.1 SPI master transfer using an interrupt method

```
#define BUFFER_LEN (64)
spi_master_handle_t spiHandle;
spi_master_config_t masterConfig;
spi_transfer_t xfer;
volatile bool isFinished = false;

const uint8_t sendData[BUFFER_LEN] = [.....];
uint8_t receiveBuff[BUFFER_LEN];

void SPI_UserCallback(SPI_Type *base, spi_master_handle_t *handle, status_t status, void *userData)
{
 isFinished = true;
}

void main(void)
{
 //...

 SPI_MasterGetDefaultConfig(&masterConfig);

 SPI_MasterInit(SPI0, &masterConfig, srcClock_Hz);
 SPI_MasterTransferCreateHandle(SPI0, &spiHandle, SPI_UserCallback, NULL);

 // Prepare to send.
 xfer.txData = sendData;
 xfer.rxData = receiveBuff;
```

## Typical use case

```
xfer.dataSize = sizeof(sendData);

// Send out.
SPI_MasterTransferNonBlocking(SPI0, &spiHandle, &xfer);

// Wait send finished.
while (!isFinished)
{
}

// ...
}
```

### 16.2.2 SPI Send/receive using a DMA method

```
#define BUFFER_LEN (64)
spi_dma_handle_t spiHandle;
dma_handle_t g_spiTxDmaHandle;
dma_handle_t g_spiRxDmaHandle;
spi_config_t masterConfig;
spi_transfer_t xfer;
volatile bool isFinished;

uint8_t sendData[BUFFER_LEN] = ...;
uint8_t receiveBuff[BUFFER_LEN];

void SPI_UserCallback(SPI_Type *base, spi_dma_handle_t *handle, status_t status, void *userData)
{
 isFinished = true;
}

void main(void)
{
 //...

 // Initialize DMA peripheral
 DMA_Init(DMA0);

 // Initialize SPI peripheral
 SPI_MasterGetDefaultConfig(&masterConfig);
 masterConfig.sselNum = SPI_SSEL;
 SPI_MasterInit(SPI0, &masterConfig, srcClock_Hz);

 // Enable DMA channels connected to SPI0 Tx/SPI0 Rx request lines
 DMA_EnableChannel(SPI0, SPI_MASTER_TX_CHANNEL);
 DMA_EnableChannel(SPI0, SPI_MASTER_RX_CHANNEL);

 // Set DMA channels priority
 DMA_SetChannelPriority(SPI0, SPI_MASTER_TX_CHANNEL,
 kDMA_ChannelPriority3);
 DMA_SetChannelPriority(SPI0, SPI_MASTER_RX_CHANNEL,
 kDMA_ChannelPriority2);

 // Creates the DMA handle.
 DMA_CreateHandle(&masterTxHandle, SPI0, SPI_MASTER_TX_CHANNEL);
 DMA_CreateHandle(&masterRxHandle, SPI0, SPI_MASTER_RX_CHANNEL);

 // Create SPI DMA handle
 SPI_MasterTransferCreateHandleDMA(SPI0, spiHandle, SPI_UserCallback,
 NULL, &g_spiTxDmaHandle, &g_spiRxDmaHandle);

 // Prepares to send.
 xfer.txData = sendData;
 xfer.rxData = receiveBuff;
 xfer.dataSize = sizeof(sendData);
```

```
// Sends out.
SPI_MasterTransferDMA(SPI0, &spiHandle, &xfer);

// Waits for send to complete.
while (!isFinished)
{
}

// ...
}
```

## Modules

- [SPI DMA Driver](#)
- [SPI Driver](#)
- [SPI FreeRTOS driver](#)

### 16.3 SPI Driver

#### 16.3.1 Overview

This section describes the programming interface of the SPI DMA driver.

#### Files

- file [fsl\\_spi.h](#)

#### Data Structures

- struct [spi\\_delay\\_config\\_t](#)  
*SPI delay time configure structure. [More...](#)*
- struct [spi\\_master\\_config\\_t](#)  
*SPI master user configure structure. [More...](#)*
- struct [spi\\_slave\\_config\\_t](#)  
*SPI slave user configure structure. [More...](#)*
- struct [spi\\_transfer\\_t](#)  
*SPI transfer structure. [More...](#)*
- struct [spi\\_half\\_duplex\\_transfer\\_t](#)  
*SPI half-duplex(master only) transfer structure. [More...](#)*
- struct [spi\\_config\\_t](#)  
*Internal configuration structure used in 'spi' and 'spi\_dma' driver. [More...](#)*
- struct [spi\\_master\\_handle\\_t](#)  
*SPI transfer handle structure. [More...](#)*

#### Macros

- #define [SPI\\_DUMMYDATA](#) (0xFFU)  
*SPI dummy transfer data, the data is sent while txBuff is NULL.*

#### Typedefs

- typedef [spi\\_master\\_handle\\_t](#) [spi\\_slave\\_handle\\_t](#)  
*Slave handle type.*
- typedef void(\* [spi\\_master\\_callback\\_t](#))(SPI\_Type \*base, spi\_master\_handle\_t \*handle, [status\\_t](#) status, void \*userData)  
*SPI master callback for finished transmit.*
- typedef void(\* [spi\\_slave\\_callback\\_t](#))(SPI\_Type \*base, [spi\\_slave\\_handle\\_t](#) \*handle, [status\\_t](#) status, void \*userData)  
*SPI slave callback for finished transmit.*



## Enumerations

- enum `spi_xfer_option_t` {  
`kSPI_FrameDelay` = (SPI\_FIFOWR\_EOF\_MASK),  
`kSPI_FrameAssert` = (SPI\_FIFOWR\_EOT\_MASK) }  
*SPI transfer option.*
- enum `spi_shift_direction_t` {  
`kSPI_MsbFirst` = 0U,  
`kSPI_LsbFirst` = 1U }  
*SPI data shifter direction options.*
- enum `spi_clock_polarity_t` {  
`kSPI_ClockPolarityActiveHigh` = 0x0U,  
`kSPI_ClockPolarityActiveLow` }  
*SPI clock polarity configuration.*
- enum `spi_clock_phase_t` {  
`kSPI_ClockPhaseFirstEdge` = 0x0U,  
`kSPI_ClockPhaseSecondEdge` }  
*SPI clock phase configuration.*
- enum `spi_txfifo_watermark_t` {  
`kSPI_TxFifo0` = 0,  
`kSPI_TxFifo1` = 1,  
`kSPI_TxFifo2` = 2,  
`kSPI_TxFifo3` = 3,  
`kSPI_TxFifo4` = 4,  
`kSPI_TxFifo5` = 5,  
`kSPI_TxFifo6` = 6,  
`kSPI_TxFifo7` = 7 }  
*txFIFO watermark values*
- enum `spi_rxfifo_watermark_t` {  
`kSPI_RxFifo1` = 0,  
`kSPI_RxFifo2` = 1,  
`kSPI_RxFifo3` = 2,  
`kSPI_RxFifo4` = 3,  
`kSPI_RxFifo5` = 4,  
`kSPI_RxFifo6` = 5,  
`kSPI_RxFifo7` = 6,  
`kSPI_RxFifo8` = 7 }  
*rxFIFO watermark values*
- enum `spi_data_width_t` {

## SPI Driver

```
kSPI_Data4Bits = 3,
kSPI_Data5Bits = 4,
kSPI_Data6Bits = 5,
kSPI_Data7Bits = 6,
kSPI_Data8Bits = 7,
kSPI_Data9Bits = 8,
kSPI_Data10Bits = 9,
kSPI_Data11Bits = 10,
kSPI_Data12Bits = 11,
kSPI_Data13Bits = 12,
kSPI_Data14Bits = 13,
kSPI_Data15Bits = 14,
kSPI_Data16Bits = 15 }
```

*Transfer data width.*

- enum `spi_ssel_t` {  
    kSPI\_Ssel0 = 0,  
    kSPI\_Ssel1 = 1,  
    kSPI\_Ssel2 = 2,  
    kSPI\_Ssel3 = 3 }

*Slave select.*

- enum `spi_spol_t`  
    *ssel polarity*
- enum `_spi_status` {  
    kStatus\_SPI\_Busy = MAKE\_STATUS(kStatusGroup\_LPC\_SPI, 0),  
    kStatus\_SPI\_Idle = MAKE\_STATUS(kStatusGroup\_LPC\_SPI, 1),  
    kStatus\_SPI\_Error = MAKE\_STATUS(kStatusGroup\_LPC\_SPI, 2),  
    kStatus\_SPI\_BaudrateNotSupport }

*SPI transfer status.*

- enum `_spi_interrupt_enable` {  
    kSPI\_RxLvlIrq = SPI\_FIFOINTENSET\_RXLVL\_MASK,  
    kSPI\_TxLvlIrq = SPI\_FIFOINTENSET\_TXLVL\_MASK }

*SPI interrupt sources.*

- enum `_spi_statusflags` {  
    kSPI\_TxEmptyFlag = SPI\_FIFOSTAT\_TXEMPTY\_MASK,  
    kSPI\_TxNotFullFlag = SPI\_FIFOSTAT\_TXNOTFULL\_MASK,  
    kSPI\_RxNotEmptyFlag = SPI\_FIFOSTAT\_RXNOTEMPTY\_MASK,  
    kSPI\_RxFullFlag = SPI\_FIFOSTAT\_RXFULL\_MASK }

*SPI status flags.*

## Functions

- uint32\_t `SPI_GetInstance` (SPI\_Type \*base)  
    *Returns instance number for SPI peripheral base address.*

## Variables

- volatile uint8\_t [s\\_dummyData](#) []  
*Global variable for dummy data value setting.*

## Driver version

- #define [FSL\\_SPI\\_DRIVER\\_VERSION](#) ([MAKE\\_VERSION](#)(2, 0, 4))  
*SPI driver version 2.0.4.*

## Initialization and deinitialization

- void [SPI\\_MasterGetDefaultConfig](#) ([spi\\_master\\_config\\_t](#) \*config)  
*Sets the SPI master configuration structure to default values.*
- [status\\_t SPI\\_MasterInit](#) ([SPI\\_Type](#) \*base, const [spi\\_master\\_config\\_t](#) \*config, uint32\_t srcClock\_Hz)  
*Initializes the SPI with master configuration.*
- void [SPI\\_SlaveGetDefaultConfig](#) ([spi\\_slave\\_config\\_t](#) \*config)  
*Sets the SPI slave configuration structure to default values.*
- [status\\_t SPI\\_SlaveInit](#) ([SPI\\_Type](#) \*base, const [spi\\_slave\\_config\\_t](#) \*config)  
*Initializes the SPI with slave configuration.*
- void [SPI\\_Deinit](#) ([SPI\\_Type](#) \*base)  
*De-initializes the SPI.*
- static void [SPI\\_Enable](#) ([SPI\\_Type](#) \*base, bool enable)  
*Enable or disable the SPI Master or Slave.*

## Status

- static uint32\_t [SPI\\_GetStatusFlags](#) ([SPI\\_Type](#) \*base)  
*Gets the status flag.*

## Interrupts

- static void [SPI\\_EnableInterrupts](#) ([SPI\\_Type](#) \*base, uint32\_t irq)
  - Enables the interrupt for the SPI.*
- static void [SPI\\_DisableInterrupts](#) ([SPI\\_Type](#) \*base, uint32\_t irq)
  - Disables the interrupt for the SPI.*

## DMA Control

- void [SPI\\_EnableTxDMA](#) ([SPI\\_Type](#) \*base, bool enable)  
*Enables the DMA request from SPI txFIFO.*
- void [SPI\\_EnableRxDMA](#) ([SPI\\_Type](#) \*base, bool enable)  
*Enables the DMA request from SPI rxFIFO.*

### Bus Operations

- void \* [SPI\\_GetConfig](#) (SPI\_Type \*base)  
*Returns the configurations.*
- [status\\_t SPI\\_MasterSetBaud](#) (SPI\_Type \*base, uint32\_t baudrate\_Bps, uint32\_t srcClock\_Hz)  
*Sets the baud rate for SPI transfer.*
- void [SPI\\_WriteData](#) (SPI\_Type \*base, uint16\_t data, uint32\_t configFlags)  
*Writes a data into the SPI data register.*
- static uint32\_t [SPI\\_ReadData](#) (SPI\_Type \*base)  
*Gets a data from the SPI data register.*
- static void [SPI\\_SetTransferDelay](#) (SPI\_Type \*base, const [spi\\_delay\\_config\\_t](#) \*config)  
*Set delay time for transfer.*
- void [SPI\\_SetDummyData](#) (SPI\_Type \*base, uint8\_t dummyData)  
*Set up the dummy data.*

### Transactional

- [status\\_t SPI\\_MasterTransferCreateHandle](#) (SPI\_Type \*base, spi\_master\_handle\_t \*handle, [spi\\_master\\_callback\\_t](#) callback, void \*userData)  
*Initializes the SPI master handle.*
- [status\\_t SPI\\_MasterTransferBlocking](#) (SPI\_Type \*base, [spi\\_transfer\\_t](#) \*xfer)  
*Transfers a block of data using a polling method.*
- [status\\_t SPI\\_MasterTransferNonBlocking](#) (SPI\_Type \*base, spi\_master\_handle\_t \*handle, [spi\\_transfer\\_t](#) \*xfer)  
*Performs a non-blocking SPI interrupt transfer.*
- [status\\_t SPI\\_MasterHalfDuplexTransferBlocking](#) (SPI\_Type \*base, [spi\\_half\\_duplex\\_transfer\\_t](#) \*xfer)  
*Transfers a block of data using a polling method.*
- [status\\_t SPI\\_MasterHalfDuplexTransferNonBlocking](#) (SPI\_Type \*base, spi\_master\_handle\_t \*handle, [spi\\_half\\_duplex\\_transfer\\_t](#) \*xfer)  
*Performs a non-blocking SPI interrupt transfer.*
- [status\\_t SPI\\_MasterTransferGetCount](#) (SPI\_Type \*base, spi\_master\_handle\_t \*handle, size\_t \*count)  
*Gets the master transfer count.*
- void [SPI\\_MasterTransferAbort](#) (SPI\_Type \*base, spi\_master\_handle\_t \*handle)  
*SPI master aborts a transfer using an interrupt.*
- void [SPI\\_MasterTransferHandleIRQ](#) (SPI\_Type \*base, spi\_master\_handle\_t \*handle)  
*Interrupts the handler for the SPI.*
- static [status\\_t SPI\\_SlaveTransferCreateHandle](#) (SPI\_Type \*base, [spi\\_slave\\_handle\\_t](#) \*handle, [spi\\_slave\\_callback\\_t](#) callback, void \*userData)  
*Initializes the SPI slave handle.*
- static [status\\_t SPI\\_SlaveTransferNonBlocking](#) (SPI\_Type \*base, [spi\\_slave\\_handle\\_t](#) \*handle, [spi\\_transfer\\_t](#) \*xfer)  
*Performs a non-blocking SPI slave interrupt transfer.*
- static [status\\_t SPI\\_SlaveTransferGetCount](#) (SPI\_Type \*base, [spi\\_slave\\_handle\\_t](#) \*handle, size\_t \*count)  
*Gets the slave transfer count.*
- static void [SPI\\_SlaveTransferAbort](#) (SPI\_Type \*base, [spi\\_slave\\_handle\\_t](#) \*handle)  
*SPI slave aborts a transfer using an interrupt.*

- static void [SPI\\_SlaveTransferHandleIRQ](#) (SPI\_Type \*base, [spi\\_slave\\_handle\\_t](#) \*handle)  
*Interrupts a handler for the SPI slave.*

## 16.3.2 Data Structure Documentation

### 16.3.2.1 struct spi\_delay\_config\_t

Note: The DLY register controls several programmable delays related to SPI signalling, it stands for how many SPI clock time will be inserted. The maximum value of these delay time is 15.

#### Data Fields

- uint8\_t [preDelay](#)  
*Delay between SSEL assertion and the beginning of transfer.*
- uint8\_t [postDelay](#)  
*Delay between the end of transfer and SSEL deassertion.*
- uint8\_t [frameDelay](#)  
*Delay between frame to frame.*
- uint8\_t [transferDelay](#)  
*Delay between transfer to transfer.*

#### 16.3.2.1.0.19 Field Documentation

16.3.2.1.0.19.1 uint8\_t spi\_delay\_config\_t::preDelay

16.3.2.1.0.19.2 uint8\_t spi\_delay\_config\_t::postDelay

16.3.2.1.0.19.3 uint8\_t spi\_delay\_config\_t::frameDelay

16.3.2.1.0.19.4 uint8\_t spi\_delay\_config\_t::transferDelay

### 16.3.2.2 struct spi\_master\_config\_t

#### Data Fields

- bool [enableLoopback](#)  
*Enable loopback for test purpose.*
- bool [enableMaster](#)  
*Enable SPI at initialization time.*
- [spi\\_clock\\_polarity\\_t](#) polarity  
*Clock polarity.*
- [spi\\_clock\\_phase\\_t](#) phase  
*Clock phase.*
- [spi\\_shift\\_direction\\_t](#) direction  
*MSB or LSB.*
- uint32\_t [baudRate\\_Bps](#)  
*Baud Rate for SPI in Hz.*
- [spi\\_data\\_width\\_t](#) dataWidth

## SPI Driver

- `spi_ssel_t sselNum`  
*Width of the data.*
- `spi_spol_t sselPol`  
*Slave select number.*
- `spi_txfifo_watermark_t txWatermark`  
*Configure active CS polarity.*
- `spi_rxfifo_watermark_t rxWatermark`  
*txFIFO watermark*
- `spi_delay_config_t delayConfig`  
*rxFIFO watermark*
- `spi_delay_config_t delayConfig`  
*Delay configuration.*

### 16.3.2.2.0.20 Field Documentation

#### 16.3.2.2.0.20.1 `spi_delay_config_t spi_master_config_t::delayConfig`

### 16.3.2.3 `struct spi_slave_config_t`

#### Data Fields

- `bool enableSlave`  
*Enable SPI at initialization time.*
- `spi_clock_polarity_t polarity`  
*Clock polarity.*
- `spi_clock_phase_t phase`  
*Clock phase.*
- `spi_shift_direction_t direction`  
*MSB or LSB.*
- `spi_data_width_t dataWidth`  
*Width of the data.*
- `spi_spol_t sselPol`  
*Configure active CS polarity.*
- `spi_txfifo_watermark_t txWatermark`  
*txFIFO watermark*
- `spi_rxfifo_watermark_t rxWatermark`  
*rxFIFO watermark*

### 16.3.2.4 `struct spi_transfer_t`

#### Data Fields

- `uint8_t * txData`  
*Send buffer.*
- `uint8_t * rxData`  
*Receive buffer.*
- `uint32_t configFlags`  
*Additional option to control transfer, `spi_xfer_option_t`.*
- `size_t dataSize`  
*Transfer bytes.*

#### 16.3.2.4.0.21 Field Documentation

##### 16.3.2.4.0.21.1 uint32\_t spi\_transfer\_t::configFlags

#### 16.3.2.5 struct spi\_half\_duplex\_transfer\_t

##### Data Fields

- uint8\_t \* [txData](#)  
*Send buffer.*
- uint8\_t \* [rxData](#)  
*Receive buffer.*
- size\_t [txDataSize](#)  
*Transfer bytes for transmit.*
- size\_t [rxDataSize](#)  
*Transfer bytes.*
- uint32\_t [configFlags](#)  
*Transfer configuration flags, [spi\\_xfer\\_option\\_t](#).*
- bool [isPcsAssertInTransfer](#)  
*If PCS pin keep assert between transmit and receive.*
- bool [isTransmitFirst](#)  
*True for transmit first and false for receive first.*

#### 16.3.2.5.0.22 Field Documentation

##### 16.3.2.5.0.22.1 uint32\_t spi\_half\_duplex\_transfer\_t::configFlags

##### 16.3.2.5.0.22.2 bool spi\_half\_duplex\_transfer\_t::isPcsAssertInTransfer

true for assert and false for deassert.

##### 16.3.2.5.0.22.3 bool spi\_half\_duplex\_transfer\_t::isTransmitFirst

#### 16.3.2.6 struct spi\_config\_t

#### 16.3.2.7 struct \_spi\_master\_handle

Master handle type.

##### Data Fields

- uint8\_t \*volatile [txData](#)  
*Transfer buffer.*
- uint8\_t \*volatile [rxData](#)  
*Receive buffer.*
- volatile size\_t [txRemainingBytes](#)  
*Number of data to be transmitted [in bytes].*
- volatile size\_t [rxRemainingBytes](#)  
*Number of data to be received [in bytes].*
- volatile size\_t [toReceiveCount](#)

## SPI Driver

- *Receive data remaining in bytes.*  
size\_t **totalByteCount**
- *A number of transfer bytes.*  
volatile uint32\_t **state**
- *SPI internal state.*  
spi\_master\_callback\_t **callback**
- *SPI callback.*  
void \* **userData**
- *Callback parameter.*  
uint8\_t **dataWidth**
- *Width of the data [Valid values: 1 to 16].*  
uint8\_t **sselNum**
- *Slave select number to be asserted when transferring data [Valid values: 0 to 3].*  
uint32\_t **configFlags**
- *Additional option to control transfer.*  
spi\_txfifo\_watermark\_t **txWatermark**
- *txFIFO watermark*  
spi\_rxfifo\_watermark\_t **rxWatermark**
- *rxFIFO watermark*

### 16.3.3 Macro Definition Documentation

16.3.3.1 #define FSL\_SPI\_DRIVER\_VERSION (MAKE\_VERSION(2, 0, 4))

16.3.3.2 #define SPI\_DUMMYDATA (0xFFU)

### 16.3.4 Enumeration Type Documentation

16.3.4.1 enum spi\_xfer\_option\_t

Enumerator

**kSPI\_FrameDelay** A delay may be inserted, defined in the DLY register.

**kSPI\_FrameAssert** SSEL will be deasserted at the end of a transfer.

16.3.4.2 enum spi\_shift\_direction\_t

Enumerator

**kSPI\_MsbFirst** Data transfers start with most significant bit.

**kSPI\_LsbFirst** Data transfers start with least significant bit.



### 16.3.4.3 enum spi\_clock\_polarity\_t

Enumerator

***kSPI\_ClockPolarityActiveHigh*** Active-high SPI clock (idles low).

***kSPI\_ClockPolarityActiveLow*** Active-low SPI clock (idles high).

### 16.3.4.4 enum spi\_clock\_phase\_t

Enumerator

***kSPI\_ClockPhaseFirstEdge*** First edge on SCK occurs at the middle of the first cycle of a data transfer.

***kSPI\_ClockPhaseSecondEdge*** First edge on SCK occurs at the start of the first cycle of a data transfer.

### 16.3.4.5 enum spi\_txfifo\_watermark\_t

Enumerator

***kSPI\_TxFifo0*** SPI tx watermark is empty.

***kSPI\_TxFifo1*** SPI tx watermark at 1 item.

***kSPI\_TxFifo2*** SPI tx watermark at 2 items.

***kSPI\_TxFifo3*** SPI tx watermark at 3 items.

***kSPI\_TxFifo4*** SPI tx watermark at 4 items.

***kSPI\_TxFifo5*** SPI tx watermark at 5 items.

***kSPI\_TxFifo6*** SPI tx watermark at 6 items.

***kSPI\_TxFifo7*** SPI tx watermark at 7 items.

### 16.3.4.6 enum spi\_rxfifo\_watermark\_t

Enumerator

***kSPI\_RxFifo1*** SPI rx watermark at 1 item.

***kSPI\_RxFifo2*** SPI rx watermark at 2 items.

***kSPI\_RxFifo3*** SPI rx watermark at 3 items.

***kSPI\_RxFifo4*** SPI rx watermark at 4 items.

***kSPI\_RxFifo5*** SPI rx watermark at 5 items.

***kSPI\_RxFifo6*** SPI rx watermark at 6 items.

***kSPI\_RxFifo7*** SPI rx watermark at 7 items.

***kSPI\_RxFifo8*** SPI rx watermark at 8 items.

### 16.3.4.7 enum spi\_data\_width\_t

Enumerator

|                        |                    |
|------------------------|--------------------|
| <i>kSPI_Data4Bits</i>  | 4 bits data width  |
| <i>kSPI_Data5Bits</i>  | 5 bits data width  |
| <i>kSPI_Data6Bits</i>  | 6 bits data width  |
| <i>kSPI_Data7Bits</i>  | 7 bits data width  |
| <i>kSPI_Data8Bits</i>  | 8 bits data width  |
| <i>kSPI_Data9Bits</i>  | 9 bits data width  |
| <i>kSPI_Data10Bits</i> | 10 bits data width |
| <i>kSPI_Data11Bits</i> | 11 bits data width |
| <i>kSPI_Data12Bits</i> | 12 bits data width |
| <i>kSPI_Data13Bits</i> | 13 bits data width |
| <i>kSPI_Data14Bits</i> | 14 bits data width |
| <i>kSPI_Data15Bits</i> | 15 bits data width |
| <i>kSPI_Data16Bits</i> | 16 bits data width |

### 16.3.4.8 enum spi\_ssel\_t

Enumerator

|                   |                 |
|-------------------|-----------------|
| <i>kSPI_Ssel0</i> | Slave select 0. |
| <i>kSPI_Ssel1</i> | Slave select 1. |
| <i>kSPI_Ssel2</i> | Slave select 2. |
| <i>kSPI_Ssel3</i> | Slave select 3. |

### 16.3.4.9 enum \_spi\_status

Enumerator

|                                       |                                                  |
|---------------------------------------|--------------------------------------------------|
| <i>kStatus_SPI_Busy</i>               | SPI bus is busy.                                 |
| <i>kStatus_SPI_Idle</i>               | SPI is idle.                                     |
| <i>kStatus_SPI_Error</i>              | SPI error.                                       |
| <i>kStatus_SPI_BaudrateNotSupport</i> | Baudrate is not support in current clock source. |

### 16.3.4.10 enum \_spi\_interrupt\_enable

Enumerator

|                      |                     |
|----------------------|---------------------|
| <i>kSPI_RxLvllrq</i> | Rx level interrupt. |
| <i>kSPI_TxLvllrq</i> | Tx level interrupt. |

### 16.3.4.11 enum \_spi\_statusflags

Enumerator

*kSPI\_TxEmptyFlag* txFifo is empty  
*kSPI\_TxNotFullFlag* txFifo is not full  
*kSPI\_RxNotEmptyFlag* rxFIFO is not empty  
*kSPI\_RxFullFlag* rxFIFO is full

## 16.3.5 Function Documentation

### 16.3.5.1 uint32\_t SPI\_GetInstance ( SPI\_Type \* *base* )

### 16.3.5.2 void SPI\_MasterGetDefaultConfig ( spi\_master\_config\_t \* *config* )

The purpose of this API is to get the configuration structure initialized for use in [SPI\\_MasterInit\(\)](#). User may use the initialized structure unchanged in [SPI\\_MasterInit\(\)](#), or modify some fields of the structure before calling [SPI\\_MasterInit\(\)](#). After calling this API, the master is ready to transfer. Example:

```
spi_master_config_t config;
SPI_MasterGetDefaultConfig(&config);
```

Parameters

|               |                                    |
|---------------|------------------------------------|
| <i>config</i> | pointer to master config structure |
|---------------|------------------------------------|

### 16.3.5.3 status\_t SPI\_MasterInit ( SPI\_Type \* *base*, const spi\_master\_config\_t \* *config*, uint32\_t *srcClock\_Hz* )

The configuration structure can be filled by user from scratch, or be set with default values by [SPI\\_MasterGetDefaultConfig\(\)](#). After calling this API, the slave is ready to transfer. Example

```
spi_master_config_t config = {
 .baudRate_Bps = 400000,
 ...
};
SPI_MasterInit(SPI0, &config);
```

Parameters

## SPI Driver

|                    |                                           |
|--------------------|-------------------------------------------|
| <i>base</i>        | SPI base pointer                          |
| <i>config</i>      | pointer to master configuration structure |
| <i>srcClock_Hz</i> | Source clock frequency.                   |

### 16.3.5.4 void SPI\_SlaveGetDefaultConfig ( spi\_slave\_config\_t \* *config* )

The purpose of this API is to get the configuration structure initialized for use in [SPI\\_SlaveInit\(\)](#). Modify some fields of the structure before calling [SPI\\_SlaveInit\(\)](#). Example:

```
spi_slave_config_t config;
SPI_SlaveGetDefaultConfig(&config);
```

#### Parameters

|               |                                          |
|---------------|------------------------------------------|
| <i>config</i> | pointer to slave configuration structure |
|---------------|------------------------------------------|

### 16.3.5.5 status\_t SPI\_SlaveInit ( SPI\_Type \* *base*, const spi\_slave\_config\_t \* *config* )

The configuration structure can be filled by user from scratch or be set with default values by [SPI\\_SlaveGetDefaultConfig\(\)](#). After calling this API, the slave is ready to transfer. Example

```
spi_slave_config_t config = {
.polarity = flexSPIClockPolarity_ActiveHigh;
.phase = flexSPIClockPhase_FirstEdge;
.direction = flexSPIMsbFirst;
...
};
SPI_SlaveInit(SPI0, &config);
```

#### Parameters

|               |                                          |
|---------------|------------------------------------------|
| <i>base</i>   | SPI base pointer                         |
| <i>config</i> | pointer to slave configuration structure |

### 16.3.5.6 void SPI\_Deinit ( SPI\_Type \* *base* )

Calling this API resets the SPI module, gates the SPI clock. The SPI module can't work unless calling the SPI\_MasterInit/SPI\_SlaveInit to initialize module.

## Parameters

|             |                  |
|-------------|------------------|
| <i>base</i> | SPI base pointer |
|-------------|------------------|

**16.3.5.7 static void SPI\_Enable ( SPI\_Type \* *base*, bool *enable* ) [inline], [static]**

## Parameters

|               |                                              |
|---------------|----------------------------------------------|
| <i>base</i>   | SPI base pointer                             |
| <i>enable</i> | or disable ( true = enable, false = disable) |

**16.3.5.8 static uint32\_t SPI\_GetStatusFlags ( SPI\_Type \* *base* ) [inline], [static]**

## Parameters

|             |                  |
|-------------|------------------|
| <i>base</i> | SPI base pointer |
|-------------|------------------|

## Returns

SPI Status, use status flag to AND [\\_spi\\_statusflags](#) could get the related status.

**16.3.5.9 static void SPI\_EnableInterrupts ( SPI\_Type \* *base*, uint32\_t *irqs* ) [inline], [static]**

## Parameters

|             |                                                                                                                                                                                |
|-------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i> | SPI base pointer                                                                                                                                                               |
| <i>irqs</i> | SPI interrupt source. The parameter can be any combination of the following values: <ul style="list-style-type: none"> <li>• kSPI_RxLvlIrq</li> <li>• kSPI_TxLvlIrq</li> </ul> |

**16.3.5.10 static void SPI\_DisableInterrupts ( SPI\_Type \* *base*, uint32\_t *irqs* ) [inline], [static]**

## SPI Driver

### Parameters

|             |                                                                                                                                                                             |
|-------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i> | SPI base pointer                                                                                                                                                            |
| <i>irqs</i> | SPI interrupt source. The parameter can be any combination of the following values: <ul style="list-style-type: none"><li>• kSPI_RxLvllrq</li><li>• kSPI_TxLvllrq</li></ul> |

#### 16.3.5.11 void SPI\_EnableTxDMA ( SPI\_Type \* *base*, bool *enable* )

### Parameters

|               |                                                |
|---------------|------------------------------------------------|
| <i>base</i>   | SPI base pointer                               |
| <i>enable</i> | True means enable DMA, false means disable DMA |

#### 16.3.5.12 void SPI\_EnableRxDMA ( SPI\_Type \* *base*, bool *enable* )

### Parameters

|               |                                                |
|---------------|------------------------------------------------|
| <i>base</i>   | SPI base pointer                               |
| <i>enable</i> | True means enable DMA, false means disable DMA |

#### 16.3.5.13 void\* SPI\_GetConfig ( SPI\_Type \* *base* )

### Parameters

|             |                         |
|-------------|-------------------------|
| <i>base</i> | SPI peripheral address. |
|-------------|-------------------------|

### Returns

return configurations which contain datawidth and SSEL numbers. return data type is a pointer of [spi\\_config\\_t](#).

#### 16.3.5.14 status\_t SPI\_MasterSetBaud ( SPI\_Type \* *base*, uint32\_t *baudrate\_Bps*, uint32\_t *srcClock\_Hz* )

This is only used in master.

## Parameters

|                     |                                   |
|---------------------|-----------------------------------|
| <i>base</i>         | SPI base pointer                  |
| <i>baudrate_Bps</i> | baud rate needed in Hz.           |
| <i>srcClock_Hz</i>  | SPI source clock frequency in Hz. |

**16.3.5.15 void SPI\_WriteData ( SPI\_Type \* *base*, uint16\_t *data*, uint32\_t *configFlags* )**

## Parameters

|                    |                                                                  |
|--------------------|------------------------------------------------------------------|
| <i>base</i>        | SPI base pointer                                                 |
| <i>data</i>        | needs to be write.                                               |
| <i>configFlags</i> | transfer configuration options <a href="#">spi_xfer_option_t</a> |

**16.3.5.16 static uint32\_t SPI\_ReadData ( SPI\_Type \* *base* ) [inline], [static]**

## Parameters

|             |                  |
|-------------|------------------|
| <i>base</i> | SPI base pointer |
|-------------|------------------|

## Returns

Data in the register.

**16.3.5.17 static void SPI\_SetTransferDelay ( SPI\_Type \* *base*, const spi\_delay\_config\_t \* *config* ) [inline], [static]**

the delay uint is SPI clock time, maximum value is 0xF.

## Parameters

|               |                                                                     |
|---------------|---------------------------------------------------------------------|
| <i>base</i>   | SPI base pointer                                                    |
| <i>config</i> | configuration for delay option <a href="#">spi_delay_config_t</a> . |

**16.3.5.18 void SPI\_SetDummyData ( SPI\_Type \* *base*, uint8\_t *dummyData* )**

## SPI Driver

### Parameters

|                  |                                                |
|------------------|------------------------------------------------|
| <i>base</i>      | SPI peripheral address.                        |
| <i>dummyData</i> | Data to be transferred when tx buffer is NULL. |

**16.3.5.19** `status_t SPI_MasterTransferCreateHandle ( SPI_Type * base, spi_master_handle_t * handle, spi_master_callback_t callback, void * userData )`

This function initializes the SPI master handle which can be used for other SPI master transactional APIs. Usually, for a specified SPI instance, call this API once to get the initialized handle.

### Parameters

|                 |                              |
|-----------------|------------------------------|
| <i>base</i>     | SPI peripheral base address. |
| <i>handle</i>   | SPI handle pointer.          |
| <i>callback</i> | Callback function.           |
| <i>userData</i> | User data.                   |

**16.3.5.20** `status_t SPI_MasterTransferBlocking ( SPI_Type * base, spi_transfer_t * xfer )`

### Parameters

|             |                                        |
|-------------|----------------------------------------|
| <i>base</i> | SPI base pointer                       |
| <i>xfer</i> | pointer to spi_xfer_config_t structure |

### Return values

|                                |                                |
|--------------------------------|--------------------------------|
| <i>kStatus_Success</i>         | Successfully start a transfer. |
| <i>kStatus_InvalidArgument</i> | Input argument is invalid.     |

**16.3.5.21** `status_t SPI_MasterTransferNonBlocking ( SPI_Type * base, spi_master_handle_t * handle, spi_transfer_t * xfer )`



## Parameters

|               |                                                                          |
|---------------|--------------------------------------------------------------------------|
| <i>base</i>   | SPI peripheral base address.                                             |
| <i>handle</i> | pointer to spi_master_handle_t structure which stores the transfer state |
| <i>xfer</i>   | pointer to spi_xfer_config_t structure                                   |

## Return values

|                                |                                               |
|--------------------------------|-----------------------------------------------|
| <i>kStatus_Success</i>         | Successfully start a transfer.                |
| <i>kStatus_InvalidArgument</i> | Input argument is invalid.                    |
| <i>kStatus_SPI_Busy</i>        | SPI is not idle, is running another transfer. |

### 16.3.5.22 status\_t SPI\_MasterHalfDuplexTransferBlocking ( SPI\_Type \* *base*, spi\_half\_duplex\_transfer\_t \* *xfer* )

This function will do a half-duplex transfer for SPI master, This is a blocking function, which does not return until all transfer have been completed. And data transfer mechanism is half-duplex, users can set transmit first or receive first.

## Parameters

|             |                                                                 |
|-------------|-----------------------------------------------------------------|
| <i>base</i> | SPI base pointer                                                |
| <i>xfer</i> | pointer to <a href="#">spi_half_duplex_transfer_t</a> structure |

## Returns

status of status\_t.

### 16.3.5.23 status\_t SPI\_MasterHalfDuplexTransferNonBlocking ( SPI\_Type \* *base*, spi\_master\_handle\_t \* *handle*, spi\_half\_duplex\_transfer\_t \* *xfer* )

This function using polling way to do the first half transmission and using interrupts to do the second half transmission, the transfer mechanism is half-duplex. When do the second half transmission, code will return right away. When all data is transferred, the callback function is called.

## Parameters

## SPI Driver

|               |                                                                          |
|---------------|--------------------------------------------------------------------------|
| <i>base</i>   | SPI peripheral base address.                                             |
| <i>handle</i> | pointer to spi_master_handle_t structure which stores the transfer state |
| <i>xfer</i>   | pointer to <a href="#">spi_half_duplex_transfer_t</a> structure          |

Returns

status of status\_t.

### 16.3.5.24 status\_t SPI\_MasterTransferGetCount ( SPI\_Type \* *base*, spi\_master\_handle\_t \* *handle*, size\_t \* *count* )

This function gets the master transfer count.

Parameters

|               |                                                                               |
|---------------|-------------------------------------------------------------------------------|
| <i>base</i>   | SPI peripheral base address.                                                  |
| <i>handle</i> | Pointer to the spi_master_handle_t structure which stores the transfer state. |
| <i>count</i>  | The number of bytes transferred by using the non-blocking transaction.        |

Returns

status of status\_t.

### 16.3.5.25 void SPI\_MasterTransferAbort ( SPI\_Type \* *base*, spi\_master\_handle\_t \* *handle* )

This function aborts a transfer using an interrupt.

Parameters

|               |                                                                               |
|---------------|-------------------------------------------------------------------------------|
| <i>base</i>   | SPI peripheral base address.                                                  |
| <i>handle</i> | Pointer to the spi_master_handle_t structure which stores the transfer state. |

### 16.3.5.26 void SPI\_MasterTransferHandleIRQ ( SPI\_Type \* *base*, spi\_master\_handle\_t \* *handle* )

## Parameters

|               |                                                                           |
|---------------|---------------------------------------------------------------------------|
| <i>base</i>   | SPI peripheral base address.                                              |
| <i>handle</i> | pointer to spi_master_handle_t structure which stores the transfer state. |

**16.3.5.27 static status\_t SPI\_SlaveTransferCreateHandle ( SPI\_Type \* *base*, spi\_slave\_handle\_t \* *handle*, spi\_slave\_callback\_t *callback*, void \* *userData* ) [inline], [static]**

This function initializes the SPI slave handle which can be used for other SPI slave transactional APIs. Usually, for a specified SPI instance, call this API once to get the initialized handle.

## Parameters

|                 |                              |
|-----------------|------------------------------|
| <i>base</i>     | SPI peripheral base address. |
| <i>handle</i>   | SPI handle pointer.          |
| <i>callback</i> | Callback function.           |
| <i>userData</i> | User data.                   |

**16.3.5.28 static status\_t SPI\_SlaveTransferNonBlocking ( SPI\_Type \* *base*, spi\_slave\_handle\_t \* *handle*, spi\_transfer\_t \* *xfer* ) [inline], [static]**

## Note

The API returns immediately after the transfer initialization is finished.

## Parameters

|               |                                                                          |
|---------------|--------------------------------------------------------------------------|
| <i>base</i>   | SPI peripheral base address.                                             |
| <i>handle</i> | pointer to spi_master_handle_t structure which stores the transfer state |
| <i>xfer</i>   | pointer to spi_xfer_config_t structure                                   |

## Return values

|                                |                                               |
|--------------------------------|-----------------------------------------------|
| <i>kStatus_Success</i>         | Successfully start a transfer.                |
| <i>kStatus_InvalidArgument</i> | Input argument is invalid.                    |
| <i>kStatus_SPI_Busy</i>        | SPI is not idle, is running another transfer. |

**16.3.5.29** `static status_t SPI_SlaveTransferGetCount ( SPI_Type * base,  
spi_slave_handle_t * handle, size_t * count ) [inline], [static]`

This function gets the slave transfer count.

## Parameters

|               |                                                                               |
|---------------|-------------------------------------------------------------------------------|
| <i>base</i>   | SPI peripheral base address.                                                  |
| <i>handle</i> | Pointer to the spi_master_handle_t structure which stores the transfer state. |
| <i>count</i>  | The number of bytes transferred by using the non-blocking transaction.        |

## Returns

status of status\_t.

**16.3.5.30 static void SPI\_SlaveTransferAbort ( SPI\_Type \* *base*, spi\_slave\_handle\_t \* *handle* ) [inline], [static]**

This function aborts a transfer using an interrupt.

## Parameters

|               |                                                                              |
|---------------|------------------------------------------------------------------------------|
| <i>base</i>   | SPI peripheral base address.                                                 |
| <i>handle</i> | Pointer to the spi_slave_handle_t structure which stores the transfer state. |

**16.3.5.31 static void SPI\_SlaveTransferHandleIRQ ( SPI\_Type \* *base*, spi\_slave\_handle\_t \* *handle* ) [inline], [static]**

## Parameters

|               |                                                                         |
|---------------|-------------------------------------------------------------------------|
| <i>base</i>   | SPI peripheral base address.                                            |
| <i>handle</i> | pointer to spi_slave_handle_t structure which stores the transfer state |

## 16.3.6 Variable Documentation

**16.3.6.1 volatile uint8\_t s\_dummyData[]**

### 16.4 SPI DMA Driver

#### 16.4.1 Overview

This section describes the programming interface of the SPI DMA driver.

#### Files

- file [fsl\\_spi\\_dma.h](#)

#### Data Structures

- struct [spi\\_dma\\_handle\\_t](#)  
*SPI DMA transfer handle, users should not touch the content of the handle. [More...](#)*

#### Typedefs

- typedef void(\* [spi\\_dma\\_callback\\_t](#))(SPI\_Type \*base, spi\_dma\_handle\_t \*handle, [status\\_t](#) status, void \*userData)  
*SPI DMA callback called at the end of transfer.*

#### Driver version

- #define [FSL\\_SPI\\_DMA\\_DRIVER\\_VERSION](#) ([MAKE\\_VERSION](#)(2, 0, 4))  
*SPI DMA driver version 2.0.4.*

#### DMA Transactional

- [status\\_t SPI\\_MasterTransferCreateHandleDMA](#) (SPI\_Type \*base, spi\_dma\_handle\_t \*handle, [spi\\_dma\\_callback\\_t](#) callback, void \*userData, [dma\\_handle\\_t](#) \*txHandle, [dma\\_handle\\_t](#) \*rxHandle)  
*Initialize the SPI master DMA handle.*
- [status\\_t SPI\\_MasterTransferDMA](#) (SPI\_Type \*base, spi\_dma\_handle\_t \*handle, [spi\\_transfer\\_t](#) \*xfer)  
*Perform a non-blocking SPI transfer using DMA.*
- [status\\_t SPI\\_MasterHalfDuplexTransferDMA](#) (SPI\_Type \*base, spi\_dma\_handle\_t \*handle, [spi\\_half\\_duplex\\_transfer\\_t](#) \*xfer)  
*Transfers a block of data using a DMA method.*
- static [status\\_t SPI\\_SlaveTransferCreateHandleDMA](#) (SPI\_Type \*base, spi\_dma\_handle\_t \*handle, [spi\\_dma\\_callback\\_t](#) callback, void \*userData, [dma\\_handle\\_t](#) \*txHandle, [dma\\_handle\\_t](#) \*rxHandle)  
*Initialize the SPI slave DMA handle.*
- static [status\\_t SPI\\_SlaveTransferDMA](#) (SPI\_Type \*base, spi\_dma\_handle\_t \*handle, [spi\\_transfer\\_t](#) \*xfer)  
*Perform a non-blocking SPI transfer using DMA.*

- void [SPI\\_MasterTransferAbortDMA](#) (SPI\_Type \*base, spi\_dma\_handle\_t \*handle)  
*Abort a SPI transfer using DMA.*
- [status\\_t SPI\\_MasterTransferGetCountDMA](#) (SPI\_Type \*base, spi\_dma\_handle\_t \*handle, size\_t \*count)  
*Gets the master DMA transfer remaining bytes.*
- static void [SPI\\_SlaveTransferAbortDMA](#) (SPI\_Type \*base, spi\_dma\_handle\_t \*handle)  
*Abort a SPI transfer using DMA.*
- static [status\\_t SPI\\_SlaveTransferGetCountDMA](#) (SPI\_Type \*base, spi\_dma\_handle\_t \*handle, size\_t \*count)  
*Gets the slave DMA transfer remaining bytes.*

## 16.4.2 Data Structure Documentation

### 16.4.2.1 struct \_spi\_dma\_handle

#### Data Fields

- volatile bool [txInProgress](#)  
*Send transfer finished.*
- volatile bool [rxInProgress](#)  
*Receive transfer finished.*
- [dma\\_handle\\_t](#) \* [txHandle](#)  
*DMA handler for SPI send.*
- [dma\\_handle\\_t](#) \* [rxHandle](#)  
*DMA handler for SPI receive.*
- uint8\_t [bytesPerFrame](#)  
*Bytes in a frame for SPI transfer.*
- [spi\\_dma\\_callback\\_t](#) [callback](#)  
*Callback for SPI DMA transfer.*
- void \* [userData](#)  
*User Data for SPI DMA callback.*
- uint32\_t [state](#)  
*Internal state of SPI DMA transfer.*
- size\_t [transferSize](#)  
*Bytes need to be transfer.*

### 16.4.3 Macro Definition Documentation

16.4.3.1 **#define FSL\_SPI\_DMA\_DRIVER\_VERSION (MAKE\_VERSION(2, 0, 4))**

### 16.4.4 Typedef Documentation

16.4.4.1 **typedef void(\* spi\_dma\_callback\_t)(SPI\_Type \*base, spi\_dma\_handle\_t \*handle, status\_t status, void \*userData)**

### 16.4.5 Function Documentation

16.4.5.1 **status\_t SPI\_MasterTransferCreateHandleDMA ( SPI\_Type \* *base*, spi\_dma\_handle\_t \* *handle*, spi\_dma\_callback\_t *callback*, void \* *userData*, dma\_handle\_t \* *txHandle*, dma\_handle\_t \* *rxHandle* )**

This function initializes the SPI master DMA handle which can be used for other SPI master transactional APIs. Usually, for a specified SPI instance, user need only call this API once to get the initialized handle.



## Parameters

|                 |                                                                               |
|-----------------|-------------------------------------------------------------------------------|
| <i>base</i>     | SPI peripheral base address.                                                  |
| <i>handle</i>   | SPI handle pointer.                                                           |
| <i>callback</i> | User callback function called at the end of a transfer.                       |
| <i>userData</i> | User data for callback.                                                       |
| <i>txHandle</i> | DMA handle pointer for SPI Tx, the handle shall be static allocated by users. |
| <i>rxHandle</i> | DMA handle pointer for SPI Rx, the handle shall be static allocated by users. |

#### 16.4.5.2 **status\_t SPI\_MasterTransferDMA ( SPI\_Type \* *base*, spi\_dma\_handle\_t \* *handle*, spi\_transfer\_t \* *xfer* )**

## Note

This interface returned immediately after transfer initiates, users should call SPI\_GetTransferStatus to poll the transfer status to check whether SPI transfer finished.

## Parameters

|               |                                    |
|---------------|------------------------------------|
| <i>base</i>   | SPI peripheral base address.       |
| <i>handle</i> | SPI DMA handle pointer.            |
| <i>xfer</i>   | Pointer to dma transfer structure. |

## Return values

|                                |                                               |
|--------------------------------|-----------------------------------------------|
| <i>kStatus_Success</i>         | Successfully start a transfer.                |
| <i>kStatus_InvalidArgument</i> | Input argument is invalid.                    |
| <i>kStatus_SPI_Busy</i>        | SPI is not idle, is running another transfer. |

#### 16.4.5.3 **status\_t SPI\_MasterHalfDuplexTransferDMA ( SPI\_Type \* *base*, spi\_dma\_handle\_t \* *handle*, spi\_half\_duplex\_transfer\_t \* *xfer* )**

This function using polling way to do the first half transimission and using DMA way to do the srcond half transimission, the transfer mechanism is half-duplex. When do the second half transimission, code will return right away. When all data is transferred, the callback function is called.

## SPI DMA Driver

### Parameters

|                 |                                                                                                  |
|-----------------|--------------------------------------------------------------------------------------------------|
| <i>base</i>     | SPI base pointer                                                                                 |
| <i>handle</i>   | A pointer to the <code>spi_master_dma_handle_t</code> structure which stores the transfer state. |
| <i>transfer</i> | A pointer to the <a href="#">spi_half_duplex_transfer_t</a> structure.                           |

### Returns

status of `status_t`.

**16.4.5.4 static status\_t SPI\_SlaveTransferCreateHandleDMA ( SPI\_Type \* *base*, spi\_dma\_handle\_t \* *handle*, spi\_dma\_callback\_t *callback*, void \* *userData*, dma\_handle\_t \* *txHandle*, dma\_handle\_t \* *rxHandle* ) [inline], [static]**

This function initializes the SPI slave DMA handle which can be used for other SPI master transactional APIs. Usually, for a specified SPI instance, user need only call this API once to get the initialized handle.

### Parameters

|                 |                                                                               |
|-----------------|-------------------------------------------------------------------------------|
| <i>base</i>     | SPI peripheral base address.                                                  |
| <i>handle</i>   | SPI handle pointer.                                                           |
| <i>callback</i> | User callback function called at the end of a transfer.                       |
| <i>userData</i> | User data for callback.                                                       |
| <i>txHandle</i> | DMA handle pointer for SPI Tx, the handle shall be static allocated by users. |
| <i>rxHandle</i> | DMA handle pointer for SPI Rx, the handle shall be static allocated by users. |

**16.4.5.5 static status\_t SPI\_SlaveTransferDMA ( SPI\_Type \* *base*, spi\_dma\_handle\_t \* *handle*, spi\_transfer\_t \* *xfer* ) [inline], [static]**

### Note

This interface returned immediately after transfer initiates, users should call `SPI_GetTransferStatus` to poll the transfer status to check whether SPI transfer finished.

### Parameters

|               |                                    |
|---------------|------------------------------------|
| <i>base</i>   | SPI peripheral base address.       |
| <i>handle</i> | SPI DMA handle pointer.            |
| <i>xfer</i>   | Pointer to dma transfer structure. |

Return values

|                                |                                               |
|--------------------------------|-----------------------------------------------|
| <i>kStatus_Success</i>         | Successfully start a transfer.                |
| <i>kStatus_InvalidArgument</i> | Input argument is invalid.                    |
| <i>kStatus_SPI_Busy</i>        | SPI is not idle, is running another transfer. |

#### 16.4.5.6 void SPI\_MasterTransferAbortDMA ( SPI\_Type \* *base*, spi\_dma\_handle\_t \* *handle* )

Parameters

|               |                              |
|---------------|------------------------------|
| <i>base</i>   | SPI peripheral base address. |
| <i>handle</i> | SPI DMA handle pointer.      |

#### 16.4.5.7 status\_t SPI\_MasterTransferGetCountDMA ( SPI\_Type \* *base*, spi\_dma\_handle\_t \* *handle*, size\_t \* *count* )

This function gets the master DMA transfer remaining bytes.

Parameters

|               |                                                                              |
|---------------|------------------------------------------------------------------------------|
| <i>base</i>   | SPI peripheral base address.                                                 |
| <i>handle</i> | A pointer to the spi_dma_handle_t structure which stores the transfer state. |
| <i>count</i>  | A number of bytes transferred by the non-blocking transaction.               |

Returns

status of status\_t.

#### 16.4.5.8 static void SPI\_SlaveTransferAbortDMA ( SPI\_Type \* *base*, spi\_dma\_handle\_t \* *handle* ) [inline], [static]

## SPI DMA Driver

### Parameters

|               |                              |
|---------------|------------------------------|
| <i>base</i>   | SPI peripheral base address. |
| <i>handle</i> | SPI DMA handle pointer.      |

#### 16.4.5.9 static status\_t SPI\_SlaveTransferGetCountDMA ( SPI\_Type \* *base*, spi\_dma\_handle\_t \* *handle*, size\_t \* *count* ) [inline], [static]

This function gets the slave DMA transfer remaining bytes.

### Parameters

|               |                                                                              |
|---------------|------------------------------------------------------------------------------|
| <i>base</i>   | SPI peripheral base address.                                                 |
| <i>handle</i> | A pointer to the spi_dma_handle_t structure which stores the transfer state. |
| <i>count</i>  | A number of bytes transferred by the non-blocking transaction.               |

### Returns

status of status\_t.

## 16.5 SPI FreeRTOS driver

### 16.5.1 Overview

This section describes the programming interface of the SPI FreeRTOS driver.

#### Files

- file [fsl\\_spi\\_freertos.h](#)

#### Data Structures

- struct [spi\\_rtos\\_handle\\_t](#)  
*SPI FreeRTOS handle. [More...](#)*

#### Driver version

- #define [FSL\\_SPI\\_FREERTOS\\_DRIVER\\_VERSION](#) ([MAKE\\_VERSION](#)(2, 0, 4))  
*SPI freertos driver version 2.0.4.*

#### SPI RTOS Operation

- [status\\_t SPI\\_RTOS\\_Init](#) ([spi\\_rtos\\_handle\\_t](#) \*handle, SPI\_Type \*base, const [spi\\_master\\_config\\_t](#) \*masterConfig, uint32\_t srcClock\_Hz)  
*Initializes SPI.*
- [status\\_t SPI\\_RTOS\\_Deinit](#) ([spi\\_rtos\\_handle\\_t](#) \*handle)  
*Deinitializes the SPI.*
- [status\\_t SPI\\_RTOS\\_Transfer](#) ([spi\\_rtos\\_handle\\_t](#) \*handle, [spi\\_transfer\\_t](#) \*transfer)  
*Performs SPI transfer.*

### 16.5.2 Data Structure Documentation

#### 16.5.2.1 struct spi\_rtos\_handle\_t

##### Data Fields

- SPI\_Type \* [base](#)  
*SPI base address.*
- [spi\\_master\\_handle\\_t](#) [drv\\_handle](#)  
*Handle of the underlying driver, treated as opaque by the RTOS layer.*
- SemaphoreHandle\_t [mutex](#)  
*Mutex to lock the handle during a transfer.*
- SemaphoreHandle\_t [event](#)

## SPI FreeRTOS driver

*Semaphore to notify and unblock task when transfer ends.*

### 16.5.3 Macro Definition Documentation

**16.5.3.1 #define FSL\_SPI\_FREERTOS\_DRIVER\_VERSION (MAKE\_VERSION(2, 0, 4))**

### 16.5.4 Function Documentation

**16.5.4.1 status\_t SPI\_RTOS\_Init ( spi\_rtos\_handle\_t \* *handle*, SPI\_Type \* *base*, const spi\_master\_config\_t \* *masterConfig*, uint32\_t *srcClock\_Hz* )**

This function initializes the SPI module and related RTOS context.

Parameters

|                     |                                                                          |
|---------------------|--------------------------------------------------------------------------|
| <i>handle</i>       | The RTOS SPI handle, the pointer to an allocated space for RTOS context. |
| <i>base</i>         | The pointer base address of the SPI instance to initialize.              |
| <i>masterConfig</i> | Configuration structure to set-up SPI in master mode.                    |
| <i>srcClock_Hz</i>  | Frequency of input clock of the SPI module.                              |

Returns

status of the operation.

**16.5.4.2 status\_t SPI\_RTOS\_Deinit ( spi\_rtos\_handle\_t \* *handle* )**

This function deinitializes the SPI module and related RTOS context.

Parameters

|               |                      |
|---------------|----------------------|
| <i>handle</i> | The RTOS SPI handle. |
|---------------|----------------------|

**16.5.4.3 status\_t SPI\_RTOS\_Transfer ( spi\_rtos\_handle\_t \* *handle*, spi\_transfer\_t \* *transfer* )**

This function performs an SPI transfer according to data given in the transfer structure.

## Parameters

|                 |                                               |
|-----------------|-----------------------------------------------|
| <i>handle</i>   | The RTOS SPI handle.                          |
| <i>transfer</i> | Structure specifying the transfer parameters. |

## Returns

status of the operation.





## Chapter 17

# USART: Universal Asynchronous Receiver/Transmitter Driver

### 17.1 Overview

The MCUXpresso SDK provides a peripheral UART driver for the Universal Synchronous Receiver/-Transmitter (USART) module of MCUXpresso SDK devices. Driver does not support synchronous mode.

The USART driver includes two parts: functional APIs and transactional APIs.

Functional APIs are used for USART initialization/configuration/operation for optimization/customization purpose. Using the functional API requires the knowledge of the USART peripheral and know how to organize functional APIs to meet the application requirements. All functional API use the peripheral base address as the first parameter. USART functional operation groups provide the functional APIs set.

Transactional APIs can be used to enable the peripheral quickly and in the application if the code size and performance of transactional APIs can satisfy the requirements. If the code size and performance are critical requirements, see the transactional API implementation and write custom code. All transactional APIs use the `usart_handle_t` as the second parameter. Initialize the handle by calling the [USART\\_TransferCreateHandle\(\)](#) API.

Transactional APIs support asynchronous transfer, which means that the functions [USART\\_TransferSendNonBlocking\(\)](#) and [USART\\_TransferReceiveNonBlocking\(\)](#) set up an interrupt for data transfer. When the transfer completes, the upper layer is notified through a callback function with the `kStatus_USART_TxIdle` and `kStatus_USART_RxIdle`.

Transactional receive APIs support the ring buffer. Prepare the memory for the ring buffer and pass in the start address and size while calling the [USART\\_TransferCreateHandle\(\)](#). If passing NULL, the ring buffer feature is disabled. When the ring buffer is enabled, the received data is saved to the ring buffer in the background. The [USART\\_TransferReceiveNonBlocking\(\)](#) function first gets data from the ring buffer. If the ring buffer does not have enough data, the function first returns the data in the ring buffer and then saves the received data to user memory. When all data is received, the upper layer is informed through a callback with the `kStatus_USART_RxIdle`.

If the receive ring buffer is full, the upper layer is informed through a callback with the `kStatus_USART_RxRingBufferOverflow`. In the callback function, the upper layer reads data out from the ring buffer. If not, the oldest data is overwritten by the new data.

The ring buffer size is specified when creating the handle. Note that one byte is reserved for the ring buffer maintenance. When creating handle using the following code:

```
USART_TransferCreateHandle(USART0, &handle, USART_UserCallback, NULL);
```

In this example, the buffer size is 32, but only 31 bytes are used for saving data.

## Typical use case

### 17.2 Typical use case

#### 17.2.1 USART Send/receive using a polling method

```
uint8_t ch;
USART_GetDefaultConfig(&user_config);
user_config.baudRate_Bps = 115200U;
user_config.enableTx = true;
user_config.enableRx = true;

USART_Init(USART1, &user_config, 120000000U);

while(1)
{
 USART_ReadBlocking(USART1, &ch, 1);
 USART_WriteBlocking(USART1, &ch, 1);
}
```

#### 17.2.2 USART Send/receive using an interrupt method

```
usart_handle_t g_usartHandle;
usart_config_t user_config;
usart_transfer_t sendXfer;
usart_transfer_t receiveXfer;
volatile bool txFinished;
volatile bool rxFinished;
uint8_t sendData[] = {'H', 'e', 'l', 'l', 'o'};
uint8_t receiveData[32];

void USART_UserCallback(usart_handle_t *handle, status_t status, void *userData)
{
 userData = userData;

 if (kStatus_USART_TxIdle == status)
 {
 txFinished = true;
 }

 if (kStatus_USART_RxIdle == status)
 {
 rxFinished = true;
 }
}

void main(void)
{
 //...

 USART_GetDefaultConfig(&user_config);
 user_config.baudRate_Bps = 115200U;
 user_config.enableTx = true;
 user_config.enableRx = true;

 USART_Init(USART1, &user_config, 120000000U);
 USART_TransferCreateHandle(USART1, &g_usartHandle, USART_UserCallback, NULL);

 // Prepare to send.
 sendXfer.data = sendData;
 sendXfer.dataSize = sizeof(sendData);
 txFinished = false;

 // Send out.
 USART_TransferSendNonBlocking(USART1, &g_usartHandle, &sendXfer);
}
```

```

// Wait send finished.
while (!txFinished)
{
}

// Prepare to receive.
receiveXfer.data = receiveData;
receiveXfer.dataSize = sizeof(receiveData);
rxFinished = false;

// Receive.
USART_TransferReceiveNonBlocking(USART1, &g_usartHandle, &receiveXfer,
 NULL);

// Wait receive finished.
while (!rxFinished)
{
}

// ...
}

```

### 17.2.3 USART Receive using the ringbuffer feature

```

#define RING_BUFFER_SIZE 64
#define RX_DATA_SIZE 32

usart_handle_t g_usartHandle;
usart_config_t user_config;
usart_transfer_t sendXfer;
usart_transfer_t receiveXfer;
volatile bool txFinished;
volatile bool rxFinished;
uint8_t receiveData[RX_DATA_SIZE];
uint8_t ringBuffer[RING_BUFFER_SIZE];

void USART_UserCallback(usart_handle_t *handle, status_t status, void *userData)
{
 userData = userData;

 if (kStatus_USART_RxIdle == status)
 {
 rxFinished = true;
 }
}

void main(void)
{
 size_t bytesRead;
 //...

 USART_GetDefaultConfig(&user_config);
 user_config.baudRate_Bps = 115200U;
 user_config.enableTx = true;
 user_config.enableRx = true;

 USART_Init(USART1, &user_config, 120000000U);
 USART_TransferCreateHandle(USART1, &g_usartHandle, USART_UserCallback, NULL);
 USART_TransferStartRingBuffer(USART1, &g_usartHandle, ringBuffer,
 RING_BUFFER_SIZE);
 // Now the RX is working in background, receive in to ring buffer.

 // Prepare to receive.
 receiveXfer.data = receiveData;
 receiveXfer.dataSize = sizeof(receiveData);
}

```

## Typical use case

```
rxFinished = false;

// Receive.
USART_TransferReceiveNonBlocking(USART1, &g_usartHandle, &receiveXfer);

if (bytesRead == RX_DATA_SIZE) /* Have read enough data. */
{
 ;
}
else
{
 if (bytesRead) /* Received some data, process first. */
 {
 ;
 }

 // Wait receive finished.
 while (!rxFinished)
 {
 }
}

// ...
}
```

### 17.2.4 USART Send/Receive using the DMA method

```
usart_handle_t g_usartHandle;
dma_handle_t g_usartTxDmaHandle;
dma_handle_t g_usartRxDmaHandle;
usart_config_t user_config;
usart_transfer_t sendXfer;
usart_transfer_t receiveXfer;
volatile bool txFinished;
volatile bool rxFinished;
uint8_t sendData[] = {'H', 'e', 'l', 'l', 'o'};
uint8_t receiveData[32];

void USART_UserCallback(usart_handle_t *handle, status_t status, void *userData)
{
 userData = userData;

 if (kStatus_USART_TxIdle == status)
 {
 txFinished = true;
 }

 if (kStatus_USART_RxIdle == status)
 {
 rxFinished = true;
 }
}

void main(void)
{
 //...

 USART_GetDefaultConfig(&user_config);
 user_config.baudRate_Bps = 115200U;
 user_config.enableTx = true;
 user_config.enableRx = true;

 USART_Init(USART1, &user_config, 120000000U);

 // Set up the DMA
```

```

DMA_Init(DMA0);
DMA_EnableChannel(DMA0, USART_TX_DMA_CHANNEL);
DMA_EnableChannel(DMA0, USART_RX_DMA_CHANNEL);

DMA_CreateHandle(&g_usartTxDmaHandle, DMA0, USART_TX_DMA_CHANNEL);
DMA_CreateHandle(&g_usartRxDmaHandle, DMA0, USART_RX_DMA_CHANNEL);

USART_TransferCreateHandleDMA(USART1, &g_usartHandle, USART_UserCallback,
 NULL, &g_usartTxDmaHandle, &g_usartRxDmaHandle);

// Prepare to send.
sendXfer.data = sendData;
sendXfer.dataSize = sizeof(sendData);
txFinished = false;

// Send out.
USART_TransferSendDMA(USART1, &g_usartHandle, &sendXfer);

// Wait send finished.
while (!txFinished)
{
}

// Prepare to receive.
receiveXfer.data = receiveData;
receiveXfer.dataSize = sizeof(receiveData);
rxFinished = false;

// Receive.
USART_TransferReceiveDMA(USART1, &g_usartHandle, &receiveXfer);

// Wait receive finished.
while (!rxFinished)
{
}

// ...
}

```

## Modules

- [USART DMA Driver](#)
- [USART Driver](#)
- [USART FreeRTOS Driver](#)

### 17.3 USART Driver

#### 17.3.1 Overview

##### Data Structures

- struct `usart_config_t`  
*USART configuration structure. [More...](#)*
- struct `usart_transfer_t`  
*USART transfer structure. [More...](#)*
- struct `usart_handle_t`  
*USART handle structure. [More...](#)*

##### Typedefs

- typedef `void(* usart_transfer_callback_t)(USART_Type *base, usart_handle_t *handle, status_t status, void *userData)`  
*USART transfer callback function.*

##### Enumerations

- enum `_usart_status` {  
    `kStatus_USART_TxBusy` = MAKE\_STATUS(kStatusGroup\_LPC\_USART, 0),  
    `kStatus_USART_RxBusy` = MAKE\_STATUS(kStatusGroup\_LPC\_USART, 1),  
    `kStatus_USART_TxIdle` = MAKE\_STATUS(kStatusGroup\_LPC\_USART, 2),  
    `kStatus_USART_RxIdle` = MAKE\_STATUS(kStatusGroup\_LPC\_USART, 3),  
    `kStatus_USART_TxError` = MAKE\_STATUS(kStatusGroup\_LPC\_USART, 7),  
    `kStatus_USART_RxError` = MAKE\_STATUS(kStatusGroup\_LPC\_USART, 9),  
    `kStatus_USART_RxRingBufferOverrun` = MAKE\_STATUS(kStatusGroup\_LPC\_USART, 8),  
    `kStatus_USART_NoiseError` = MAKE\_STATUS(kStatusGroup\_LPC\_USART, 10),  
    `kStatus_USART_FramingError` = MAKE\_STATUS(kStatusGroup\_LPC\_USART, 11),  
    `kStatus_USART_ParityError` = MAKE\_STATUS(kStatusGroup\_LPC\_USART, 12),  
    `kStatus_USART_BaudrateNotSupport` }  
    *Error codes for the USART driver.*
- enum `usart_sync_mode_t` {  
    `kUSART_SyncModeDisabled` = 0x0U,  
    `kUSART_SyncModeSlave` = 0x2U,  
    `kUSART_SyncModeMaster` = 0x3U }  
    *USART synchronous mode.*
- enum `usart_parity_mode_t` {  
    `kUSART_ParityDisabled` = 0x0U,  
    `kUSART_ParityEven` = 0x2U,  
    `kUSART_ParityOdd` = 0x3U }  
    *USART parity mode.*

- enum `usart_stop_bit_count_t` {  
`kUSART_OneStopBit` = 0U,  
`kUSART_TwoStopBit` = 1U }  
*USART stop bit count.*
- enum `usart_data_len_t` {  
`kUSART_7BitsPerChar` = 0U,  
`kUSART_8BitsPerChar` = 1U }  
*USART data size.*
- enum `usart_clock_polarity_t` {  
`kUSART_RxSampleOnFallingEdge` = 0x0U,  
`kUSART_RxSampleOnRisingEdge` = 0x1U }  
*USART clock polarity configuration, used in sync mode.*
- enum `usart_txfifo_watermark_t` {  
`kUSART_TxFifo0` = 0,  
`kUSART_TxFifo1` = 1,  
`kUSART_TxFifo2` = 2,  
`kUSART_TxFifo3` = 3,  
`kUSART_TxFifo4` = 4,  
`kUSART_TxFifo5` = 5,  
`kUSART_TxFifo6` = 6,  
`kUSART_TxFifo7` = 7 }  
*txFIFO watermark values*
- enum `usart_rxfifo_watermark_t` {  
`kUSART_RxFifo1` = 0,  
`kUSART_RxFifo2` = 1,  
`kUSART_RxFifo3` = 2,  
`kUSART_RxFifo4` = 3,  
`kUSART_RxFifo5` = 4,  
`kUSART_RxFifo6` = 5,  
`kUSART_RxFifo7` = 6,  
`kUSART_RxFifo8` = 7 }  
*rxFIFO watermark values*
- enum `_usart_interrupt_enable`  
*USART interrupt configuration structure, default settings all disabled.*
- enum `_usart_flags` {  
`kUSART_TxError` = (USART\_FIFOSTAT\_TXERR\_MASK),  
`kUSART_RxError` = (USART\_FIFOSTAT\_RXERR\_MASK),  
`kUSART_TxFifoEmptyFlag` = (USART\_FIFOSTAT\_TXEMPTY\_MASK),  
`kUSART_TxFifoNotFullFlag` = (USART\_FIFOSTAT\_TXNOTFULL\_MASK),  
`kUSART_RxFifoNotEmptyFlag` = (USART\_FIFOSTAT\_RXNOTEMPTY\_MASK),  
`kUSART_RxFifoFullFlag` = (USART\_FIFOSTAT\_RXFULL\_MASK) }  
*USART status flags.*

## Functions

- `uint32_t USART_GetInstance` (USART\_Type \*base)

## USART Driver

*Returns instance number for USART peripheral base address.*

### Driver version

- #define **FSL\_USART\_DRIVER\_VERSION** (**MAKE\_VERSION**(2, 1, 0))  
*USART driver version 2.1.0.*

### Initialization and deinitialization

- **status\_t USART\_Init** (USART\_Type \*base, const **usart\_config\_t** \*config, uint32\_t srcClock\_Hz)  
*Initializes a USART instance with user configuration structure and peripheral clock.*
- void **USART\_Deinit** (USART\_Type \*base)  
*Deinitializes a USART instance.*
- void **USART\_GetDefaultConfig** (**usart\_config\_t** \*config)  
*Gets the default configuration structure.*
- **status\_t USART\_SetBaudRate** (USART\_Type \*base, uint32\_t baudrate\_Bps, uint32\_t srcClock\_Hz)  
*Sets the USART instance baud rate.*

### Status

- static uint32\_t **USART\_GetStatusFlags** (USART\_Type \*base)  
*Get USART status flags.*
- static void **USART\_ClearStatusFlags** (USART\_Type \*base, uint32\_t mask)  
*Clear USART status flags.*

### Interrupts

- static void **USART\_EnableInterrupts** (USART\_Type \*base, uint32\_t mask)  
*Enables USART interrupts according to the provided mask.*
- static void **USART\_DisableInterrupts** (USART\_Type \*base, uint32\_t mask)  
*Disables USART interrupts according to a provided mask.*
- static uint32\_t **USART\_GetEnabledInterrupts** (USART\_Type \*base)  
*Returns enabled USART interrupts.*
- static void **USART\_EnableTxDMA** (USART\_Type \*base, bool enable)  
*Enable DMA for Tx.*
- static void **USART\_EnableRxDMA** (USART\_Type \*base, bool enable)  
*Enable DMA for Rx.*
- static void **USART\_EnableCTS** (USART\_Type \*base, bool enable)  
*Enable CTS.*
- static void **USART\_EnableContinuousSCLK** (USART\_Type \*base, bool enable)  
*Continuous Clock generation.*
- static void **USART\_EnableAutoClearSCLK** (USART\_Type \*base, bool enable)  
*Enable Continuous Clock generation bit auto clear.*



## Bus Operations

- static void **USART\_WriteByte** (USART\_Type \*base, uint8\_t data)  
*Writes to the FIFOWR register.*
- static uint8\_t **USART\_ReadByte** (USART\_Type \*base)  
*Reads the FIFORD register directly.*
- void **USART\_WriteBlocking** (USART\_Type \*base, const uint8\_t \*data, size\_t length)  
*Writes to the TX register using a blocking method.*
- status\_t **USART\_ReadBlocking** (USART\_Type \*base, uint8\_t \*data, size\_t length)  
*Read RX data register using a blocking method.*

## Transactional

- status\_t **USART\_TransferCreateHandle** (USART\_Type \*base, usart\_handle\_t \*handle, usart\_transfer\_callback\_t callback, void \*userData)  
*Initializes the USART handle.*
- status\_t **USART\_TransferSendNonBlocking** (USART\_Type \*base, usart\_handle\_t \*handle, usart\_transfer\_t \*xfer)  
*Transmits a buffer of data using the interrupt method.*
- void **USART\_TransferStartRingBuffer** (USART\_Type \*base, usart\_handle\_t \*handle, uint8\_t \*ringBuffer, size\_t ringBufferSize)  
*Sets up the RX ring buffer.*
- void **USART\_TransferStopRingBuffer** (USART\_Type \*base, usart\_handle\_t \*handle)  
*Aborts the background transfer and uninstalls the ring buffer.*
- size\_t **USART\_TransferGetRxRingBufferLength** (usart\_handle\_t \*handle)  
*Get the length of received data in RX ring buffer.*
- void **USART\_TransferAbortSend** (USART\_Type \*base, usart\_handle\_t \*handle)  
*Aborts the interrupt-driven data transmit.*
- status\_t **USART\_TransferGetSendCount** (USART\_Type \*base, usart\_handle\_t \*handle, uint32\_t \*count)  
*Get the number of bytes that have been written to USART TX register.*
- status\_t **USART\_TransferReceiveNonBlocking** (USART\_Type \*base, usart\_handle\_t \*handle, usart\_transfer\_t \*xfer, size\_t \*receivedBytes)  
*Receives a buffer of data using an interrupt method.*
- void **USART\_TransferAbortReceive** (USART\_Type \*base, usart\_handle\_t \*handle)  
*Aborts the interrupt-driven data receiving.*
- status\_t **USART\_TransferGetReceiveCount** (USART\_Type \*base, usart\_handle\_t \*handle, uint32\_t \*count)  
*Get the number of bytes that have been received.*
- void **USART\_TransferHandleIRQ** (USART\_Type \*base, usart\_handle\_t \*handle)  
*USART IRQ handle function.*

### 17.3.2 Data Structure Documentation

#### 17.3.2.1 struct usart\_config\_t

##### Data Fields

- uint32\_t [baudRate\\_Bps](#)  
*USART baud rate.*
- usart\_parity\_mode\_t [parityMode](#)  
*Parity mode, disabled (default), even, odd.*
- usart\_stop\_bit\_count\_t [stopBitCount](#)  
*Number of stop bits, 1 stop bit (default) or 2 stop bits.*
- usart\_data\_len\_t [bitCountPerChar](#)  
*Data length - 7 bit, 8 bit.*
- bool [loopback](#)  
*Enable peripheral loopback.*
- bool [enableRx](#)  
*Enable RX.*
- bool [enableTx](#)  
*Enable TX.*
- bool [enableContinuousSCLK](#)  
*USART continuous Clock generation enable in synchronous master mode.*
- usart\_txfifo\_watermark\_t [txWatermark](#)  
*txFIFO watermark*
- usart\_rxfifo\_watermark\_t [rxWatermark](#)  
*rxFIFO watermark*
- usart\_sync\_mode\_t [syncMode](#)  
*Transfer mode select - asynchronous, synchronous master, synchronous slave.*
- usart\_clock\_polarity\_t [clockPolarity](#)  
*Selects the clock polarity and sampling edge in synchronous mode.*

##### 17.3.2.1.0.23 Field Documentation

17.3.2.1.0.23.1 bool usart\_config\_t::enableContinuousSCLK

17.3.2.1.0.23.2 usart\_sync\_mode\_t usart\_config\_t::syncMode

17.3.2.1.0.23.3 usart\_clock\_polarity\_t usart\_config\_t::clockPolarity

#### 17.3.2.2 struct usart\_transfer\_t

##### Data Fields

- uint8\_t \* [data](#)  
*The buffer of data to be transfer.*
- size\_t [dataSize](#)  
*The byte count to be transfer.*

### 17.3.2.2.0.24 Field Documentation

17.3.2.2.0.24.1 `uint8_t* usart_transfer_t::data`

17.3.2.2.0.24.2 `size_t usart_transfer_t::dataSize`

### 17.3.2.3 `struct _usart_handle`

#### Data Fields

- `uint8_t *volatile txData`  
*Address of remaining data to send.*
- `volatile size_t txDataSize`  
*Size of the remaining data to send.*
- `size_t txDataSizeAll`  
*Size of the data to send out.*
- `uint8_t *volatile rxData`  
*Address of remaining data to receive.*
- `volatile size_t rxDataSize`  
*Size of the remaining data to receive.*
- `size_t rxDataSizeAll`  
*Size of the data to receive.*
- `uint8_t * rxRingBuffer`  
*Start address of the receiver ring buffer.*
- `size_t rxRingBufferSize`  
*Size of the ring buffer.*
- `volatile uint16_t rxRingBufferHead`  
*Index for the driver to store received data into ring buffer.*
- `volatile uint16_t rxRingBufferTail`  
*Index for the user to get data from the ring buffer.*
- `usart_transfer_callback_t callback`  
*Callback function.*
- `void * userData`  
*USART callback function parameter.*
- `volatile uint8_t txState`  
*TX transfer state.*
- `volatile uint8_t rxState`  
*RX transfer state.*
- `usart_txfifo_watermark_t txWatermark`  
*txFIFO watermark*
- `usart_rxfifo_watermark_t rxWatermark`  
*rxFIFO watermark*

## USART Driver

### 17.3.2.3.0.25 Field Documentation

- 17.3.2.3.0.25.1 `uint8_t* volatile usart_handle_t::txData`
- 17.3.2.3.0.25.2 `volatile size_t usart_handle_t::txDataSize`
- 17.3.2.3.0.25.3 `size_t usart_handle_t::txDataSizeAll`
- 17.3.2.3.0.25.4 `uint8_t* volatile usart_handle_t::rxData`
- 17.3.2.3.0.25.5 `volatile size_t usart_handle_t::rxDataSize`
- 17.3.2.3.0.25.6 `size_t usart_handle_t::rxDataSizeAll`
- 17.3.2.3.0.25.7 `uint8_t* usart_handle_t::rxRingBuffer`
- 17.3.2.3.0.25.8 `size_t usart_handle_t::rxRingBufferSize`
- 17.3.2.3.0.25.9 `volatile uint16_t usart_handle_t::rxRingBufferHead`
- 17.3.2.3.0.25.10 `volatile uint16_t usart_handle_t::rxRingBufferTail`
- 17.3.2.3.0.25.11 `usart_transfer_callback_t usart_handle_t::callback`
- 17.3.2.3.0.25.12 `void* usart_handle_t::userData`
- 17.3.2.3.0.25.13 `volatile uint8_t usart_handle_t::txState`

### 17.3.3 Macro Definition Documentation

- 17.3.3.1 `#define FSL_USART_DRIVER_VERSION (MAKE_VERSION(2, 1, 0))`

### 17.3.4 Typedef Documentation

- 17.3.4.1 `typedef void(* usart_transfer_callback_t)(USART_Type *base, usart_handle_t *handle, status_t status, void *userData)`

### 17.3.5 Enumeration Type Documentation

#### 17.3.5.1 `enum _usart_status`

Enumerator

- kStatus\_USART\_TxBusy* Transmitter is busy.
- kStatus\_USART\_RxBusy* Receiver is busy.
- kStatus\_USART\_TxIdle* USART transmitter is idle.
- kStatus\_USART\_RxIdle* USART receiver is idle.
- kStatus\_USART\_TxError* Error happens on txFIFO.

***kStatus\_USART\_RxError*** Error happens on rxFIFO.  
***kStatus\_USART\_RxRingBufferOverflow*** Error happens on rx ring buffer.  
***kStatus\_USART\_NoiseError*** USART noise error.  
***kStatus\_USART\_FramingError*** USART framing error.  
***kStatus\_USART\_ParityError*** USART parity error.  
***kStatus\_USART\_BaudrateNotSupport*** Baudrate is not support in current clock source.

### 17.3.5.2 enum usart\_sync\_mode\_t

Enumerator

***kUSART\_SyncModeDisabled*** Asynchronous mode.  
***kUSART\_SyncModeSlave*** Synchronous slave mode.  
***kUSART\_SyncModeMaster*** Synchronous master mode.

### 17.3.5.3 enum usart\_parity\_mode\_t

Enumerator

***kUSART\_ParityDisabled*** Parity disabled.  
***kUSART\_ParityEven*** Parity enabled, type even, bit setting: PE|PT = 10.  
***kUSART\_ParityOdd*** Parity enabled, type odd, bit setting: PE|PT = 11.

### 17.3.5.4 enum usart\_stop\_bit\_count\_t

Enumerator

***kUSART\_OneStopBit*** One stop bit.  
***kUSART\_TwoStopBit*** Two stop bits.

### 17.3.5.5 enum usart\_data\_len\_t

Enumerator

***kUSART\_7BitsPerChar*** Seven bit mode.  
***kUSART\_8BitsPerChar*** Eight bit mode.

### 17.3.5.6 enum usart\_clock\_polarity\_t

Enumerator

***kUSART\_RxSampleOnFallingEdge*** Un\_RXD is sampled on the falling edge of SCLK.  
***kUSART\_RxSampleOnRisingEdge*** Un\_RXD is sampled on the rising edge of SCLK.

### 17.3.5.7 enum usart\_txfifo\_watermark\_t

Enumerator

***kUSART\_TxFifo0*** USART tx watermark is empty.  
***kUSART\_TxFifo1*** USART tx watermark at 1 item.  
***kUSART\_TxFifo2*** USART tx watermark at 2 items.  
***kUSART\_TxFifo3*** USART tx watermark at 3 items.  
***kUSART\_TxFifo4*** USART tx watermark at 4 items.  
***kUSART\_TxFifo5*** USART tx watermark at 5 items.  
***kUSART\_TxFifo6*** USART tx watermark at 6 items.  
***kUSART\_TxFifo7*** USART tx watermark at 7 items.

### 17.3.5.8 enum usart\_rxfifo\_watermark\_t

Enumerator

***kUSART\_RxFifo1*** USART rx watermark at 1 item.  
***kUSART\_RxFifo2*** USART rx watermark at 2 items.  
***kUSART\_RxFifo3*** USART rx watermark at 3 items.  
***kUSART\_RxFifo4*** USART rx watermark at 4 items.  
***kUSART\_RxFifo5*** USART rx watermark at 5 items.  
***kUSART\_RxFifo6*** USART rx watermark at 6 items.  
***kUSART\_RxFifo7*** USART rx watermark at 7 items.  
***kUSART\_RxFifo8*** USART rx watermark at 8 items.

### 17.3.5.9 enum \_usart\_flags

This provides constants for the USART status flags for use in the USART functions.

Enumerator

***kUSART\_TxError*** TEERR bit, sets if TX buffer is error.  
***kUSART\_RxError*** RXERR bit, sets if RX buffer is error.  
***kUSART\_TxFifoEmptyFlag*** TXEMPTY bit, sets if TX buffer is empty.  
***kUSART\_TxFifoNotFullFlag*** TXNOTFULL bit, sets if TX buffer is not full.  
***kUSART\_RxFifoNotEmptyFlag*** RXNOEMPTY bit, sets if RX buffer is not empty.  
***kUSART\_RxFifoFullFlag*** RXFULL bit, sets if RX buffer is full.

## 17.3.6 Function Documentation

**17.3.6.1** `uint32_t USART_GetInstance ( USART_Type * base )`

**17.3.6.2** `status_t USART_Init ( USART_Type * base, const usart_config_t * config,  
uint32_t srcClock_Hz )`

This function configures the USART module with the user-defined settings. The user can configure the configuration structure and also get the default configuration by using the [USART\\_GetDefaultConfig\(\)](#) function. Example below shows how to use this API to configure USART.

```
* usart_config_t usartConfig;
* usartConfig.baudRate_Bps = 115200U;
* usartConfig.parityMode = kUSART_ParityDisabled;
* usartConfig.stopBitCount = kUSART_OneStopBit;
* USART_Init(USART1, &usartConfig, 20000000U);
*
```

### Parameters

|                    |                                                  |
|--------------------|--------------------------------------------------|
| <i>base</i>        | USART peripheral base address.                   |
| <i>config</i>      | Pointer to user-defined configuration structure. |
| <i>srcClock_Hz</i> | USART clock source frequency in HZ.              |

### Return values

|                                          |                                                  |
|------------------------------------------|--------------------------------------------------|
| <i>kStatus_USART_-BaudrateNotSupport</i> | Baudrate is not support in current clock source. |
| <i>kStatus_InvalidArgument</i>           | USART base address is not valid                  |
| <i>kStatus_Success</i>                   | Status USART initialize succeed                  |

**17.3.6.3** `void USART_Deinit ( USART_Type * base )`

This function waits for TX complete, disables TX and RX, and disables the USART clock.

### Parameters

|             |                                |
|-------------|--------------------------------|
| <i>base</i> | USART peripheral base address. |
|-------------|--------------------------------|

**17.3.6.4** `void USART_GetDefaultConfig ( usart_config_t * config )`

This function initializes the USART configuration structure to a default value. The default values are: usartConfig->baudRate\_Bps = 115200U; usartConfig->parityMode = kUSART\_ParityDisabled; usart-

## USART Driver

```
Config->stopBitCount = kUSART_OneStopBit; usartConfig->bitCountPerChar = kUSART_8BitsPerChar; usartConfig->loopback = false; usartConfig->enableTx = false; usartConfig->enableRx = false;
```



## Parameters

|               |                                     |
|---------------|-------------------------------------|
| <i>config</i> | Pointer to configuration structure. |
|---------------|-------------------------------------|

### 17.3.6.5 status\_t USART\_SetBaudRate ( USART\_Type \* *base*, uint32\_t *baudrate\_Bps*, uint32\_t *srcClock\_Hz* )

This function configures the USART module baud rate. This function is used to update the USART module baud rate after the USART module is initialized by the USART\_Init.

```
* USART_SetBaudRate(USART1, 115200U, 200000000U);
*
```

## Parameters

|                     |                                     |
|---------------------|-------------------------------------|
| <i>base</i>         | USART peripheral base address.      |
| <i>baudrate_Bps</i> | USART baudrate to be set.           |
| <i>srcClock_Hz</i>  | USART clock source frequency in HZ. |

## Return values

|                                          |                                                  |
|------------------------------------------|--------------------------------------------------|
| <i>kStatus_USART_-BaudrateNotSupport</i> | Baudrate is not support in current clock source. |
| <i>kStatus_Success</i>                   | Set baudrate succeed.                            |
| <i>kStatus_InvalidArgument</i>           | One or more arguments are invalid.               |

### 17.3.6.6 static uint32\_t USART\_GetStatusFlags ( USART\_Type \* *base* ) [inline], [static]

This function get all USART status flags, the flags are returned as the logical OR value of the enumerators [\\_usart\\_flags](#). To check a specific status, compare the return value with enumerators in [\\_usart\\_flags](#). For example, to check whether the TX is empty:

```
* if (kUSART_TxFifoNotFullFlag &
* USART_GetStatusFlags(USART1))
* {
* ...
* }
*
```

## USART Driver

### Parameters

|             |                                |
|-------------|--------------------------------|
| <i>base</i> | USART peripheral base address. |
|-------------|--------------------------------|

### Returns

USART status flags which are ORed by the enumerators in the `_usart_flags`.

#### 17.3.6.7 `static void USART_ClearStatusFlags ( USART_Type * base, uint32_t mask )` `[inline], [static]`

This function clear supported USART status flags. Flags that can be cleared or set are: `kUSART_TxError`, `kUSART_RxError`. For example:

```
* USART_ClearStatusFlags(USART1, kUSART_TxError |
* kUSART_RxError)
```

### Parameters

|             |                                |
|-------------|--------------------------------|
| <i>base</i> | USART peripheral base address. |
| <i>mask</i> | status flags to be cleared.    |

#### 17.3.6.8 `static void USART_EnableInterrupts ( USART_Type * base, uint32_t mask )` `[inline], [static]`

This function enables the USART interrupts according to the provided mask. The mask is a logical OR of enumeration members. See [\\_usart\\_interrupt\\_enable](#). For example, to enable TX empty interrupt and RX full interrupt:

```
* USART_EnableInterrupts(USART1, kUSART_TxLevelInterruptEnable |
* kUSART_RxLevelInterruptEnable);
```

### Parameters

|             |                                |
|-------------|--------------------------------|
| <i>base</i> | USART peripheral base address. |
|-------------|--------------------------------|

|             |                                                                                   |
|-------------|-----------------------------------------------------------------------------------|
| <i>mask</i> | The interrupts to enable. Logical OR of <a href="#">_usart_interrupt_enable</a> . |
|-------------|-----------------------------------------------------------------------------------|

#### 17.3.6.9 static void USART\_DisableInterrupts ( USART\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

This function disables the USART interrupts according to a provided mask. The mask is a logical OR of enumeration members. See [\\_usart\\_interrupt\\_enable](#). This example shows how to disable the TX empty interrupt and RX full interrupt:

```
* USART_DisableInterrupts(USART1, kUSART_TxLevelInterruptEnable |
kUSART_RxLevelInterruptEnable);
*
```

##### Parameters

|             |                                                                                    |
|-------------|------------------------------------------------------------------------------------|
| <i>base</i> | USART peripheral base address.                                                     |
| <i>mask</i> | The interrupts to disable. Logical OR of <a href="#">_usart_interrupt_enable</a> . |

#### 17.3.6.10 static uint32\_t USART\_GetEnabledInterrupts ( USART\_Type \* *base* ) [inline], [static]

This function returns the enabled USART interrupts.

##### Parameters

|             |                                |
|-------------|--------------------------------|
| <i>base</i> | USART peripheral base address. |
|-------------|--------------------------------|

#### 17.3.6.11 static void USART\_EnableCTS ( USART\_Type \* *base*, bool *enable* ) [inline], [static]

This function will determine whether CTS is used for flow control.

##### Parameters

|               |                                                           |
|---------------|-----------------------------------------------------------|
| <i>base</i>   | USART peripheral base address.                            |
| <i>enable</i> | Enable CTS or not, true for enable and false for disable. |

## USART Driver

**17.3.6.12 static void USART\_EnableContinuousSCLK ( USART\_Type \* *base*, bool *enable* ) [inline], [static]**

By default, SCLK is only output while data is being transmitted in synchronous mode. Enable this function, SCLK will run continuously in synchronous mode, allowing characters to be received on Un\_RxD independently from transmission on Un\_TXD).

Parameters

|               |                                                                                        |
|---------------|----------------------------------------------------------------------------------------|
| <i>base</i>   | USART peripheral base address.                                                         |
| <i>enable</i> | Enable Continuous Clock generation mode or not, true for enable and false for disable. |

**17.3.6.13 static void USART\_EnableAutoClearSCLK ( USART\_Type \* *base*, bool *enable* ) [inline], [static]**

While enable this function, the Continuous Clock bit is automatically cleared when a complete character has been received. This bit is cleared at the same time.

Parameters

|               |                                                                  |
|---------------|------------------------------------------------------------------|
| <i>base</i>   | USART peripheral base address.                                   |
| <i>enable</i> | Enable auto clear or not, true for enable and false for disable. |

**17.3.6.14 static void USART\_WriteByte ( USART\_Type \* *base*, uint8\_t *data* ) [inline], [static]**

This function writes data to the txFIFO directly. The upper layer must ensure that txFIFO has space for data to write before calling this function.

Parameters

|             |                                |
|-------------|--------------------------------|
| <i>base</i> | USART peripheral base address. |
| <i>data</i> | The byte to write.             |

**17.3.6.15 static uint8\_t USART\_ReadByte ( USART\_Type \* *base* ) [inline], [static]**

This function reads data from the rxFIFO directly. The upper layer must ensure that the rxFIFO is not empty before calling this function.

### Parameters

|             |                                |
|-------------|--------------------------------|
| <i>base</i> | USART peripheral base address. |
|-------------|--------------------------------|

### Returns

The byte read from USART data register.

### 17.3.6.16 void USART\_WriteBlocking ( USART\_Type \* *base*, const uint8\_t \* *data*, size\_t *length* )

This function polls the TX register, waits for the TX register to be empty or for the TX FIFO to have room and writes data to the TX buffer.

### Parameters

|               |                                     |
|---------------|-------------------------------------|
| <i>base</i>   | USART peripheral base address.      |
| <i>data</i>   | Start address of the data to write. |
| <i>length</i> | Size of the data to write.          |

### 17.3.6.17 status\_t USART\_ReadBlocking ( USART\_Type \* *base*, uint8\_t \* *data*, size\_t *length* )

This function polls the RX register, waits for the RX register to be full or for RX FIFO to have data and read data from the TX register.

### Parameters

|               |                                                         |
|---------------|---------------------------------------------------------|
| <i>base</i>   | USART peripheral base address.                          |
| <i>data</i>   | Start address of the buffer to store the received data. |
| <i>length</i> | Size of the buffer.                                     |

### Return values

|                                   |                                                 |
|-----------------------------------|-------------------------------------------------|
| <i>kStatus_USART_FramingError</i> | Receiver overrun happened while receiving data. |
| <i>kStatus_USART_ParityError</i>  | Noise error happened while receiving data.      |
| <i>kStatus_USART_NoiseError</i>   | Framing error happened while receiving data.    |
| <i>kStatus_USART_RxError</i>      | Overflow or underflow rxFIFO happened.          |
| <i>kStatus_Success</i>            | Successfully received all data.                 |

### 17.3.6.18 **status\_t USART\_TransferCreateHandle ( USART\_Type \* *base*, usart\_handle\_t \* *handle*, usart\_transfer\_callback\_t *callback*, void \* *userData* )**

This function initializes the USART handle which can be used for other USART transactional APIs. Usually, for a specified USART instance, call this API once to get the initialized handle.

Parameters

|                 |                                         |
|-----------------|-----------------------------------------|
| <i>base</i>     | USART peripheral base address.          |
| <i>handle</i>   | USART handle pointer.                   |
| <i>callback</i> | The callback function.                  |
| <i>userData</i> | The parameter of the callback function. |

### 17.3.6.19 **status\_t USART\_TransferSendNonBlocking ( USART\_Type \* *base*, usart\_handle\_t \* *handle*, usart\_transfer\_t \* *xfer* )**

This function sends data using an interrupt method. This is a non-blocking function, which returns directly without waiting for all data to be written to the TX register. When all data is written to the TX register in the IRQ handler, the USART driver calls the callback function and passes the [kStatus\\_USART\\_TxIdle](#) as status parameter.

Note

The [kStatus\\_USART\\_TxIdle](#) is passed to the upper layer when all data is written to the TX register. However it does not ensure that all data are sent out. Before disabling the TX, check the [kUSART\\_TransmissionCompleteFlag](#) to ensure that the TX is finished.

Parameters

|               |                                                                  |
|---------------|------------------------------------------------------------------|
| <i>base</i>   | USART peripheral base address.                                   |
| <i>handle</i> | USART handle pointer.                                            |
| <i>xfer</i>   | USART transfer structure. See <a href="#">usart_transfer_t</a> . |

Return values

|                                |                                                                                    |
|--------------------------------|------------------------------------------------------------------------------------|
| <i>kStatus_Success</i>         | Successfully start the data transmission.                                          |
| <i>kStatus_USART_TxBusy</i>    | Previous transmission still not finished, data not all written to TX register yet. |
| <i>kStatus_InvalidArgument</i> | Invalid argument.                                                                  |

### 17.3.6.20 void USART\_TransferStartRingBuffer ( USART\_Type \* *base*, usart\_handle\_t \* *handle*, uint8\_t \* *ringBuffer*, size\_t *ringBufferSize* )

This function sets up the RX ring buffer to a specific USART handle.

When the RX ring buffer is used, data received are stored into the ring buffer even when the user doesn't call the [USART\\_TransferReceiveNonBlocking\(\)](#) API. If there is already data received in the ring buffer, the user can get the received data from the ring buffer directly.

#### Note

When using the RX ring buffer, one byte is reserved for internal use. In other words, if `ringBufferSize` is 32, then only 31 bytes are used for saving data.

#### Parameters

|                       |                                                                                                  |
|-----------------------|--------------------------------------------------------------------------------------------------|
| <i>base</i>           | USART peripheral base address.                                                                   |
| <i>handle</i>         | USART handle pointer.                                                                            |
| <i>ringBuffer</i>     | Start address of the ring buffer for background receiving. Pass NULL to disable the ring buffer. |
| <i>ringBufferSize</i> | size of the ring buffer.                                                                         |

### 17.3.6.21 void USART\_TransferStopRingBuffer ( USART\_Type \* *base*, usart\_handle\_t \* *handle* )

This function aborts the background transfer and uninstalls the ring buffer.

#### Parameters

|               |                                |
|---------------|--------------------------------|
| <i>base</i>   | USART peripheral base address. |
| <i>handle</i> | USART handle pointer.          |

### 17.3.6.22 size\_t USART\_TransferGetRxRingBufferLength ( usart\_handle\_t \* *handle* )

#### Parameters

|               |                       |
|---------------|-----------------------|
| <i>handle</i> | USART handle pointer. |
|---------------|-----------------------|

#### Returns

Length of received data in RX ring buffer.

### 17.3.6.23 void USART\_TransferAbortSend ( USART\_Type \* *base*, usart\_handle\_t \* *handle* )

This function aborts the interrupt driven data sending. The user can get the remainBtyes to find out how many bytes are still not sent out.



## Parameters

|               |                                |
|---------------|--------------------------------|
| <i>base</i>   | USART peripheral base address. |
| <i>handle</i> | USART handle pointer.          |

#### 17.3.6.24 **status\_t USART\_TransferGetSendCount ( USART\_Type \* *base*, usart\_handle\_t \* *handle*, uint32\_t \* *count* )**

This function gets the number of bytes that have been written to USART TX register by interrupt method.

## Parameters

|               |                                |
|---------------|--------------------------------|
| <i>base</i>   | USART peripheral base address. |
| <i>handle</i> | USART handle pointer.          |
| <i>count</i>  | Send bytes count.              |

## Return values

|                                     |                                                       |
|-------------------------------------|-------------------------------------------------------|
| <i>kStatus_NoTransferInProgress</i> | No send in progress.                                  |
| <i>kStatus_InvalidArgument</i>      | Parameter is invalid.                                 |
| <i>kStatus_Success</i>              | Get successfully through the parameter <i>count</i> ; |

#### 17.3.6.25 **status\_t USART\_TransferReceiveNonBlocking ( USART\_Type \* *base*, usart\_handle\_t \* *handle*, usart\_transfer\_t \* *xfer*, size\_t \* *receivedBytes* )**

This function receives data using an interrupt method. This is a non-blocking function, which returns without waiting for all data to be received. If the RX ring buffer is used and not empty, the data in the ring buffer is copied and the parameter *receivedBytes* shows how many bytes are copied from the ring buffer. After copying, if the data in the ring buffer is not enough to read, the receive request is saved by the USART driver. When the new data arrives, the receive request is serviced first. When all data is received, the USART driver notifies the upper layer through a callback function and passes the status parameter *kStatus\_USART\_RxIdle*. For example, the upper layer needs 10 bytes but there are only 5 bytes in the ring buffer. The 5 bytes are copied to the *xfer->data* and this function returns with the parameter *receivedBytes* set to 5. For the left 5 bytes, newly arrived data is saved from the *xfer->data[5]*. When 5 bytes are received, the USART driver notifies the upper layer. If the RX ring buffer is not enabled, this function enables the RX and RX interrupt to receive data to the *xfer->data*. When all data is received, the upper layer is notified.

## USART Driver

### Parameters

|                      |                                                                  |
|----------------------|------------------------------------------------------------------|
| <i>base</i>          | USART peripheral base address.                                   |
| <i>handle</i>        | USART handle pointer.                                            |
| <i>xfer</i>          | USART transfer structure, see <a href="#">usart_transfer_t</a> . |
| <i>receivedBytes</i> | Bytes received from the ring buffer directly.                    |

### Return values

|                                |                                                      |
|--------------------------------|------------------------------------------------------|
| <i>kStatus_Success</i>         | Successfully queue the transfer into transmit queue. |
| <i>kStatus_USART_RxBusy</i>    | Previous receive request is not finished.            |
| <i>kStatus_InvalidArgument</i> | Invalid argument.                                    |

#### 17.3.6.26 void USART\_TransferAbortReceive ( USART\_Type \* *base*, usart\_handle\_t \* *handle* )

This function aborts the interrupt-driven data receiving. The user can get the remainBytes to find out how many bytes not received yet.

### Parameters

|               |                                |
|---------------|--------------------------------|
| <i>base</i>   | USART peripheral base address. |
| <i>handle</i> | USART handle pointer.          |

#### 17.3.6.27 status\_t USART\_TransferGetReceiveCount ( USART\_Type \* *base*, usart\_handle\_t \* *handle*, uint32\_t \* *count* )

This function gets the number of bytes that have been received.

### Parameters

|               |                                |
|---------------|--------------------------------|
| <i>base</i>   | USART peripheral base address. |
| <i>handle</i> | USART handle pointer.          |
| <i>count</i>  | Receive bytes count.           |

Return values

|                                     |                                               |
|-------------------------------------|-----------------------------------------------|
| <i>kStatus_NoTransferInProgress</i> | No receive in progress.                       |
| <i>kStatus_InvalidArgument</i>      | Parameter is invalid.                         |
| <i>kStatus_Success</i>              | Get successfully through the parameter count; |

#### 17.3.6.28 void USART\_TransferHandleIRQ ( USART\_Type \* *base*, usart\_handle\_t \* *handle* )

This function handles the USART transmit and receive IRQ request.

Parameters

|               |                                |
|---------------|--------------------------------|
| <i>base</i>   | USART peripheral base address. |
| <i>handle</i> | USART handle pointer.          |

### 17.4 USART DMA Driver

#### 17.4.1 Overview

##### Files

- file [fsl\\_usart\\_dma.h](#)

##### Data Structures

- struct [usart\\_dma\\_handle\\_t](#)  
*USART DMA handle. [More...](#)*

##### Typedefs

- typedef void(\* [usart\\_dma\\_transfer\\_callback\\_t](#))(USART\_Type \*base, usart\_dma\_handle\_t \*handle, [status\\_t](#) status, void \*userData)  
*USART transfer callback function.*

##### Driver version

- #define [FSL\\_USART\\_DMA\\_DRIVER\\_VERSION](#) ([MAKE\\_VERSION](#)(2, 1, 0))  
*USART dma driver version 2.0.1.*

##### DMA transactional

- [status\\_t](#) [USART\\_TransferCreateHandleDMA](#) (USART\_Type \*base, usart\_dma\_handle\_t \*handle, [usart\\_dma\\_transfer\\_callback\\_t](#) callback, void \*userData, [dma\\_handle\\_t](#) \*txDmaHandle, [dma\\_handle\\_t](#) \*rxDmaHandle)  
*Initializes the USART handle which is used in transactional functions.*
- [status\\_t](#) [USART\\_TransferSendDMA](#) (USART\_Type \*base, usart\_dma\_handle\_t \*handle, [usart\\_transfer\\_t](#) \*xfer)  
*Sends data using DMA.*
- [status\\_t](#) [USART\\_TransferReceiveDMA](#) (USART\_Type \*base, usart\_dma\_handle\_t \*handle, [usart\\_transfer\\_t](#) \*xfer)  
*Receives data using DMA.*
- void [USART\\_TransferAbortSendDMA](#) (USART\_Type \*base, usart\_dma\_handle\_t \*handle)  
*Aborts the sent data using DMA.*
- void [USART\\_TransferAbortReceiveDMA](#) (USART\_Type \*base, usart\_dma\_handle\_t \*handle)  
*Aborts the received data using DMA.*
- [status\\_t](#) [USART\\_TransferGetReceiveCountDMA](#) (USART\_Type \*base, usart\_dma\_handle\_t \*handle, uint32\_t \*count)  
*Get the number of bytes that have been received.*

## 17.4.2 Data Structure Documentation

### 17.4.2.1 struct \_usart\_dma\_handle

#### Data Fields

- USART\_Type \* [base](#)  
*USART peripheral base address.*
- [usart\\_dma\\_transfer\\_callback\\_t](#) [callback](#)  
*Callback function.*
- void \* [userData](#)  
*USART callback function parameter.*
- size\_t [rxDataSizeAll](#)  
*Size of the data to receive.*
- size\_t [txDataSizeAll](#)  
*Size of the data to send out.*
- [dma\\_handle\\_t](#) \* [txDmaHandle](#)  
*The DMA TX channel used.*
- [dma\\_handle\\_t](#) \* [rxDmaHandle](#)  
*The DMA RX channel used.*
- volatile uint8\_t [txState](#)  
*TX transfer state.*
- volatile uint8\_t [rxState](#)  
*RX transfer state.*

## USART DMA Driver

### 17.4.2.1.0.26 Field Documentation

17.4.2.1.0.26.1 **USART\_Type\* usart\_dma\_handle\_t::base**

17.4.2.1.0.26.2 **usart\_dma\_transfer\_callback\_t usart\_dma\_handle\_t::callback**

17.4.2.1.0.26.3 **void\* usart\_dma\_handle\_t::userData**

17.4.2.1.0.26.4 **size\_t usart\_dma\_handle\_t::rxDataSizeAll**

17.4.2.1.0.26.5 **size\_t usart\_dma\_handle\_t::txDataSizeAll**

17.4.2.1.0.26.6 **dma\_handle\_t\* usart\_dma\_handle\_t::txDmaHandle**

17.4.2.1.0.26.7 **dma\_handle\_t\* usart\_dma\_handle\_t::rxDmaHandle**

17.4.2.1.0.26.8 **volatile uint8\_t usart\_dma\_handle\_t::txState**

### 17.4.3 Macro Definition Documentation

17.4.3.1 **#define FSL\_USART\_DMA\_DRIVER\_VERSION (MAKE\_VERSION(2, 1, 0))**

### 17.4.4 Typedef Documentation

17.4.4.1 **typedef void(\* usart\_dma\_transfer\_callback\_t)(USART\_Type \*base,  
usart\_dma\_handle\_t \*handle, status\_t status, void \*userData)**

### 17.4.5 Function Documentation

17.4.5.1 **status\_t USART\_TransferCreateHandleDMA ( USART\_Type \* *base*,  
usart\_dma\_handle\_t \* *handle*, usart\_dma\_transfer\_callback\_t *callback*, void \*  
*userData*, dma\_handle\_t \* *txDmaHandle*, dma\_handle\_t \* *rxDmaHandle* )**

## Parameters

|                    |                                                |
|--------------------|------------------------------------------------|
| <i>base</i>        | USART peripheral base address.                 |
| <i>handle</i>      | Pointer to usart_dma_handle_t structure.       |
| <i>callback</i>    | Callback function.                             |
| <i>userData</i>    | User data.                                     |
| <i>txDmaHandle</i> | User-requested DMA handle for TX DMA transfer. |
| <i>rxDmaHandle</i> | User-requested DMA handle for RX DMA transfer. |

#### 17.4.5.2 status\_t USART\_TransferSendDMA ( USART\_Type \* *base*, usart\_dma\_handle\_t \* *handle*, usart\_transfer\_t \* *xfer* )

This function sends data using DMA. This is a non-blocking function, which returns right away. When all data is sent, the send callback function is called.

## Parameters

|               |                                                                      |
|---------------|----------------------------------------------------------------------|
| <i>base</i>   | USART peripheral base address.                                       |
| <i>handle</i> | USART handle pointer.                                                |
| <i>xfer</i>   | USART DMA transfer structure. See <a href="#">usart_transfer_t</a> . |

## Return values

|                                |                             |
|--------------------------------|-----------------------------|
| <i>kStatus_Success</i>         | if succeed, others failed.  |
| <i>kStatus_USART_TxBusy</i>    | Previous transfer on going. |
| <i>kStatus_InvalidArgument</i> | Invalid argument.           |

#### 17.4.5.3 status\_t USART\_TransferReceiveDMA ( USART\_Type \* *base*, usart\_dma\_handle\_t \* *handle*, usart\_transfer\_t \* *xfer* )

This function receives data using DMA. This is a non-blocking function, which returns right away. When all data is received, the receive callback function is called.

## Parameters

---

## USART DMA Driver

|               |                                                                      |
|---------------|----------------------------------------------------------------------|
| <i>base</i>   | USART peripheral base address.                                       |
| <i>handle</i> | Pointer to usart_dma_handle_t structure.                             |
| <i>xfer</i>   | USART DMA transfer structure. See <a href="#">usart_transfer_t</a> . |

Return values

|                                |                             |
|--------------------------------|-----------------------------|
| <i>kStatus_Success</i>         | if succeed, others failed.  |
| <i>kStatus_USART_RxBusy</i>    | Previous transfer on going. |
| <i>kStatus_InvalidArgument</i> | Invalid argument.           |

### 17.4.5.4 void USART\_TransferAbortSendDMA ( USART\_Type \* *base*, usart\_dma\_handle\_t \* *handle* )

This function aborts send data using DMA.

Parameters

|               |                                         |
|---------------|-----------------------------------------|
| <i>base</i>   | USART peripheral base address           |
| <i>handle</i> | Pointer to usart_dma_handle_t structure |

### 17.4.5.5 void USART\_TransferAbortReceiveDMA ( USART\_Type \* *base*, usart\_dma\_handle\_t \* *handle* )

This function aborts the received data using DMA.

Parameters

|               |                                         |
|---------------|-----------------------------------------|
| <i>base</i>   | USART peripheral base address           |
| <i>handle</i> | Pointer to usart_dma_handle_t structure |

### 17.4.5.6 status\_t USART\_TransferGetReceiveCountDMA ( USART\_Type \* *base*, usart\_dma\_handle\_t \* *handle*, uint32\_t \* *count* )

This function gets the number of bytes that have been received.



## Parameters

|               |                                |
|---------------|--------------------------------|
| <i>base</i>   | USART peripheral base address. |
| <i>handle</i> | USART handle pointer.          |
| <i>count</i>  | Receive bytes count.           |

## Return values

|                                     |                                               |
|-------------------------------------|-----------------------------------------------|
| <i>kStatus_NoTransferInProgress</i> | No receive in progress.                       |
| <i>kStatus_InvalidArgument</i>      | Parameter is invalid.                         |
| <i>kStatus_Success</i>              | Get successfully through the parameter count; |

### 17.5 USART FreeRTOS Driver

#### 17.5.1 Overview

##### Files

- file [fsl\\_usart\\_freertos.h](#)

##### Data Structures

- struct [rtos\\_usart\\_config](#)  
*FLEX USART configuration structure. [More...](#)*
- struct [usart\\_rtos\\_handle\\_t](#)  
*FLEX USART FreeRTOS handle. [More...](#)*

##### Driver version

- #define [FSL\\_USART\\_FREERTOS\\_DRIVER\\_VERSION](#) ([MAKE\\_VERSION](#)(2, 1, 0))  
*USART freertos driver version 2.0.1.*

##### USART RTOS Operation

- int [USART\\_RTOS\\_Init](#) ([usart\\_rtos\\_handle\\_t](#) \*handle, [usart\\_handle\\_t](#) \*t\_handle, const struct [rtos\\_usart\\_config](#) \*cfg)  
*Initializes a USART instance for operation in RTOS.*
- int [USART\\_RTOS\\_Deinit](#) ([usart\\_rtos\\_handle\\_t](#) \*handle)  
*Deinitializes a USART instance for operation.*

##### USART transactional Operation

- int [USART\\_RTOS\\_Send](#) ([usart\\_rtos\\_handle\\_t](#) \*handle, const [uint8\\_t](#) \*buffer, [uint32\\_t](#) length)  
*Sends data in the background.*
- int [USART\\_RTOS\\_Receive](#) ([usart\\_rtos\\_handle\\_t](#) \*handle, [uint8\\_t](#) \*buffer, [uint32\\_t](#) length, [size\\_t](#) \*received)  
*Receives data.*

#### 17.5.2 Data Structure Documentation

##### 17.5.2.1 struct [rtos\\_usart\\_config](#)

##### Data Fields

- USART\_Type \* [base](#)

- *USART base address.*  
uint32\_t [srcclk](#)
- *USART source clock in Hz.*  
uint32\_t [baudrate](#)
- *Desired communication speed.*  
[usart\\_parity\\_mode\\_t](#) [parity](#)
- *Parity setting.*  
[usart\\_stop\\_bit\\_count\\_t](#) [stopbits](#)
- *Number of stop bits to use.*  
uint8\_t \* [buffer](#)
- *Buffer for background reception.*  
uint32\_t [buffer\\_size](#)
- *Size of buffer for background reception.*

### 17.5.2.2 struct usart\_rtos\_handle\_t

#### Data Fields

- USART\_Type \* [base](#)  
*USART base address.*
- [usart\\_transfer\\_t](#) [txTransfer](#)  
*TX transfer structure.*
- [usart\\_transfer\\_t](#) [rxTransfer](#)  
*RX transfer structure.*
- SemaphoreHandle\_t [rxSemaphore](#)  
*RX semaphore for resource sharing.*
- SemaphoreHandle\_t [txSemaphore](#)  
*TX semaphore for resource sharing.*
- EventGroupHandle\_t [rxEvent](#)  
*RX completion event.*
- EventGroupHandle\_t [txEvent](#)  
*TX completion event.*
- void \* [t\\_state](#)  
*Transactional state of the underlying driver.*

### 17.5.3 Macro Definition Documentation

#### 17.5.3.1 #define FSL\_USART\_FREERTOS\_DRIVER\_VERSION (MAKE\_VERSION(2, 1, 0))

### 17.5.4 Function Documentation

#### 17.5.4.1 int USART\_RTOS\_Init ( usart\_rtos\_handle\_t \* *handle*, usart\_handle\_t \* *t\_handle*, const struct rtos\_usart\_config \* *cfg* )

## USART FreeRTOS Driver

### Parameters

|                 |                                                                                     |
|-----------------|-------------------------------------------------------------------------------------|
| <i>handle</i>   | The RTOS USART handle, the pointer to allocated space for RTOS context.             |
| <i>t_handle</i> | The pointer to allocated space where to store transactional layer internal state.   |
| <i>cfg</i>      | The pointer to the parameters required to configure the USART after initialization. |

### Returns

0 succeed, others fail.

#### 17.5.4.2 int USART\_RTOS\_Deinit ( usart\_rtos\_handle\_t \* *handle* )

This function deinitializes the USART module, sets all register values to reset value, and releases the resources.

### Parameters

|               |                        |
|---------------|------------------------|
| <i>handle</i> | The RTOS USART handle. |
|---------------|------------------------|

#### 17.5.4.3 int USART\_RTOS\_Send ( usart\_rtos\_handle\_t \* *handle*, const uint8\_t \* *buffer*, uint32\_t *length* )

This function sends data. It is a synchronous API. If the hardware buffer is full, the task is in the blocked state.

### Parameters

|               |                                |
|---------------|--------------------------------|
| <i>handle</i> | The RTOS USART handle.         |
| <i>buffer</i> | The pointer to buffer to send. |
| <i>length</i> | The number of bytes to send.   |

#### 17.5.4.4 int USART\_RTOS\_Receive ( usart\_rtos\_handle\_t \* *handle*, uint8\_t \* *buffer*, uint32\_t *length*, size\_t \* *received* )

This function receives data from USART. It is a synchronous API. If data is immediately available, it is returned immediately and the number of bytes received.

## Parameters

|                 |                                                                                  |
|-----------------|----------------------------------------------------------------------------------|
| <i>handle</i>   | The RTOS USART handle.                                                           |
| <i>buffer</i>   | The pointer to buffer where to write received data.                              |
| <i>length</i>   | The number of bytes to receive.                                                  |
| <i>received</i> | The pointer to a variable of size_t where the number of received data is filled. |



## Chapter 18

# FMEAS: Frequency Measure Driver

### 18.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Frequency Measure function of MCUXpresso SDK devices' SYSCON module.

It measures frequency of any on-chip or off-chip clock signal. The more precise and higher accuracy clock is selected as a reference clock. The resulting frequency is internally computed from the ratio of value of selected target and reference clock counters.

### 18.2 Frequency Measure Driver operation

[INPUTMUX\\_AttachSignal\(\)](#) function has to be used to select reference and target clock signal sources.

[FMEAS\\_StartMeasure\(\)](#) function starts the measurement cycle.

[FMEAS\\_IsMeasureComplete\(\)](#) can be polled to check if the measurement cycle has finished.

[FMEAS\\_GetFrequency\(\)](#) returns the frequency of the target clock. Frequency of the reference clock has to be provided as a parameter.

### 18.3 Typical use case

Refer to the driver examples codes located at <SDK\_ROOT>/boards/<BOARD>/driver\_examples/fmeas

#### Files

- file [fsl\\_fmeas.h](#)

#### Macros

- #define [FMEAS\\_INDEX](#) 20  
*The calibration duration is  $2^{\text{FMEAS\_INDEX}}$  times the reference clock period.*

#### Driver version

- #define [FSL\\_FMEAS\\_DRIVER\\_VERSION](#) ([MAKE\\_VERSION](#)(2, 1, 1))  
*Defines LPC Frequency Measure driver version 2.1.1.*

### FMEAS Functional Operation

- static void [FMEAS\\_StartMeasure](#) (FMEAS\_SYSCON\_Type \*base)  
*Starts a frequency measurement cycle.*
- static void [FMEAS\\_StartMeasureWithScale](#) (FMEAS\_SYSCON\_Type \*base, uint8\_t scale)  
*Starts a frequency measurement cycle with specific time.*

## Function Documentation

- static bool [FMEAS\\_IsMeasureComplete](#) (FMEAS\_SYSCON\_Type \*base)  
*Indicates when a frequency measurement cycle is complete.*
- uint32\_t [FMEAS\\_GetFrequency](#) (FMEAS\_SYSCON\_Type \*base, uint32\_t refClockRate)  
*Returns the computed value for a frequency measurement cycle.*
- void [FMEAS\\_GetCountWithScale](#) (FMEAS\_SYSCON\_Type \*base, uint8\_t scale, uint32\_t \*refClockCount, uint32\_t \*targetClockCount)  
*Get the clock count during the measurement time.*

## 18.4 Macro Definition Documentation

### 18.4.1 #define FSL\_FMEAS\_DRIVER\_VERSION (MAKE\_VERSION(2, 1, 1))

## 18.5 Function Documentation

### 18.5.1 static void FMEAS\_StartMeasure ( FMEAS\_SYSCON\_Type \* *base* ) [inline], [static]

Parameters

|             |                                   |
|-------------|-----------------------------------|
| <i>base</i> | : SYSCON peripheral base address. |
|-------------|-----------------------------------|

### 18.5.2 static void FMEAS\_StartMeasureWithScale ( FMEAS\_SYSCON\_Type \* *base*, uint8\_t *scale* ) [inline], [static]

Parameters

|              |                                                                                           |
|--------------|-------------------------------------------------------------------------------------------|
| <i>base</i>  | : SYSCON peripheral base address.                                                         |
| <i>scale</i> | : measurement time is 2 <sup>scale</sup> cycle of reference clock, value is from 2 to 31. |

### 18.5.3 static bool FMEAS\_IsMeasureComplete ( FMEAS\_SYSCON\_Type \* *base* ) [inline], [static]

Parameters

|             |                                   |
|-------------|-----------------------------------|
| <i>base</i> | : SYSCON peripheral base address. |
|-------------|-----------------------------------|

Returns

true if a measurement cycle is active, otherwise false.



**18.5.4** `uint32_t FMEAS_GetFrequency ( FMEAS_SYSCON_Type * base, uint32_t refClockRate )`

## Function Documentation

### Parameters

|                     |                                                                     |
|---------------------|---------------------------------------------------------------------|
| <i>base</i>         | : SYSCON peripheral base address.                                   |
| <i>refClockRate</i> | : Reference clock rate used during the frequency measurement cycle. |

### Returns

Frequency in Hz.

### 18.5.5 void FMEAS\_GetCountWithScale ( FMEAS\_SYSCON\_Type \* *base*, uint8\_t *scale*, uint32\_t \* *refClockCount*, uint32\_t \* *targetClockCount* )

### Parameters

|                          |                                                                                           |
|--------------------------|-------------------------------------------------------------------------------------------|
| <i>base</i>              | : SYSCON peripheral base address.                                                         |
| <i>scale</i>             | : measurement time is $2^{\text{scale}}$ cycle of reference clock, value is from 2 to 31. |
| <i>refClockCount</i>     | : Reference clock cycle during the measurement time.                                      |
| <i>targetClock-Count</i> | : Target clock cycle during the measurement time.                                         |

## Chapter 19

# GINT: Group GPIO Input Interrupt Driver

### 19.1 Overview

The MCUXpresso SDK provides a driver for the Group GPIO Input Interrupt (GINT).

It can configure one or more pins to generate a group interrupt when the pin conditions are met. The pins do not have to be configured as GPIO pins.

### 19.2 Group GPIO Input Interrupt Driver operation

[GINT\\_SetCtrl\(\)](#) and [GINT\\_ConfigPins\(\)](#) functions configure the pins.

[GINT\\_EnableCallback\(\)](#) function enables the callback functionality. Callback function is called when the pin conditions are met.

### 19.3 Typical use case

Refer to the driver examples codes located at `<SDK_ROOT>/boards/<BOARD>/driver_examples/gint`

#### Files

- file [fsl\\_gint.h](#)

#### Typedefs

- typedef void(\* [gint\\_cb\\_t](#))(void)  
*GINT Callback function.*

#### Enumerations

- enum [gint\\_comb\\_t](#) {  
    [kGINT\\_CombineOr](#) = 0U,  
    [kGINT\\_CombineAnd](#) = 1U }  
*GINT combine inputs type.*
- enum [gint\\_trig\\_t](#) {  
    [kGINT\\_TrigEdge](#) = 0U,  
    [kGINT\\_TrigLevel](#) = 1U }  
*GINT trigger type.*

#### Functions

- void [GINT\\_Init](#) (GINT\_Type \*base)  
*Initialize GINT peripheral.*
- void [GINT\\_SetCtrl](#) (GINT\_Type \*base, [gint\\_comb\\_t](#) comb, [gint\\_trig\\_t](#) trig, [gint\\_cb\\_t](#) callback)

## Enumeration Type Documentation

- Setup GINT peripheral control parameters.*
  - void [GINT\\_GetCtrl](#) (GINT\_Type \*base, [gint\\_comb\\_t](#) \*comb, [gint\\_trig\\_t](#) \*trig, [gint\\_cb\\_t](#) \*callback)
- Get GINT peripheral control parameters.*
  - void [GINT\\_ConfigPins](#) (GINT\_Type \*base, [gint\\_port\\_t](#) port, uint32\_t polarityMask, uint32\_t enableMask)
- Configure GINT peripheral pins.*
  - void [GINT\\_GetConfigPins](#) (GINT\_Type \*base, [gint\\_port\\_t](#) port, uint32\_t \*polarityMask, uint32\_t \*enableMask)
- Get GINT peripheral pin configuration.*
  - void [GINT\\_EnableCallback](#) (GINT\_Type \*base)
- Enable callback.*
  - void [GINT\\_DisableCallback](#) (GINT\_Type \*base)
- Disable callback.*
  - static void [GINT\\_ClrStatus](#) (GINT\_Type \*base)
- Clear GINT status.*
  - static uint32\_t [GINT\\_GetStatus](#) (GINT\_Type \*base)
- Get GINT status.*
  - void [GINT\\_Deinit](#) (GINT\_Type \*base)
- Deinitialize GINT peripheral.*

## Driver version

- #define [FSL\\_GINT\\_DRIVER\\_VERSION](#) ([MAKE\\_VERSION](#)(2, 0, 1))  
*Version 2.0.1.*

## 19.4 Macro Definition Documentation

### 19.4.1 #define FSL\_GINT\_DRIVER\_VERSION (MAKE\_VERSION(2, 0, 1))

## 19.5 Typedef Documentation

### 19.5.1 typedef void(\* gint\_cb\_t)(void)

## 19.6 Enumeration Type Documentation

### 19.6.1 enum gint\_comb\_t

Enumerator

***kGINT\_CombineOr*** A grouped interrupt is generated when any one of the enabled inputs is active.  
***kGINT\_CombineAnd*** A grouped interrupt is generated when all enabled inputs are active.

### 19.6.2 enum gint\_trig\_t

Enumerator

***kGINT\_TrigEdge*** Edge triggered based on polarity.  
***kGINT\_TrigLevel*** Level triggered based on polarity.

## 19.7 Function Documentation

### 19.7.1 void GINT\_Init ( GINT\_Type \* *base* )

This function initializes the GINT peripheral and enables the clock.

## Function Documentation

### Parameters

|             |                                      |
|-------------|--------------------------------------|
| <i>base</i> | Base address of the GINT peripheral. |
|-------------|--------------------------------------|

### Return values

|              |  |
|--------------|--|
| <i>None.</i> |  |
|--------------|--|

### 19.7.2 void GINT\_SetCtrl ( GINT\_Type \* *base*, gint\_comb\_t *comb*, gint\_trig\_t *trig*, gint\_cb\_t *callback* )

This function sets the control parameters of GINT peripheral.

### Parameters

|                 |                                                                                      |
|-----------------|--------------------------------------------------------------------------------------|
| <i>base</i>     | Base address of the GINT peripheral.                                                 |
| <i>comb</i>     | Controls if the enabled inputs are logically ORed or ANDed for interrupt generation. |
| <i>trig</i>     | Controls if the enabled inputs are level or edge sensitive based on polarity.        |
| <i>callback</i> | This function is called when configured group interrupt is generated.                |

### Return values

|              |  |
|--------------|--|
| <i>None.</i> |  |
|--------------|--|

### 19.7.3 void GINT\_GetCtrl ( GINT\_Type \* *base*, gint\_comb\_t \* *comb*, gint\_trig\_t \* *trig*, gint\_cb\_t \* *callback* )

This function returns the control parameters of GINT peripheral.

### Parameters

|             |                                       |
|-------------|---------------------------------------|
| <i>base</i> | Base address of the GINT peripheral.  |
| <i>comb</i> | Pointer to store combine input value. |
| <i>trig</i> | Pointer to store trigger value.       |

|                 |                                     |
|-----------------|-------------------------------------|
| <i>callback</i> | Pointer to store callback function. |
|-----------------|-------------------------------------|

Return values

|              |  |
|--------------|--|
| <i>None.</i> |  |
|--------------|--|

#### 19.7.4 void GINT\_ConfigPins ( GINT\_Type \* *base*, gint\_port\_t *port*, uint32\_t *polarityMask*, uint32\_t *enableMask* )

This function enables and controls the polarity of enabled pin(s) of a given port.

Parameters

|                     |                                                                                                                                 |
|---------------------|---------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>         | Base address of the GINT peripheral.                                                                                            |
| <i>port</i>         | Port number.                                                                                                                    |
| <i>polarityMask</i> | Each bit position selects the polarity of the corresponding enabled pin. 0 = The pin is active LOW. 1 = The pin is active HIGH. |
| <i>enableMask</i>   | Each bit position selects if the corresponding pin is enabled or not. 0 = The pin is disabled. 1 = The pin is enabled.          |

Return values

|              |  |
|--------------|--|
| <i>None.</i> |  |
|--------------|--|

#### 19.7.5 void GINT\_GetConfigPins ( GINT\_Type \* *base*, gint\_port\_t *port*, uint32\_t \* *polarityMask*, uint32\_t \* *enableMask* )

This function returns the pin configuration of a given port.

Parameters

|                     |                                                                                                                                                                      |
|---------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>         | Base address of the GINT peripheral.                                                                                                                                 |
| <i>port</i>         | Port number.                                                                                                                                                         |
| <i>polarityMask</i> | Pointer to store the polarity mask Each bit position indicates the polarity of the corresponding enabled pin. 0 = The pin is active LOW. 1 = The pin is active HIGH. |

## Function Documentation

|                   |                                                                                                                                                            |
|-------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>enableMask</i> | Pointer to store the enable mask. Each bit position indicates if the corresponding pin is enabled or not. 0 = The pin is disabled. 1 = The pin is enabled. |
|-------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------|

Return values

|              |  |
|--------------|--|
| <i>None.</i> |  |
|--------------|--|

### 19.7.6 void GINT\_EnableCallback ( GINT\_Type \* *base* )

This function enables the interrupt for the selected GINT peripheral. Although the pin(s) are monitored as soon as they are enabled, the callback function is not enabled until this function is called.

Parameters

|             |                                      |
|-------------|--------------------------------------|
| <i>base</i> | Base address of the GINT peripheral. |
|-------------|--------------------------------------|

Return values

|              |  |
|--------------|--|
| <i>None.</i> |  |
|--------------|--|

### 19.7.7 void GINT\_DisableCallback ( GINT\_Type \* *base* )

This function disables the interrupt for the selected GINT peripheral. Although the pins are still being monitored but the callback function is not called.

Parameters

|             |                                 |
|-------------|---------------------------------|
| <i>base</i> | Base address of the peripheral. |
|-------------|---------------------------------|

Return values

|              |  |
|--------------|--|
| <i>None.</i> |  |
|--------------|--|

### 19.7.8 static void GINT\_ClrStatus ( GINT\_Type \* *base* ) [inline], [static]

This function clears the GINT status bit.



## Parameters

|             |                                      |
|-------------|--------------------------------------|
| <i>base</i> | Base address of the GINT peripheral. |
|-------------|--------------------------------------|

## Return values

|              |  |
|--------------|--|
| <i>None.</i> |  |
|--------------|--|

### 19.7.9 static uint32\_t GINT\_GetStatus ( GINT\_Type \* *base* ) [inline], [static]

This function returns the GINT status.

## Parameters

|             |                                      |
|-------------|--------------------------------------|
| <i>base</i> | Base address of the GINT peripheral. |
|-------------|--------------------------------------|

## Return values

|               |                                                                     |
|---------------|---------------------------------------------------------------------|
| <i>status</i> | = 0 No group interrupt request. = 1 Group interrupt request active. |
|---------------|---------------------------------------------------------------------|

### 19.7.10 void GINT\_Deinit ( GINT\_Type \* *base* )

This function disables the GINT clock.

## Parameters

|             |                                      |
|-------------|--------------------------------------|
| <i>base</i> | Base address of the GINT peripheral. |
|-------------|--------------------------------------|

## Return values

|              |  |
|--------------|--|
| <i>None.</i> |  |
|--------------|--|



## Chapter 20

# GPIO: General Purpose I/O

### 20.1 Overview

The MCUXpresso SDK provides a peripheral driver for the General Purpose I/O (GPIO) module of MCUXpresso SDK devices.

### 20.2 Function groups

#### 20.2.1 Initialization and deinitialization

The function [GPIO\\_PinInit\(\)](#) initializes the GPIO with specified configuration.

#### 20.2.2 Pin manipulation

The function [GPIO\\_PinWrite\(\)](#) set output state of selected GPIO pin. The function [GPIO\\_PinRead\(\)](#) read input value of selected GPIO pin.

#### 20.2.3 Port manipulation

The function [GPIO\\_PortSet\(\)](#) sets the output level of selected GPIO pins to the logic 1. The function [GPIO\\_PortClear\(\)](#) sets the output level of selected GPIO pins to the logic 0. The function [GPIO\\_PortToggle\(\)](#) reverse the output level of selected GPIO pins. The function [GPIO\\_PortRead\(\)](#) read input value of selected port.

#### 20.2.4 Port masking

The function [GPIO\\_PortMaskedSet\(\)](#) set port mask, only pins masked by 0 will be enabled in following functions. The function [GPIO\\_PortMaskedWrite\(\)](#) sets the state of selected GPIO port, only pins masked by 0 will be affected. The function [GPIO\\_PortMaskedRead\(\)](#) reads the state of selected GPIO port, only pins masked by 0 are enabled for read, pins masked by 1 are read as 0.

### 20.3 Typical use case

Example use of GPIO API. Refer to the driver examples codes located at `<SDK_ROOT>/boards/<BOARD>/driver_examples/gpio`

## Typical use case

## Files

- file [fsl\\_gpio.h](#)

## Data Structures

- struct [gpio\\_pin\\_config\\_t](#)  
*The GPIO pin configuration structure. [More...](#)*

## Enumerations

- enum [gpio\\_pin\\_direction\\_t](#) {  
    [kGPIO\\_DigitalInput](#) = 0U,  
    [kGPIO\\_DigitalOutput](#) = 1U }  
*LPC GPIO direction definition.*

## Functions

- static void [GPIO\\_PortSet](#) (GPIO\_Type \*base, uint32\_t port, uint32\_t mask)  
*Sets the output level of the multiple GPIO pins to the logic 1.*
- static void [GPIO\\_PortClear](#) (GPIO\_Type \*base, uint32\_t port, uint32\_t mask)  
*Sets the output level of the multiple GPIO pins to the logic 0.*
- static void [GPIO\\_PortToggle](#) (GPIO\_Type \*base, uint32\_t port, uint32\_t mask)  
*Reverses current output logic of the multiple GPIO pins.*

## Driver version

- #define [FSL\\_GPIO\\_DRIVER\\_VERSION](#) ([MAKE\\_VERSION](#)(2, 1, 4))  
*LPC GPIO driver version 2.1.3.*

## GPIO Configuration

- void [GPIO\\_PortInit](#) (GPIO\_Type \*base, uint32\_t port)  
*Initializes the GPIO peripheral.*
- void [GPIO\\_PinInit](#) (GPIO\_Type \*base, uint32\_t port, uint32\_t pin, const [gpio\\_pin\\_config\\_t](#) \*config)  
*Initializes a GPIO pin used by the board.*

## GPIO Output Operations

- static void [GPIO\\_PinWrite](#) (GPIO\_Type \*base, uint32\_t port, uint32\_t pin, uint8\_t output)  
*Sets the output level of the one GPIO pin to the logic 1 or 0.*

## GPIO Input Operations

- static uint32\_t [GPIO\\_PinRead](#) (GPIO\_Type \*base, uint32\_t port, uint32\_t pin)  
*Reads the current input value of the GPIO PIN.*

## 20.4 Data Structure Documentation

### 20.4.1 struct gpio\_pin\_config\_t

Every pin can only be configured as either output pin or input pin at a time. If configured as a input pin, then leave the outputConfig unused.

#### Data Fields

- [gpio\\_pin\\_direction\\_t pinDirection](#)  
*GPIO direction, input or output.*
- uint8\_t [outputLogic](#)  
*Set default output logic, no use in input.*

## 20.5 Macro Definition Documentation

### 20.5.1 #define FSL\_GPIO\_DRIVER\_VERSION (MAKE\_VERSION(2, 1, 4))

## 20.6 Enumeration Type Documentation

### 20.6.1 enum gpio\_pin\_direction\_t

Enumerator

*kGPIO\_DigitalInput* Set current pin as digital input.  
*kGPIO\_DigitalOutput* Set current pin as digital output.

## 20.7 Function Documentation

### 20.7.1 void GPIO\_PortInit ( GPIO\_Type \* *base*, uint32\_t *port* )

This function ungates the GPIO clock.

Parameters

|             |                               |
|-------------|-------------------------------|
| <i>base</i> | GPIO peripheral base pointer. |
| <i>port</i> | GPIO port number.             |

### 20.7.2 void GPIO\_PinInit ( GPIO\_Type \* *base*, uint32\_t *port*, uint32\_t *pin*, const gpio\_pin\_config\_t \* *config* )

To initialize the GPIO, define a pin configuration, either input or output, in the user file. Then, call the [GPIO\\_PinInit\(\)](#) function.

This is an example to define an input pin or output pin configuration:

## Function Documentation

```
* // Define a digital input pin configuration,
* gpio_pin_config_t config =
* {
* kGPIO_DigitalInput,
* 0,
* }
* //Define a digital output pin configuration,
* gpio_pin_config_t config =
* {
* kGPIO_DigitalOutput,
* 0,
* }
*
```

### Parameters

|               |                                              |
|---------------|----------------------------------------------|
| <i>base</i>   | GPIO peripheral base pointer(Typically GPIO) |
| <i>port</i>   | GPIO port number                             |
| <i>pin</i>    | GPIO pin number                              |
| <i>config</i> | GPIO pin configuration pointer               |

### 20.7.3 static void GPIO\_PinWrite ( GPIO\_Type \* *base*, uint32\_t *port*, uint32\_t *pin*, uint8\_t *output* ) [inline], [static]

#### Parameters

|               |                                                                                                                                                                                     |
|---------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>   | GPIO peripheral base pointer(Typically GPIO)                                                                                                                                        |
| <i>port</i>   | GPIO port number                                                                                                                                                                    |
| <i>pin</i>    | GPIO pin number                                                                                                                                                                     |
| <i>output</i> | GPIO pin output logic level. <ul style="list-style-type: none"><li>• 0: corresponding pin output low-logic level.</li><li>• 1: corresponding pin output high-logic level.</li></ul> |

### 20.7.4 static uint32\_t GPIO\_PinRead ( GPIO\_Type \* *base*, uint32\_t *port*, uint32\_t *pin* ) [inline], [static]

#### Parameters

|             |                                              |
|-------------|----------------------------------------------|
| <i>base</i> | GPIO peripheral base pointer(Typically GPIO) |
| <i>port</i> | GPIO port number                             |
| <i>pin</i>  | GPIO pin number                              |

Return values

|             |                                                                                                                                                                          |
|-------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>GPIO</i> | port input value <ul style="list-style-type: none"> <li>• 0: corresponding pin input low-logic level.</li> <li>• 1: corresponding pin input high-logic level.</li> </ul> |
|-------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

**20.7.5 static void GPIO\_PortSet ( GPIO\_Type \* *base*, uint32\_t *port*, uint32\_t *mask* ) [inline], [static]**

Parameters

|             |                                              |
|-------------|----------------------------------------------|
| <i>base</i> | GPIO peripheral base pointer(Typically GPIO) |
| <i>port</i> | GPIO port number                             |
| <i>mask</i> | GPIO pin number macro                        |

**20.7.6 static void GPIO\_PortClear ( GPIO\_Type \* *base*, uint32\_t *port*, uint32\_t *mask* ) [inline], [static]**

Parameters

|             |                                              |
|-------------|----------------------------------------------|
| <i>base</i> | GPIO peripheral base pointer(Typically GPIO) |
| <i>port</i> | GPIO port number                             |
| <i>mask</i> | GPIO pin number macro                        |

**20.7.7 static void GPIO\_PortToggle ( GPIO\_Type \* *base*, uint32\_t *port*, uint32\_t *mask* ) [inline], [static]**

## Function Documentation

### Parameters

|             |                                              |
|-------------|----------------------------------------------|
| <i>base</i> | GPIO peripheral base pointer(Typically GPIO) |
| <i>port</i> | GPIO port number                             |
| <i>mask</i> | GPIO pin number macro                        |



## Chapter 21

# INPUTMUX: Input Multiplexing Driver

### 21.1 Overview

The MCUXpresso SDK provides a driver for the Input multiplexing (INPUTMUX).

It configures the inputs to the pin interrupt block, DMA trigger, and frequency measure function. Once configured, the clock is not needed for the inputmux.

### 21.2 Input Multiplexing Driver operation

INPUTMUX\_AttachSignal function configures the specified input

### 21.3 Typical use case

Refer to the driver examples codes located at <SDK\_ROOT>/boards/<BOARD>/driver\_examples/inputmux

#### Files

- file [fsl\\_inputmux.h](#)
- file [fsl\\_inputmux\\_connections.h](#)

#### Macros

- #define [PINTSEL\\_PMUX\\_ID](#) (offsetof(INPUTMUX\_Type, PINTSEL))  
*Periphinmux IDs.*
- #define [DMA\\_ITRIG\\_PMUX\\_ID](#) (offsetof(INPUTMUX\_Type, DMA\_ITRIG\_INMUX))  
*0xE0U*
- #define [DMA\\_OTRIG\\_PMUX\\_ID](#) (offsetof(INPUTMUX\_Type, DMA\_OTRIG\_INMUX))  
*0x160U*
- #define [FREQMEAS\\_PMUX\\_ID](#) (offsetof(INPUTMUX\_Type, FREQMEAS\_REF))  
*0x180U*
- #define [PMUX\\_SHIFT](#) 20U  
*20U*

#### Enumerations

- enum [inputmux\\_connection\\_t](#) {  
[kINPUTMUX\\_ClkInToFreqmeas](#) = 0U + (FREQMEAS\_PMUX\_ID << PMUX\_SHIFT),  
[kINPUTMUX\\_Xtal32MhzToFreqmeas](#) = 1U + (FREQMEAS\_PMUX\_ID << PMUX\_SHIFT),  
[kINPUTMUX\\_Fro1MhzToFreqmeas](#) = 2U + (FREQMEAS\_PMUX\_ID << PMUX\_SHIFT),  
[kINPUTMUX\\_32KhzOscToFreqmeas](#) = 3U + (FREQMEAS\_PMUX\_ID << PMUX\_SHIFT),  
[kINPUTMUX\\_MainClkToFreqmeas](#) = 4U + (FREQMEAS\_PMUX\_ID << PMUX\_SHIFT),  
[kINPUTMUX\\_GpioPort0Pin4ToFreqmeas](#) = 5U + (FREQMEAS\_PMUX\_ID << PMUX\_SHIFT),  
[kINPUTMUX\\_GpioPort0Pin20ToFreqmeas](#) = 6U + (FREQMEAS\_PMUX\_ID << PMUX\_SHIF-

## Typical use case

T),

`kINPUTMUX_GpioPort0Pin16ToFreqmeas` = 7U + (FREQMEAS\_PMUX\_ID << PMUX\_SHIF-

T),

`kINPUTMUX_GpioPort0Pin15ToFreqmeas` = 8U + (FREQMEAS\_PMUX\_ID << PMUX\_SHIF-

T),

```

kINPUTMUX_GpioPort0Pin0ToPintsel = INPUTMUX_GpioPortPinToPintsel(0,0U),
kINPUTMUX_GpioPort0Pin1ToPintsel = INPUTMUX_GpioPortPinToPintsel(0,1U),
kINPUTMUX_GpioPort0Pin2ToPintsel = INPUTMUX_GpioPortPinToPintsel(0,2U),
kINPUTMUX_GpioPort0Pin3ToPintsel = INPUTMUX_GpioPortPinToPintsel(0,3U),
kINPUTMUX_GpioPort0Pin4ToPintsel = INPUTMUX_GpioPortPinToPintsel(0,4U),
kINPUTMUX_GpioPort0Pin5ToPintsel = INPUTMUX_GpioPortPinToPintsel(0,5U),
kINPUTMUX_GpioPort0Pin6ToPintsel = INPUTMUX_GpioPortPinToPintsel(0,6U),
kINPUTMUX_GpioPort0Pin7ToPintsel = INPUTMUX_GpioPortPinToPintsel(0,7U),
kINPUTMUX_GpioPort0Pin8ToPintsel = INPUTMUX_GpioPortPinToPintsel(0,8U),
kINPUTMUX_GpioPort0Pin9ToPintsel = INPUTMUX_GpioPortPinToPintsel(0,9U),
kINPUTMUX_GpioPort0Pin10ToPintsel = INPUTMUX_GpioPortPinToPintsel(0,10U),
kINPUTMUX_GpioPort0Pin11ToPintsel = INPUTMUX_GpioPortPinToPintsel(0,11U),
kINPUTMUX_GpioPort0Pin12ToPintsel = INPUTMUX_GpioPortPinToPintsel(0,12U),
kINPUTMUX_GpioPort0Pin13ToPintsel = INPUTMUX_GpioPortPinToPintsel(0,13U),
kINPUTMUX_GpioPort0Pin14ToPintsel = INPUTMUX_GpioPortPinToPintsel(0,14U),
kINPUTMUX_GpioPort0Pin15ToPintsel = INPUTMUX_GpioPortPinToPintsel(0,15U),
kINPUTMUX_GpioPort0Pin16ToPintsel = INPUTMUX_GpioPortPinToPintsel(0,16U),
kINPUTMUX_GpioPort0Pin17ToPintsel = INPUTMUX_GpioPortPinToPintsel(0,17U),
kINPUTMUX_GpioPort0Pin18ToPintsel = INPUTMUX_GpioPortPinToPintsel(0,18U),
kINPUTMUX_GpioPort0Pin19ToPintsel = INPUTMUX_GpioPortPinToPintsel(0,19U),
kINPUTMUX_GpioPort0Pin20ToPintsel = INPUTMUX_GpioPortPinToPintsel(0,20U),
kINPUTMUX_GpioPort0Pin21ToPintsel = INPUTMUX_GpioPortPinToPintsel(0,21U),
kINPUTMUX_Adc0SeqaIrqToDma = 0U + (DMA_ITRIG_PMUX_ID << PMUX_SHIFT),
kINPUTMUX_Adc0SeqbIrqToDma = 1U + (DMA_ITRIG_PMUX_ID << PMUX_SHIFT),
kINPUTMUX_Ctimer0M0ToDma = 2U + (DMA_ITRIG_PMUX_ID << PMUX_SHIFT),
kINPUTMUX_Ctimer0M1ToDma = 3U + (DMA_ITRIG_PMUX_ID << PMUX_SHIFT),
kINPUTMUX_Ctimer1M0ToDma = 4U + (DMA_ITRIG_PMUX_ID << PMUX_SHIFT),
kINPUTMUX_Ctimer1M1ToDma = 5U + (DMA_ITRIG_PMUX_ID << PMUX_SHIFT),
kINPUTMUX_PinInt0ToDma = 6U + (DMA_ITRIG_PMUX_ID << PMUX_SHIFT),
kINPUTMUX_PinInt1ToDma = 7U + (DMA_ITRIG_PMUX_ID << PMUX_SHIFT),
kINPUTMUX_PinInt2ToDma = 8U + (DMA_ITRIG_PMUX_ID << PMUX_SHIFT),
kINPUTMUX_PinInt3ToDma = 9U + (DMA_ITRIG_PMUX_ID << PMUX_SHIFT),
kINPUTMUX_AesRxToDma = 10U + (DMA_ITRIG_PMUX_ID << PMUX_SHIFT),
kINPUTMUX_AesTxToDma = 11U + (DMA_ITRIG_PMUX_ID << PMUX_SHIFT),
kINPUTMUX_HashRxToDma = 12U + (DMA_ITRIG_PMUX_ID << PMUX_SHIFT),
kINPUTMUX_HashTxToDma = 13U + (DMA_ITRIG_PMUX_ID << PMUX_SHIFT),
kINPUTMUX_Otrig0ToDma = 14U + (DMA_ITRIG_PMUX_ID << PMUX_SHIFT),
kINPUTMUX_Otrig1ToDma = 15U + (DMA_ITRIG_PMUX_ID << PMUX_SHIFT),
kINPUTMUX_Otrig2ToDma = 16U + (DMA_ITRIG_PMUX_ID << PMUX_SHIFT),
kINPUTMUX_Otrig3ToDma = 17U + (DMA_ITRIG_PMUX_ID << PMUX_SHIFT),
kINPUTMUX_DmaUsart0RxTrigoutToTriginChannels = 0U + (DMA_OTRIG_PMUX_ID << P-
MUX_SHIFT),
kINPUTMUX_DmaUsart0TxTrigoutToTriginChannels = 1U + (DMA_OTRIG_PMUX_ID << P-

```

## Typical use case

```
MUX_SHIFT),
kINPUTMUX_DmaUsart1RxTrigoutToTriginChannels = 2U + (DMA_OTRIG_PMUX_ID << P-
MUX_SHIFT),
kINPUTMUX_DmaUsart1TxTrigoutToTriginChannels = 3U + (DMA_OTRIG_PMUX_ID << P-
MUX_SHIFT),
kINPUTMUX_DmaI2c0SlaveTrigoutToTriginChannels = 4U + (DMA_OTRIG_PMUX_ID << P-
MUX_SHIFT),
kINPUTMUX_DmaI2c0MasterTrigoutToTriginChannels = 5U + (DMA_OTRIG_PMUX_ID <<
PMUX_SHIFT),
kINPUTMUX_DmaI2c1SlaveTrigoutToTriginChannels = 6U + (DMA_OTRIG_PMUX_ID << P-
MUX_SHIFT),
kINPUTMUX_DmaI2c1MasterTrigoutToTriginChannels = 7U + (DMA_OTRIG_PMUX_ID <<
PMUX_SHIFT),
kINPUTMUX_DmaSpi0RxTrigoutToTriginChannels = 8U + (DMA_OTRIG_PMUX_ID << PM-
UX_SHIFT),
kINPUTMUX_DmaSpi0TxTrigoutToTriginChannels = 9U + (DMA_OTRIG_PMUX_ID << PM-
UX_SHIFT),
kINPUTMUX_DmaSpi1RxTrigoutToTriginChannels = 10U + (DMA_OTRIG_PMUX_ID << P-
MUX_SHIFT),
kINPUTMUX_DmaSpi1TxTrigoutToTriginChannels = 11U + (DMA_OTRIG_PMUX_ID << P-
MUX_SHIFT),
kINPUTMUX_DmaSpifi0TrigoutToTriginChannels = 12U + (DMA_OTRIG_PMUX_ID << PM-
UX_SHIFT),
kINPUTMUX_DmaI2c2SlaveTrigoutToTriginChannels = 13U + (DMA_OTRIG_PMUX_ID <<
PMUX_SHIFT),
kINPUTMUX_DmaI2c2MasterTrigoutToTriginChannels = 14U + (DMA_OTRIG_PMUX_ID <<
PMUX_SHIFT),
kINPUTMUX_DmaDmic0Ch0TrigoutToTriginChannels = 15U + (DMA_OTRIG_PMUX_ID <<
PMUX_SHIFT),
kINPUTMUX_DmaDmic0Ch1TrigoutToTriginChannels = 16U + (DMA_OTRIG_PMUX_ID <<
PMUX_SHIFT),
kINPUTMUX_DmaHash0RxTrigoutToTriginChannels = 17U + (DMA_OTRIG_PMUX_ID <<
PMUX_SHIFT),
kINPUTMUX_DmaHash0TxTrigoutToTriginChannels = 18U + (DMA_OTRIG_PMUX_ID << P-
MUX_SHIFT) }
```

*INPUTMUX connections type.*

## Functions

- void **INPUTMUX\_Init** (INPUTMUX\_Type \*base)  
*Initialize INPUTMUX peripheral.*
- void **INPUTMUX\_AttachSignal** (INPUTMUX\_Type \*base, uint32\_t index, [inputmux\\_connection\\_t](#) connection)  
*Attaches a signal.*
- void **INPUTMUX\_Deinit** (INPUTMUX\_Type \*base)  
*Deinitialize INPUTMUX peripheral.*

## Driver version

- #define **FSL\_INPUTMUX\_DRIVER\_VERSION** (MAKE\_VERSION(2, 0, 1))  
Group interrupt driver version for SDK.

## 21.4 Macro Definition Documentation

### 21.4.1 #define PINTSEL\_PMUX\_ID (offsetof(INPUTMUX\_Type, PINTSEL))

0xC0U

### 21.4.2 #define FSL\_INPUTMUX\_DRIVER\_VERSION (MAKE\_VERSION(2, 0, 1))

Version 2.0.1.

## 21.5 Enumeration Type Documentation

### 21.5.1 enum inputmux\_connection\_t

Enumerator

**kINPUTMUX\_ClkInToFreqmeas** Clock Input to Frequency measure.  
**kINPUTMUX\_Xtal32MhzToFreqmeas** XTAL 32MHZ to Frequency measure.  
**kINPUTMUX\_Fro1MhzToFreqmeas** Fro 1MHz to Frequency measure.  
**kINPUTMUX\_32KhzOscToFreqmeas** 32KHz OSC to Frequency measure.  
**kINPUTMUX\_MainClkToFreqmeas** Main Clock to Frequency measure.  
**kINPUTMUX\_GpioPort0Pin4ToFreqmeas** GPIO PORT 0 Pin 4 to Frequency measure.  
**kINPUTMUX\_GpioPort0Pin20ToFreqmeas** GPIO Port 0 Pin 20 to Frequency measure.  
**kINPUTMUX\_GpioPort0Pin16ToFreqmeas** GPIO Port 0 Pin 16 to Frequency measure.  
**kINPUTMUX\_GpioPort0Pin15ToFreqmeas** GPIO Port 0 Pin 15 to Frequency measure. Pin Interrupt.  
**kINPUTMUX\_GpioPort0Pin0ToPintsel** Port 0 Pin 0 to PINT select.  
**kINPUTMUX\_GpioPort0Pin1ToPintsel** Port 0 Pin 1 to PINT select.  
**kINPUTMUX\_GpioPort0Pin2ToPintsel** Port 0 Pin 2 to PINT select.  
**kINPUTMUX\_GpioPort0Pin3ToPintsel** Port 0 Pin 3 to PINT select.  
**kINPUTMUX\_GpioPort0Pin4ToPintsel** Port 0 Pin 4 to PINT select.  
**kINPUTMUX\_GpioPort0Pin5ToPintsel** Port 0 Pin 5 to PINT select.  
**kINPUTMUX\_GpioPort0Pin6ToPintsel** Port 0 Pin 6 to PINT select.  
**kINPUTMUX\_GpioPort0Pin7ToPintsel** Port 0 Pin 7 to PINT select.  
**kINPUTMUX\_GpioPort0Pin8ToPintsel** Port 0 Pin 8 to PINT select.  
**kINPUTMUX\_GpioPort0Pin9ToPintsel** Port 0 Pin 9 to PINT select.  
**kINPUTMUX\_GpioPort0Pin10ToPintsel** Port 0 Pin 10 to PINT select.  
**kINPUTMUX\_GpioPort0Pin11ToPintsel** Port 0 Pin 11 to PINT select.  
**kINPUTMUX\_GpioPort0Pin12ToPintsel** Port 0 Pin 12 to PINT select.  
**kINPUTMUX\_GpioPort0Pin13ToPintsel** Port 0 Pin 13 to PINT select.

*kINPUTMUX\_GpioPort0Pin14ToPintsel* Port 0 Pin 14 to PINT select.  
*kINPUTMUX\_GpioPort0Pin15ToPintsel* Port 0 Pin 15 to PINT select.  
*kINPUTMUX\_GpioPort0Pin16ToPintsel* Port 0 Pin 16 to PINT select.  
*kINPUTMUX\_GpioPort0Pin17ToPintsel* Port 0 Pin 17 to PINT select.  
*kINPUTMUX\_GpioPort0Pin18ToPintsel* Port 0 Pin 18 to PINT select.  
*kINPUTMUX\_GpioPort0Pin19ToPintsel* Port 0 Pin 19 to PINT select.  
*kINPUTMUX\_GpioPort0Pin20ToPintsel* Port 0 Pin 20 to PINT select.  
*kINPUTMUX\_GpioPort0Pin21ToPintsel* Port 0 Pin 21 to PINT select. DMA ITRIG.  
*kINPUTMUX\_Adc0SeqaIrqToDma* ADC Interrupt (Sequence A)  
*kINPUTMUX\_Adc0SeqbIrqToDma* ADC Interrupt (Sequence B)  
*kINPUTMUX\_Ctimer0M0ToDma* Timer CT32B0 Match 0 DMA request.  
*kINPUTMUX\_Ctimer0M1ToDma* Timer CT32B0 Match 1 DMA request.  
*kINPUTMUX\_Ctimer1M0ToDma* Timer CT32B1 Match 0 DMA request.  
*kINPUTMUX\_Ctimer1M1ToDma* Timer CT32B1 Match 1 DMA request.  
*kINPUTMUX\_PinInt0ToDma* Pin interrupt 0.  
*kINPUTMUX\_PinInt1ToDma* Pin interrupt 1.  
*kINPUTMUX\_PinInt2ToDma* Pin interrupt 2.  
*kINPUTMUX\_PinInt3ToDma* Pin interrupt 3.  
*kINPUTMUX\_AesRxToDma* AES RX.  
*kINPUTMUX\_AesTxToDma* AES TX.  
*kINPUTMUX\_HashRxToDma* Hash RX.  
*kINPUTMUX\_HashTxToDma* Hash TX.  
*kINPUTMUX\_Otrig0ToDma* DMA output trigger 0.  
*kINPUTMUX\_Otrig1ToDma* DMA output trigger 1.  
*kINPUTMUX\_Otrig2ToDma* DMA output trigger 2.  
*kINPUTMUX\_Otrig3ToDma* DMA output trigger 3. DMA OTRIG.  
*kINPUTMUX\_DmaUsart0RxTrigoutToTriginChannels* USART 0 RX.  
*kINPUTMUX\_DmaUsart0TxTrigoutToTriginChannels* USART 0 TX.  
*kINPUTMUX\_DmaUsart1RxTrigoutToTriginChannels* USART 1 RX.  
*kINPUTMUX\_DmaUsart1TxTrigoutToTriginChannels* USART 1 TX.  
*kINPUTMUX\_DmaI2c0SlaveTrigoutToTriginChannels* I2C 0 Slave.  
*kINPUTMUX\_DmaI2c0MasterTrigoutToTriginChannels* I2C 0 Master.  
*kINPUTMUX\_DmaI2c1SlaveTrigoutToTriginChannels* I2C 1 Slave.  
*kINPUTMUX\_DmaI2c1MasterTrigoutToTriginChannels* I2C 1 Master.  
*kINPUTMUX\_DmaSpi0RxTrigoutToTriginChannels* SPI 0 RX.  
*kINPUTMUX\_DmaSpi0TxTrigoutToTriginChannels* SPI 0 TX.  
*kINPUTMUX\_DmaSpi1RxTrigoutToTriginChannels* SPI 1 RX.  
*kINPUTMUX\_DmaSpi1TxTrigoutToTriginChannels* SPI 1 TX.  
*kINPUTMUX\_DmaSpifi0TrigoutToTriginChannels* SPIFI.  
*kINPUTMUX\_DmaI2c2SlaveTrigoutToTriginChannels* I2C 2 Slave.  
*kINPUTMUX\_DmaI2c2MasterTrigoutToTriginChannels* I2C 2 Master.  
*kINPUTMUX\_DmaDmic0Ch0TrigoutToTriginChannels* DMIC Channel 0.  
*kINPUTMUX\_DmaDmic0Ch1TrigoutToTriginChannels* DMIC Channel 1.  
*kINPUTMUX\_DmaHash0RxTrigoutToTriginChannels* Hash RX.  
*kINPUTMUX\_DmaHash0TxTrigoutToTriginChannels* Hash TX.

## 21.6 Function Documentation

### 21.6.1 void INPUTMUX\_Init ( INPUTMUX\_Type \* *base* )

This function enables the INPUTMUX clock.

Parameters

|             |                                          |
|-------------|------------------------------------------|
| <i>base</i> | Base address of the INPUTMUX peripheral. |
|-------------|------------------------------------------|

Return values

|              |  |
|--------------|--|
| <i>None.</i> |  |
|--------------|--|

### 21.6.2 void INPUTMUX\_AttachSignal ( INPUTMUX\_Type \* *base*, uint32\_t *index*, inputmux\_connection\_t *connection* )

This function gates the INPUTMUX clock.

Parameters

|                   |                                                 |
|-------------------|-------------------------------------------------|
| <i>base</i>       | Base address of the INPUTMUX peripheral.        |
| <i>index</i>      | Destination peripheral to attach the signal to. |
| <i>connection</i> | Selects connection.                             |

Return values

|              |  |
|--------------|--|
| <i>None.</i> |  |
|--------------|--|

### 21.6.3 void INPUTMUX\_Deinit ( INPUTMUX\_Type \* *base* )

This function disables the INPUTMUX clock.

Parameters

|             |                                          |
|-------------|------------------------------------------|
| <i>base</i> | Base address of the INPUTMUX peripheral. |
|-------------|------------------------------------------|

Return values

---

## Function Documentation

|              |  |
|--------------|--|
| <i>None.</i> |  |
|--------------|--|



## Chapter 22

# IOCON: I/O pin configuration

### 22.1 Overview

The MCUXpresso SDK provides a peripheral driver for the I/O pin configuration (IOCON) module of MCUXpresso SDK devices.

### 22.2 Function groups

#### 22.2.1 Pin mux set

The function `IOCONPinMuxSet()` sets a pinmux for a single pin according to the selected configuration.

#### 22.2.2 Pin mux set

The function `IOCON_SetPinMuxing()` sets a pinmux for group of pins according to the selected configuration.

### 22.3 Typical use case

Example use of IOCON API to selection of GPIO mode.

```
int main(void)
{
 /* enable clock for IOCON */
 CLOCK_EnableClock(kCLOCK_Iocon);

 /* Set pin mux for single pin */
 IOCON_PinMuxSet(IOCON, 0, 29, IOCON_FUNC0 |
 IOCON_GPIO_MODE | IOCON_DIGITAL_EN |
 IOCON_INPFILT_OFF);

 /* Set pin mux for group of pins */
 const iocon_group_t gpio_pins[] = {
 {0, 24, (IOCON_FUNC0 | IOCON_GPIO_MODE |
 IOCON_DIGITAL_EN | IOCON_INPFILT_OFF)},
 {0, 31, (IOCON_FUNC0 | IOCON_GPIO_MODE |
 IOCON_DIGITAL_EN | IOCON_INPFILT_OFF)},
 };

 Chip_IOCON_SetPinMuxing(IOCON, gpio_pins, sizeof(gpio_pins)/sizeof(gpio_pins[0]));
}
```

### Files

- file [fsl\\_iocon.h](#)

## Typical use case

## Data Structures

- struct `iocon_group_t`  
*Array of IOCON pin definitions passed to `IOCON_SetPinMuxing()` must be in this format. [More...](#)*

## Macros

- #define `IOCON_FUNC0` `IOCON_PIO_FUNC(0)`  
*IOCON function and mode selection definitions.*
- #define `IOCON_FUNC1` `IOCON_PIO_FUNC(1)`  
*Selects pin function 1.*
- #define `IOCON_FUNC2` `IOCON_PIO_FUNC(2)`  
*Selects pin function 2.*
- #define `IOCON_FUNC3` `IOCON_PIO_FUNC(3)`  
*Selects pin function 3.*
- #define `IOCON_FUNC4` `IOCON_PIO_FUNC(4)`  
*Selects pin function 4.*
- #define `IOCON_FUNC5` `IOCON_PIO_FUNC(5)`  
*Selects pin function 5.*
- #define `IOCON_FUNC6` `IOCON_PIO_FUNC(6)`  
*Selects pin function 6.*
- #define `IOCON_FUNC7` `IOCON_PIO_FUNC(7)`  
*Selects pin function 7.*
- #define `IOCON_MODE_PULLUP` `IOCON_PIO_MODE(0)`  
*Selects pull-up function.*
- #define `IOCON_MODE_REPEATER` `IOCON_PIO_MODE(1)`  
*Selects pin repeater function.*
- #define `IOCON_MODE_INACT` `IOCON_PIO_MODE(2)`  
*No addition pin function.*
- #define `IOCON_MODE_PULLDOWN` `IOCON_PIO_MODE(3)`  
*Selects pull-down function.*
- #define `IOCON_HYS_EN` `(0x1 << 5)`  
*Enables hysteresis ??*
- #define `IOCON_GPIO_MODE` `IOCON_PIO_SLEW0(1)`  
*GPIO Mode.*
- #define `IOCON_I2C_SLEW` `IOCON_PIO_SLEW0(1)`  
*I2C Slew Rate Control.*
- #define `IOCON_INV_EN` `IOCON_PIO_INVERT(1)`  
*Enables invert function on input.*
- #define `IOCON_ANALOG_EN` `IOCON_PIO_DIGIMODE(0)`  
*Enables analog function by setting 0 to bit 7.*
- #define `IOCON_DIGITAL_EN` `IOCON_PIO_DIGIMODE(1)`  
*Enables digital function by setting 1 to bit 7(default)*
- #define `IOCON_STDI2C_EN` `IOCON_PIO_FILTEROFF(1)`  
*I2C standard mode/fast-mode.*
- #define `IOCON_INPFILT_OFF` `IOCON_PIO_FILTEROFF(1)`  
*Input filter Off for GPIO pins.*
- #define `IOCON_INPFILT_ON` `IOCON_PIO_FILTEROFF(0)`  
*Input filter On for GPIO pins.*
- #define `IOCON_SLEW1_OFF` `IOCON_PIO_SLEW1(0)`  
*Driver Slew Rate Control.*
- #define `IOCON_SLEW1_ON` `IOCON_PIO_SLEW1(1)`

- *Driver Slew Rate Control.*  
• #define [IOCON\\_FASTI2C\\_EN](#) ([IOCON\\_INPFILT\\_ON](#) | [IOCON\\_SLEW1\\_ON](#))  
*I2C Fast-mode Plus and high-speed slave.*
- #define [IOCON\\_OPENDRAIN\\_EN](#) [IOCON\\_PIO\\_OD](#)(1)  
*Enables open-drain function.*
- #define [IOCON\\_S\\_MODE\\_0CLK](#) [IOCON\\_PIO\\_SSEL](#)(0)  
*Bypass input filter.*
- #define [IOCON\\_S\\_MODE\\_1CLK](#) [IOCON\\_PIO\\_SSEL](#)(1)  
*Input pulses shorter than 1 filter clock are rejected.*
- #define [IOCON\\_S\\_MODE\\_2CLK](#) [IOCON\\_PIO\\_SSEL](#)(2)  
*Input pulses shorter than 2 filter clock2 are rejected.*
- #define [IOCON\\_S\\_MODE\\_3CLK](#) [IOCON\\_PIO\\_SSEL](#)(3)  
*Input pulses shorter than 3 filter clock2 are rejected.*

## Functions

- \_\_STATIC\_INLINE void [IOCON\\_PinMuxSet](#) ([IOCON\\_Type](#) \*base, uint8\_t port, uint8\_t pin, uint32\_t modefunc)  
*Sets I/O Control pin mux.*
- \_\_STATIC\_INLINE void [IOCON\\_SetPinMuxing](#) ([IOCON\\_Type](#) \*base, const [iocon\\_group\\_t](#) \*pin-Array, uint32\_t arrayLength)  
*Set all I/O Control pin muxing.*
- \_\_STATIC\_INLINE void [IOCON\\_PullSet](#) ([IOCON\\_Type](#) \*base, uint8\_t port, uint8\_t pin, uint8\_t pull\_select)  
*Sets I/O Control pin mux pull select.*
- \_\_STATIC\_INLINE void [IOCON\\_FuncSet](#) ([IOCON\\_Type](#) \*base, uint8\_t port, uint8\_t pin, uint8\_t func)  
*Sets I/O Control pin mux pull select.*

## Driver version

- #define [LPC\\_IOCON\\_DRIVER\\_VERSION](#) ([MAKE\\_VERSION](#)(2, 0, 0))  
*IOCON driver version 2.0.0.*

## 22.4 Data Structure Documentation

### 22.4.1 struct iocon\_group\_t

## 22.5 Macro Definition Documentation

### 22.5.1 #define LPC\_IOCON\_DRIVER\_VERSION (MAKE\_VERSION(2, 0, 0))

### 22.5.2 #define IOCON\_FUNC0 IOCON\_PIO\_FUNC(0)

Note

See the User Manual for specific modes and functions supported by the various pins. Selects pin function 0

### 22.6 Function Documentation

22.6.1 `__STATIC_INLINE void IOCON_PinMuxSet ( IOCON_Type * base, uint8_t port, uint8_t pin, uint32_t modefunc )`

## Parameters

|                 |                                            |
|-----------------|--------------------------------------------|
| <i>base</i>     | : The base of IOCON peripheral on the chip |
| <i>port</i>     | : GPIO port to mux                         |
| <i>pin</i>      | : GPIO pin to mux                          |
| <i>modefunc</i> | : OR'ed values of type IOCON_*             |

## Returns

Nothing

### 22.6.2 **\_\_STATIC\_INLINE void IOCON\_SetPinMuxing ( IOCON\_Type \* *base*, const iocon\_group\_t \* *pinArray*, uint32\_t *arrayLength* )**

## Parameters

|                    |                                            |
|--------------------|--------------------------------------------|
| <i>base</i>        | : The base of IOCON peripheral on the chip |
| <i>pinArray</i>    | : Pointer to array of pin mux selections   |
| <i>arrayLength</i> | : Number of entries in pinArray            |

## Returns

Nothing

### 22.6.3 **\_\_STATIC\_INLINE void IOCON\_PullSet ( IOCON\_Type \* *base*, uint8\_t *port*, uint8\_t *pin*, uint8\_t *pull\_select* )**

## Parameters

|             |                                            |
|-------------|--------------------------------------------|
| <i>base</i> | : The base of IOCON peripheral on the chip |
| <i>port</i> | : GPIO port to mux                         |
| <i>pin</i>  | : GPIO pin to mux                          |

## Function Documentation

|                    |                                |
|--------------------|--------------------------------|
| <i>pull_select</i> | : OR'ed values of type IOCON_* |
|--------------------|--------------------------------|

Returns

Nothing

### 22.6.4 **\_\_STATIC\_INLINE void IOCON\_FuncSet ( IOCON\_Type \* *base*, uint8\_t *port*, uint8\_t *pin*, uint8\_t *func* )**

Parameters

|             |                                            |
|-------------|--------------------------------------------|
| <i>base</i> | : The base of IOCON peripheral on the chip |
| <i>port</i> | : GPIO port to mux                         |
| <i>pin</i>  | : GPIO pin to mux                          |
| <i>func</i> | : Pinmux function                          |

Returns

Nothing

## Chapter 23

### IRM: Infra-Red Modulator driver

#### 23.1 Overview

The MCUXpresso SDK provides a Infra-Red Modulator driver for MCUXpresso SDK devices.

#### Files

- file [fsl\\_cic\\_irb.h](#)
- file [fsl\\_cic\\_irb\\_private.h](#)

#### Data Structures

- struct [cic\\_irb\\_config\\_t](#)
- struct [cic\\_irb\\_instance\\_data\\_t](#)

#### Enumerations

- enum [cic\\_irb\\_protocols\\_t](#)
- enum [cic\\_irb\\_status\\_t](#)
- enum [cic\\_irb\\_carrier\\_frequency\\_t](#)
- enum [cic\\_irb\\_sirc\\_version\\_t](#)
- enum [cic\\_irb\\_rcmm\\_mode\\_t](#)
- enum [cic\\_irb\\_rcmm\\_signal\\_free\\_time\\_t](#)

#### CIC\_IRB Get Default Configuration

- [cic\\_irb\\_status\\_t](#) [CIC\\_IRB\\_GetDefaultConfig](#) ([cic\\_irb\\_config\\_t](#) \*config)  
*CIC\_IRB\_GetDefaultConfig.*

#### CIC\_IRB Initialization

- [cic\\_irb\\_status\\_t](#) [CIC\\_IRB\\_Init](#) ([CIC\\_IRB\\_Type](#) \*base, [cic\\_irb\\_config\\_t](#) \*config)  
*CIC\_IRB\_Init.*

#### CIC\_IRB Enable

- [cic\\_irb\\_status\\_t](#) [CIC\\_IRB\\_Enable](#) ([CIC\\_IRB\\_Type](#) \*base)  
*CIC\_IRB\_Enable.*

#### CIC\_IRB Send RC5 Packet

- [cic\\_irb\\_status\\_t](#) [CIC\\_IRB\\_SendRC5Packet](#) ([CIC\\_IRB\\_Type](#) \*base, bool toggle, [uint8\\_t](#) address, [uint8\\_t](#) command)  
*CIC\_IRB\_SendRC5Packet.*

### CIC\_IRB Send RC6 Packet

- [cic\\_irb\\_status\\_t](#) [CIC\\_IRB\\_SendRC6Packet](#) (CIC\_IRB\_Type \*base, bool toggle, uint8\_t field, uint8\_t address, uint8\_t command)  
*CIC\_IRB\_SendRC6Packet.*

### CIC\_IRB Send SIRC Packet

- [cic\\_irb\\_status\\_t](#) [CIC\\_IRB\\_SendSIRCPacket](#) (CIC\_IRB\_Type \*base, [cic\\_irb\\_sirc\\_version\\_t](#) version, uint8\_t command, uint8\_t address, uint8\_t extendedBits)  
*CIC\_IRB\_SendSIRCPacket.*

### CIC\_IRB Send RCMM Packet

- [cic\\_irb\\_status\\_t](#) [CIC\\_IRB\\_SendRCMMPacket](#) (CIC\_IRB\_Type \*base, [cic\\_irb\\_rcmm\\_mode\\_t](#) mode, uint8\_t modeBits, uint8\_t address, uint8\_t customerId, uint32\_t data, [cic\\_irb\\_rcmm\\_signal\\_free\\_time\\_t](#) signalFreeTime)  
*CIC\_IRB\_SendRCMMPacket.*

### CIC\_IRB Is Busy

- [cic\\_irb\\_status\\_t](#) [CIC\\_IRB\\_IsBusy](#) (CIC\_IRB\_Type \*base, bool \*isBusy)  
*CIC\_IRB\_IsBusy.*

### CIC\_IRB Disable

- [cic\\_irb\\_status\\_t](#) [CIC\\_IRB\\_Disable](#) (CIC\_IRB\_Type \*base)  
*CIC\_IRB\_Disable.*

### CIC\_IRB Deinitializations

- [cic\\_irb\\_status\\_t](#) [CIC\\_IRB\\_DeInit](#) (CIC\_IRB\_Type \*base)  
*CIC\_IRB\_DeInit.*

### CIC\_IRB Get Instance Data

- [cic\\_irb\\_instance\\_data\\_t](#) \* [CIC\\_IRB\\_GetInstanceData](#) (CIC\_IRB\_Type \*base)  
*CIC\_IRB\_GetInstanceData.*

### CIC\_IRB RC5 Initialise

- void [CIC\\_IRB\\_RC5Initialise](#) (CIC\_IRB\_Type \*base)  
*CIC\_IRB\_RC5Initialise.*

### CIC\_IRB RC6 Initialise

- void [CIC\\_IRB\\_RC6Initialise](#) (CIC\_IRB\_Type \*base)  
*CIC\_IRB\_RC6Initialise.*



## CIC\_IRB SIRC Initialise

- void [CIC\\_IRB\\_SIRCInitialise](#) (CIC\_IRB\_Type \*base)  
*CIC\_IRB\_SIRCInitialise.*

## CIC\_IRB RCMM Initialise

- void [CIC\\_IRB\\_RCMMInitialise](#) (CIC\_IRB\_Type \*base)  
*CIC\_IRB\_RCMMInitialise.*

## CIC\_IRB RC5 Get Instance

- uint8\_t [CIC\\_IRB\\_GetInstance](#) (CIC\_IRB\_Type \*base)  
*CIC\_IRB\_GetInstance.*

## CIC\_IRB Load And Send Fifo

- void [CIC\\_IRB\\_LoadAndSendFifo](#) (CIC\_IRB\_Type \*base)  
*CIC\_IRB\_LoadAndSendFifo.*

## CIC\_IRB RCx Append Envelopes

- void [CIC\\_IRB\\_RCxAppendEnvelopes](#) (uint32\_t bitPattern, uint8\_t bitCount, [cic\\_irb\\_instance\\_data\\_t](#) \*instanceData, uint8\_t \*previousEnvelopeLevel, int8\_t \*envelope, bool Manchester-EncodingIEEE802\_3)  
*CIC\_IRB\_RCxAppendEnvelopes.*

## 23.2 Data Structure Documentation

### 23.2.1 struct cic\_irb\_config\_t

CIC IRB configuration structure

This structure holds the configuration settings for the CIC IRB peripheral. To initialize this structure to reasonable defaults, call the [CIC\\_IRB\\_GetDefaultConfig\(\)](#) function and pass a pointer to the configuration structure instance.

### 23.2.2 struct cic\_irb\_instance\_data\_t

CIC IRB instance data structure

## 23.3 Enumeration Type Documentation

### 23.3.1 enum cic\_irb\_protocols\_t

IR protocol types

## Function Documentation

### 23.3.2 enum cic\_irb\_status\_t

Status code responses from API calls

### 23.3.3 enum cic\_irb\_carrier\_frequency\_t

Status code responses from API calls

### 23.3.4 enum cic\_irb\_sirc\_version\_t

Sony SIRC version types

### 23.3.5 enum cic\_irb\_rcmm\_mode\_t

RC-MM mode types

### 23.3.6 enum cic\_irb\_rcmm\_signal\_free\_time\_t

RC-MM signal free types

## 23.4 Function Documentation

### 23.4.1 cic\_irb\_status\_t CIC\_IRB\_GetDefaultConfig ( cic\_irb\_config\_t \* *config* )

This function get default configuration for IRB

Parameters

|               |                                      |
|---------------|--------------------------------------|
| <i>config</i> | pointer to a configuration structure |
|---------------|--------------------------------------|

Returns

A cic\_irb\_status\_t status code\*

### 23.4.2 cic\_irb\_status\_t CIC\_IRB\_Init ( CIC\_IRB\_Type \* *base*, cic\_irb\_config\_t \* *config* )

Initialize the IRB peripheral. Attaches, configures and enables source and peripheral clocks, resets peripheral and initializes instance data. The peripheral is not enabled after this.

## Parameters

|               |                                      |
|---------------|--------------------------------------|
| <i>base</i>   | CIC_IRB peripheral base address      |
| <i>config</i> | pointer to a configuration structure |

## Returns

A `cic_irb_status_t` status code

### 23.4.3 `cic_irb_status_t` **CIC\_IRB\_Enable** ( `CIC_IRB_Type` \* *base* )

Enable the IRB peripheral doing protocol specific initializations. The interrupts are enabled after this call.

## Parameters

|             |                                 |
|-------------|---------------------------------|
| <i>base</i> | CIC_IRB peripheral base address |
|-------------|---------------------------------|

## Returns

A `cic_irb_status_t` status code

### 23.4.4 `cic_irb_status_t` **CIC\_IRB\_SendRC5Packet** ( `CIC_IRB_Type` \* *base*, `bool` *toggle*, `uint8_t` *address*, `uint8_t` *command* )

Send a RC-5 packet via the IRB peripheral. The peripheral instance must have been initialised and enabled and not busy.

## Parameters

|                |                                          |
|----------------|------------------------------------------|
| <i>base</i>    | CIC_IRB peripheral base address          |
| <i>toggle</i>  | the state of the toggle bit to encode    |
| <i>address</i> | the 5 bit address to go into the message |
| <i>command</i> | the 7 bit command to go into the message |

## Returns

A `cic_irb_status_t` status code

### 23.4.5 `cic_irb_status_t` `CIC_IRB_SendRC6Packet ( CIC_IRB_Type * base, bool toggle, uint8_t field, uint8_t address, uint8_t command )`

Send a RC-5 packet via the IRB peripheral. The peripheral instance must have been initialised and enabled and not busy.

## Parameters

|                |                                          |
|----------------|------------------------------------------|
| <i>base</i>    | CIC_IRB peripheral base address          |
| <i>toggle</i>  | the state of the toggle bit to encode    |
| <i>field</i>   | the 3 bit field to go into the message   |
| <i>address</i> | the 8 bit address to go into the message |
| <i>command</i> | the 8 bit command to go into the message |

## Returns

A `cic_irb_status_t` status code

**23.4.6** `cic_irb_status_t CIC_IRB_SendSIRCPacket ( CIC_IRB_Type * base,  
cic_irb_sirc_version_t version, uint8_t command, uint8_t address, uint8_t  
extendedBits )`

Send a SIRC packet via the IRB peripheral. The peripheral instance must have been initialised and enabled and not busy.

## Parameters

|                     |                                                                                                               |
|---------------------|---------------------------------------------------------------------------------------------------------------|
| <i>base</i>         | CIC_IRB peripheral base address                                                                               |
| <i>version</i>      | the version of the protocol to use, varies the packet length command the 7 bit command to go into the message |
| <i>address</i>      | the 5 or 8 bit address to go into the message                                                                 |
| <i>extendedBits</i> | 8 bits of extra data in 20 bit message                                                                        |

## Returns

A `cic_irb_status_t` status code

**23.4.7** `cic_irb_status_t CIC_IRB_SendRCMMPacket ( CIC_IRB_Type * base,  
cic_irb_rcmm_mode_t mode, uint8_t modeBits, uint8_t address,  
uint8_t customerId, uint32_t data, cic_irb_rcmm_signal_free_time_t  
signalFreeTime )`

Send a RCMM packet via the IRB peripheral. The peripheral instance must have been initialised and enabled and not busy.

## Function Documentation

### Parameters

|                       |                                                                                                                                       |
|-----------------------|---------------------------------------------------------------------------------------------------------------------------------------|
| <i>base</i>           | CIC_IRB peripheral base address The RC-MM message mode. See RC-MM documentation Bits The mode numerical value, varying number of bits |
| <i>address</i>        | RC-MM address data field in some modes                                                                                                |
| <i>customerId</i>     | RC-MM customer if in some modes                                                                                                       |
| <i>data</i>           | the RC-MM data field, varying length depending on mode                                                                                |
| <i>signalFreeTime</i> | Silent period after packet transmission                                                                                               |

### Returns

A `cic_irb_status_t` status code

### 23.4.8 `cic_irb_status_t` **CIC\_IRB\_IsBusy** ( `CIC_IRB_Type` \* *base*, `bool` \* *isBusy* )

Determine if the IRB peripheral instance is in the process of sending the previous message.

### Parameters

|               |                                         |
|---------------|-----------------------------------------|
| <i>base</i>   | CIC_IRB peripheral base address         |
| <i>isBusy</i> | pointer to <code>bool</code> for result |

### Returns

A `cic_irb_status_t` status code

### 23.4.9 `cic_irb_status_t` **CIC\_IRB\_Disable** ( `CIC_IRB_Type` \* *base* )

Disable the IRB peripheral such that it can be enabled again without doing another init. This stops interrupts and any part sent message is abandoned.

### Parameters

|             |                                 |
|-------------|---------------------------------|
| <i>base</i> | CIC_IRB peripheral base address |
|-------------|---------------------------------|

### Returns

A `cic_irb_status_t` status code

### 23.4.10 `cic_irb_status_t` **CIC\_IRB\_DeInit** ( `CIC_IRB_Type` \* *base* )

De-initialises the IRB peripheral instance. The clock is stopped.

## Parameters

|             |                                 |
|-------------|---------------------------------|
| <i>base</i> | CIC_IRB peripheral base address |
|-------------|---------------------------------|

## Returns

A `cic_irb_status_t` status code

### 23.4.11 `cic_irb_instance_data_t*` **CIC\_IRB\_GetInstanceData ( `CIC_IRB_Type` \* *base* )**

Get a pointer to the data structure containing context information for a peripheral instance

## Parameters

|             |                                  |
|-------------|----------------------------------|
| <i>base</i> | CIC_IRB peripheral base address. |
|-------------|----------------------------------|

## Returns

Pointer to the instance data structure

NOTES: Returns NULL if an instance not found for base

### 23.4.12 `void` **CIC\_IRB\_RC5Initialise ( `CIC_IRB_Type` \* *base* )**

Initialise the peripheral ready to send RC5 messages.

## Parameters

|             |                                  |
|-------------|----------------------------------|
| <i>base</i> | CIC_IRB peripheral base address. |
|-------------|----------------------------------|

### 23.4.13 `void` **CIC\_IRB\_RC6Initialise ( `CIC_IRB_Type` \* *base* )**

Initialise the peripheral ready to send RC6 messages.

## Parameters

## Function Documentation

|             |                                  |
|-------------|----------------------------------|
| <i>base</i> | CIC_IRB peripheral base address. |
|-------------|----------------------------------|

### 23.4.14 void CIC\_IRB\_SIRCInitialise ( CIC\_IRB\_Type \* *base* )

Initialise the peripheral ready to send SIRC messages.

Parameters

|             |                                  |
|-------------|----------------------------------|
| <i>base</i> | CIC_IRB peripheral base address. |
|-------------|----------------------------------|

### 23.4.15 void CIC\_IRB\_RCMMInitialise ( CIC\_IRB\_Type \* *base* )

Initialise the peripheral ready to send RCMM messages.

Parameters

|             |                                  |
|-------------|----------------------------------|
| <i>base</i> | CIC_IRB peripheral base address. |
|-------------|----------------------------------|

### 23.4.16 uint8\_t CIC\_IRB\_GetInstance ( CIC\_IRB\_Type \* *base* )

Get the CIC\_IRB instance from peripheral base address.

Parameters

|             |                                  |
|-------------|----------------------------------|
| <i>base</i> | CIC_IRB peripheral base address. |
|-------------|----------------------------------|

Returns

CIC\_IRB instance number.

NOTES: Returns FSL\_FEATURE\_SOC\_CIC\_IRB\_COUNT if instance for base not found.

### 23.4.17 void CIC\_IRB\_LoadAndSendFifo ( CIC\_IRB\_Type \* *base* )

Load the FIFO with the first part of a series of envelopes to send. If a message's complete set of envelopes fits within the FIFO size then the whole message will be sent. If not then the maximum number of envelopes is loaded into the FIFO that is possible. Subsequent FIFO loads for the remaining envelopes are loaded into the FIFO in the interrupt handler which is triggered when the FIFO becomes empty.



## Parameters

|             |                                  |
|-------------|----------------------------------|
| <i>base</i> | CIC_IRB peripheral base address. |
|-------------|----------------------------------|

**23.4.18 void CIC\_IRB\_RCxAppendEnvelopes ( uint32\_t *bitPattern*, uint8\_t *bitCount*, cic\_irb\_instance\_data\_t \* *instanceData*, uint8\_t \* *previous-EnvelopeLevel*, int8\_t \* *envelope*, bool *ManchesterEncodingIEEE802\_3* )**

Take a bit pattern that comprises part of a RCx message and encode this in envelopes using the appropriate type of Manchester encoding. The envelopes can be appended to an array of previously encoded envelopes or can be a complete message.

## Parameters

|                                       |                                                                             |
|---------------------------------------|-----------------------------------------------------------------------------|
| <i>bitPattern</i>                     | the binary data to encode into the message                                  |
| <i>bitCount</i>                       | the number of bits in bitPattern                                            |
| <i>instanceData</i>                   | pointer to the peripheral instance data structure                           |
| <i>previous-EnvelopeLevel</i>         | the level of the last envelope of the preceding part of the encoded message |
| <i>envelope</i>                       | pointer to array to add this bit pattern's encoded envelopes                |
| <i>Manchester-EncodingIEEE-E802_3</i> | type of Manchester encoding, true for IEEE 802.3, false for Thomas          |



## Chapter 24

# PINT: Pin Interrupt and Pattern Match Driver

### 24.1 Overview

The MCUXpresso SDK provides a driver for the Pin Interrupt and Pattern match (PINT).

It can configure one or more pins to generate a pin interrupt when the pin or pattern match conditions are met. The pins do not have to be configured as gpio pins however they must be connected to PINT via INPUTMUX. Only the pin interrupt or pattern match function can be active for interrupt generation. If the pin interrupt function is enabled then the pattern match function can be used for wakeup via RXEV.

### 24.2 Pin Interrupt and Pattern match Driver operation

[PINT\\_PinInterruptConfig\(\)](#) function configures the pins for pin interrupt.

[PINT\\_PatternMatchConfig\(\)](#) function configures the pins for pattern match.

#### 24.2.1 Pin Interrupt use case

Refer to the driver examples codes located at `<SDK_ROOT>/boards/<BOARD>/driver_examples/pint`

#### 24.2.2 Pattern match use case

Refer to the driver examples codes located at `<SDK_ROOT>/boards/<BOARD>/driver_examples/pint`

### Files

- file [fsl\\_pint.h](#)

### Typedefs

- typedef void(\* [pint\\_cb\\_t](#))([pint\\_pin\\_int\\_t](#) pintr, uint32\_t pmatch\_status)  
*PINT Callback function.*

### Enumerations

- enum [pint\\_pin\\_enable\\_t](#) {  
    [kPINT\\_PinIntEnableNone](#) = 0U,  
    [kPINT\\_PinIntEnableRiseEdge](#) = PINT\_PIN\_RISE\_EDGE,  
    [kPINT\\_PinIntEnableFallEdge](#) = PINT\_PIN\_FALL\_EDGE,  
    [kPINT\\_PinIntEnableBothEdges](#) = PINT\_PIN\_BOTH\_EDGE,  
    [kPINT\\_PinIntEnableLowLevel](#) = PINT\_PIN\_LOW\_LEVEL,  
    [kPINT\\_PinIntEnableHighLevel](#) = PINT\_PIN\_HIGH\_LEVEL }

## Pin Interrupt and Pattern match Driver operation

- PINT Pin Interrupt enable type.*
  - enum `pint_pin_int_t` {  
    `kPINT_PinInt0` = 0U,  
    `kPINT_PinInt1` = 1U,  
    `kPINT_PinInt2` = 2U,  
    `kPINT_PinInt3` = 3U }
- PINT Pin Interrupt type.*
  - enum `pint_pmatch_input_src_t` {  
    `kPINT_PatternMatchInp0Src` = 0U,  
    `kPINT_PatternMatchInp1Src` = 1U,  
    `kPINT_PatternMatchInp2Src` = 2U,  
    `kPINT_PatternMatchInp3Src` = 3U,  
    `kPINT_PatternMatchInp4Src` = 4U,  
    `kPINT_PatternMatchInp5Src` = 5U,  
    `kPINT_PatternMatchInp6Src` = 6U,  
    `kPINT_PatternMatchInp7Src` = 7U }
- PINT Pattern Match bit slice input source type.*
  - enum `pint_pmatch_bslicet_t` {  
    `kPINT_PatternMatchBSlice0` = 0U,  
    `kPINT_PatternMatchBSlice1` = 1U,  
    `kPINT_PatternMatchBSlice2` = 2U,  
    `kPINT_PatternMatchBSlice3` = 3U }
- PINT Pattern Match bit slice type.*
  - enum `pint_pmatch_bslicecfg_t` {  
    `kPINT_PatternMatchAlways` = 0U,  
    `kPINT_PatternMatchStickyRise` = 1U,  
    `kPINT_PatternMatchStickyFall` = 2U,  
    `kPINT_PatternMatchStickyBothEdges` = 3U,  
    `kPINT_PatternMatchHigh` = 4U,  
    `kPINT_PatternMatchLow` = 5U,  
    `kPINT_PatternMatchNever` = 6U,  
    `kPINT_PatternMatchBothEdges` = 7U }
- PINT Pattern Match configuration type.*

## Functions

- void `PINT_Init` (`PINT_Type` \*base)  
*Initialize PINT peripheral.*
- void `PINT_PinInterruptConfig` (`PINT_Type` \*base, `pint_pin_int_t` intr, `pint_pin_enable_t` enable, `pint_cb_t` callback)  
*Configure PINT peripheral pin interrupt.*
- void `PINT_PinInterruptGetConfig` (`PINT_Type` \*base, `pint_pin_int_t` pintr, `pint_pin_enable_t` \*enable, `pint_cb_t` \*callback)  
*Get PINT peripheral pin interrupt configuration.*
- void `PINT_PinInterruptClrStatus` (`PINT_Type` \*base, `pint_pin_int_t` pintr)  
*Clear Selected pin interrupt status only when the pin was triggered by edge-sensitive.*
- static uint32\_t `PINT_PinInterruptGetStatus` (`PINT_Type` \*base, `pint_pin_int_t` pintr)

- Get Selected pin interrupt status.*
- void [PINT\\_PinInterruptClrStatusAll](#) (PINT\_Type \*base)
- Clear all pin interrupts status only when pins were triggered by edge-sensitive.*
- static uint32\_t [PINT\\_PinInterruptGetStatusAll](#) (PINT\_Type \*base)
- Get all pin interrupts status.*
- static void [PINT\\_PinInterruptClrFallFlag](#) (PINT\_Type \*base, [pint\\_pin\\_int\\_t](#) pintr)
- Clear Selected pin interrupt fall flag.*
- static uint32\_t [PINT\\_PinInterruptGetFallFlag](#) (PINT\_Type \*base, [pint\\_pin\\_int\\_t](#) pintr)
- Get selected pin interrupt fall flag.*
- static void [PINT\\_PinInterruptClrFallFlagAll](#) (PINT\_Type \*base)
- Clear all pin interrupt fall flags.*
- static uint32\_t [PINT\\_PinInterruptGetFallFlagAll](#) (PINT\_Type \*base)
- Get all pin interrupt fall flags.*
- static void [PINT\\_PinInterruptClrRiseFlag](#) (PINT\_Type \*base, [pint\\_pin\\_int\\_t](#) pintr)
- Clear Selected pin interrupt rise flag.*
- static uint32\_t [PINT\\_PinInterruptGetRiseFlag](#) (PINT\_Type \*base, [pint\\_pin\\_int\\_t](#) pintr)
- Get selected pin interrupt rise flag.*
- static void [PINT\\_PinInterruptClrRiseFlagAll](#) (PINT\_Type \*base)
- Clear all pin interrupt rise flags.*
- static uint32\_t [PINT\\_PinInterruptGetRiseFlagAll](#) (PINT\_Type \*base)
- Get all pin interrupt rise flags.*
- void [PINT\\_PatternMatchConfig](#) (PINT\_Type \*base, [pint\\_pmatch\\_bslice\\_t](#) bslice, [pint\\_pmatch\\_cfg\\_t](#) \*cfg)
- Configure PINT pattern match.*
- void [PINT\\_PatternMatchGetConfig](#) (PINT\_Type \*base, [pint\\_pmatch\\_bslice\\_t](#) bslice, [pint\\_pmatch\\_cfg\\_t](#) \*cfg)
- Get PINT pattern match configuration.*
- static uint32\_t [PINT\\_PatternMatchGetStatus](#) (PINT\_Type \*base, [pint\\_pmatch\\_bslice\\_t](#) bslice)
- Get pattern match bit slice status.*
- static uint32\_t [PINT\\_PatternMatchGetStatusAll](#) (PINT\_Type \*base)
- Get status of all pattern match bit slices.*
- uint32\_t [PINT\\_PatternMatchResetDetectLogic](#) (PINT\_Type \*base)
- Reset pattern match detection logic.*
- static void [PINT\\_PatternMatchEnable](#) (PINT\_Type \*base)
- Enable pattern match function.*
- static void [PINT\\_PatternMatchDisable](#) (PINT\_Type \*base)
- Disable pattern match function.*
- static void [PINT\\_PatternMatchEnableRXEV](#) (PINT\_Type \*base)
- Enable RXEV output.*
- static void [PINT\\_PatternMatchDisableRXEV](#) (PINT\_Type \*base)
- Disable RXEV output.*
- void [PINT\\_EnableCallback](#) (PINT\_Type \*base)
- Enable callback.*
- void [PINT\\_DisableCallback](#) (PINT\_Type \*base)
- Disable callback.*
- void [PINT\\_Deinit](#) (PINT\_Type \*base)
- Deinitialize PINT peripheral.*
- void [PINT\\_EnableCallbackByIndex](#) (PINT\_Type \*base, [pint\\_pin\\_int\\_t](#) pintIdx)
- enable callback by pin index.*
- void [PINT\\_DisableCallbackByIndex](#) (PINT\_Type \*base, [pint\\_pin\\_int\\_t](#) pintIdx)
- disable callback by pin index.*

## Enumeration Type Documentation

### Driver version

- #define **FSL\_PINT\_DRIVER\_VERSION** (MAKE\_VERSION(2, 1, 3))  
Version 2.1.3.

### 24.3 Typedef Documentation

#### 24.3.1 typedef void(\* pint\_cb\_t)(pint\_pin\_int\_t pintr, uint32\_t pmatch\_status)

### 24.4 Enumeration Type Documentation

#### 24.4.1 enum pint\_pin\_enable\_t

Enumerator

**kPINT\_PinIntEnableNone** Do not generate Pin Interrupt.  
**kPINT\_PinIntEnableRiseEdge** Generate Pin Interrupt on rising edge.  
**kPINT\_PinIntEnableFallEdge** Generate Pin Interrupt on falling edge.  
**kPINT\_PinIntEnableBothEdges** Generate Pin Interrupt on both edges.  
**kPINT\_PinIntEnableLowLevel** Generate Pin Interrupt on low level.  
**kPINT\_PinIntEnableHighLevel** Generate Pin Interrupt on high level.

#### 24.4.2 enum pint\_pin\_int\_t

Enumerator

**kPINT\_PinInt0** Pin Interrupt 0.  
**kPINT\_PinInt1** Pin Interrupt 1.  
**kPINT\_PinInt2** Pin Interrupt 2.  
**kPINT\_PinInt3** Pin Interrupt 3.

#### 24.4.3 enum pint\_pmatch\_input\_src\_t

Enumerator

**kPINT\_PatternMatchInp0Src** Input source 0.  
**kPINT\_PatternMatchInp1Src** Input source 1.  
**kPINT\_PatternMatchInp2Src** Input source 2.  
**kPINT\_PatternMatchInp3Src** Input source 3.  
**kPINT\_PatternMatchInp4Src** Input source 4.  
**kPINT\_PatternMatchInp5Src** Input source 5.  
**kPINT\_PatternMatchInp6Src** Input source 6.  
**kPINT\_PatternMatchInp7Src** Input source 7.

#### 24.4.4 enum pint\_pmatch\_bslice\_t

Enumerator

*kPINT\_PatternMatchBSlice0* Bit slice 0.  
*kPINT\_PatternMatchBSlice1* Bit slice 1.  
*kPINT\_PatternMatchBSlice2* Bit slice 2.  
*kPINT\_PatternMatchBSlice3* Bit slice 3.

#### 24.4.5 enum pint\_pmatch\_bslice\_cfg\_t

Enumerator

*kPINT\_PatternMatchAlways* Always Contributes to product term match.  
*kPINT\_PatternMatchStickyRise* Sticky Rising edge.  
*kPINT\_PatternMatchStickyFall* Sticky Falling edge.  
*kPINT\_PatternMatchStickyBothEdges* Sticky Rising or Falling edge.  
*kPINT\_PatternMatchHigh* High level.  
*kPINT\_PatternMatchLow* Low level.  
*kPINT\_PatternMatchNever* Never contributes to product term match.  
*kPINT\_PatternMatchBothEdges* Either rising or falling edge.

### 24.5 Function Documentation

#### 24.5.1 void PINT\_Init ( PINT\_Type \* *base* )

This function initializes the PINT peripheral and enables the clock.

Parameters

|             |                                      |
|-------------|--------------------------------------|
| <i>base</i> | Base address of the PINT peripheral. |
|-------------|--------------------------------------|

Return values

|              |  |
|--------------|--|
| <i>None.</i> |  |
|--------------|--|

#### 24.5.2 void PINT\_PinInterruptConfig ( PINT\_Type \* *base*, pint\_pin\_int\_t *intr*, pint\_pin\_enable\_t *enable*, pint\_cb\_t *callback* )

This function configures a given pin interrupt.

## Function Documentation

### Parameters

|                 |                                      |
|-----------------|--------------------------------------|
| <i>base</i>     | Base address of the PINT peripheral. |
| <i>intr</i>     | Pin interrupt.                       |
| <i>enable</i>   | Selects detection logic.             |
| <i>callback</i> | Callback.                            |

### Return values

|              |  |
|--------------|--|
| <i>None.</i> |  |
|--------------|--|

### 24.5.3 void PINT\_PinInterruptGetConfig ( PINT\_Type \* *base*, pint\_pin\_int\_t *pintr*, pint\_pin\_enable\_t \* *enable*, pint\_cb\_t \* *callback* )

This function returns the configuration of a given pin interrupt.

### Parameters

|                 |                                       |
|-----------------|---------------------------------------|
| <i>base</i>     | Base address of the PINT peripheral.  |
| <i>pintr</i>    | Pin interrupt.                        |
| <i>enable</i>   | Pointer to store the detection logic. |
| <i>callback</i> | Callback.                             |

### Return values

|              |  |
|--------------|--|
| <i>None.</i> |  |
|--------------|--|

### 24.5.4 void PINT\_PinInterruptClrStatus ( PINT\_Type \* *base*, pint\_pin\_int\_t *pintr* )

This function clears the selected pin interrupt status.

### Parameters

|              |                                      |
|--------------|--------------------------------------|
| <i>base</i>  | Base address of the PINT peripheral. |
| <i>pintr</i> | Pin interrupt.                       |



Return values

|              |  |
|--------------|--|
| <i>None.</i> |  |
|--------------|--|

#### 24.5.5 static uint32\_t PINT\_PinInterruptGetStatus ( PINT\_Type \* *base*, pint\_pin\_int\_t *pintr* ) [inline], [static]

This function returns the selected pin interrupt status.

Parameters

|              |                                      |
|--------------|--------------------------------------|
| <i>base</i>  | Base address of the PINT peripheral. |
| <i>pintr</i> | Pin interrupt.                       |

Return values

|               |                                                                          |
|---------------|--------------------------------------------------------------------------|
| <i>status</i> | = 0 No pin interrupt request. = 1 Selected Pin interrupt request active. |
|---------------|--------------------------------------------------------------------------|

#### 24.5.6 void PINT\_PinInterruptClrStatusAll ( PINT\_Type \* *base* )

This function clears the status of all pin interrupts.

Parameters

|             |                                      |
|-------------|--------------------------------------|
| <i>base</i> | Base address of the PINT peripheral. |
|-------------|--------------------------------------|

Return values

|              |  |
|--------------|--|
| <i>None.</i> |  |
|--------------|--|

#### 24.5.7 static uint32\_t PINT\_PinInterruptGetStatusAll ( PINT\_Type \* *base* ) [inline], [static]

This function returns the status of all pin interrupts.

Parameters

---

## Function Documentation

|             |                                      |
|-------------|--------------------------------------|
| <i>base</i> | Base address of the PINT peripheral. |
|-------------|--------------------------------------|

Return values

|               |                                                                                                                                           |
|---------------|-------------------------------------------------------------------------------------------------------------------------------------------|
| <i>status</i> | Each bit position indicates the status of corresponding pin interrupt. = 0<br>No pin interrupt request. = 1 Pin interrupt request active. |
|---------------|-------------------------------------------------------------------------------------------------------------------------------------------|

### 24.5.8 static void PINT\_PinInterruptClrFallFlag ( PINT\_Type \* *base*, pint\_pin\_int\_t *pintr* ) [inline], [static]

This function clears the selected pin interrupt fall flag.

Parameters

|              |                                      |
|--------------|--------------------------------------|
| <i>base</i>  | Base address of the PINT peripheral. |
| <i>pintr</i> | Pin interrupt.                       |

Return values

|              |  |
|--------------|--|
| <i>None.</i> |  |
|--------------|--|

### 24.5.9 static uint32\_t PINT\_PinInterruptGetFallFlag ( PINT\_Type \* *base*, pint\_pin\_int\_t *pintr* ) [inline], [static]

This function returns the selected pin interrupt fall flag.

Parameters

|              |                                      |
|--------------|--------------------------------------|
| <i>base</i>  | Base address of the PINT peripheral. |
| <i>pintr</i> | Pin interrupt.                       |

Return values

|             |                                                                             |
|-------------|-----------------------------------------------------------------------------|
| <i>flag</i> | = 0 Falling edge has not been detected. = 1 Falling edge has been detected. |
|-------------|-----------------------------------------------------------------------------|

### 24.5.10 static void PINT\_PinInterruptClrFallFlagAll ( PINT\_Type \* *base* ) [inline], [static]

This function clears the fall flag for all pin interrupts.

## Parameters

|             |                                      |
|-------------|--------------------------------------|
| <i>base</i> | Base address of the PINT peripheral. |
|-------------|--------------------------------------|

## Return values

|              |  |
|--------------|--|
| <i>None.</i> |  |
|--------------|--|

### 24.5.11 static uint32\_t PINT\_PinInterruptGetFallFlagAll ( PINT\_Type \* *base* ) [inline], [static]

This function returns the fall flag of all pin interrupts.

## Parameters

|             |                                      |
|-------------|--------------------------------------|
| <i>base</i> | Base address of the PINT peripheral. |
|-------------|--------------------------------------|

## Return values

|              |                                                                                                                                                                      |
|--------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>flags</i> | Each bit position indicates the falling edge detection of the corresponding pin interrupt. 0 Falling edge has not been detected. = 1 Falling edge has been detected. |
|--------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------|

### 24.5.12 static void PINT\_PinInterruptClrRiseFlag ( PINT\_Type \* *base*, pint\_pin\_int\_t *pintr* ) [inline], [static]

This function clears the selected pin interrupt rise flag.

## Parameters

|              |                                      |
|--------------|--------------------------------------|
| <i>base</i>  | Base address of the PINT peripheral. |
| <i>pintr</i> | Pin interrupt.                       |

## Return values

|              |  |
|--------------|--|
| <i>None.</i> |  |
|--------------|--|

---

## Function Documentation

**24.5.13** `static uint32_t PINT_PinInterruptGetRiseFlag ( PINT_Type * base,  
pint_pin_int_t pintr ) [inline], [static]`

This function returns the selected pin interrupt rise flag.

## Parameters

|              |                                      |
|--------------|--------------------------------------|
| <i>base</i>  | Base address of the PINT peripheral. |
| <i>pintr</i> | Pin interrupt.                       |

## Return values

|             |                                                                           |
|-------------|---------------------------------------------------------------------------|
| <i>flag</i> | = 0 Rising edge has not been detected. = 1 Rising edge has been detected. |
|-------------|---------------------------------------------------------------------------|

#### 24.5.14 static void PINT\_PinInterruptClrRiseFlagAll ( PINT\_Type \* *base* ) [inline], [static]

This function clears the rise flag for all pin interrupts.

## Parameters

|             |                                      |
|-------------|--------------------------------------|
| <i>base</i> | Base address of the PINT peripheral. |
|-------------|--------------------------------------|

## Return values

|              |  |
|--------------|--|
| <i>None.</i> |  |
|--------------|--|

#### 24.5.15 static uint32\_t PINT\_PinInterruptGetRiseFlagAll ( PINT\_Type \* *base* ) [inline], [static]

This function returns the rise flag of all pin interrupts.

## Parameters

|             |                                      |
|-------------|--------------------------------------|
| <i>base</i> | Base address of the PINT peripheral. |
|-------------|--------------------------------------|

## Return values

|              |                                                                                                                                                                   |
|--------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>flags</i> | Each bit position indicates the rising edge detection of the corresponding pin interrupt. 0 Rising edge has not been detected. = 1 Rising edge has been detected. |
|--------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------|

**24.5.16** void PINT\_PatternMatchConfig ( PINT\_Type \* *base*, pint\_pmatch\_bslice\_t *bslice*, pint\_pmatch\_cfg\_t \* *cfg* )

This function configures a given pattern match bit slice.

## Parameters

|               |                                      |
|---------------|--------------------------------------|
| <i>base</i>   | Base address of the PINT peripheral. |
| <i>bslice</i> | Pattern match bit slice number.      |
| <i>cfg</i>    | Pointer to bit slice configuration.  |

## Return values

|              |  |
|--------------|--|
| <i>None.</i> |  |
|--------------|--|

#### 24.5.17 void PINT\_PatternMatchGetConfig ( PINT\_Type \* *base*, pint\_pmatch\_bslice\_t *bslice*, pint\_pmatch\_cfg\_t \* *cfg* )

This function returns the configuration of a given pattern match bit slice.

## Parameters

|               |                                      |
|---------------|--------------------------------------|
| <i>base</i>   | Base address of the PINT peripheral. |
| <i>bslice</i> | Pattern match bit slice number.      |
| <i>cfg</i>    | Pointer to bit slice configuration.  |

## Return values

|              |  |
|--------------|--|
| <i>None.</i> |  |
|--------------|--|

#### 24.5.18 static uint32\_t PINT\_PatternMatchGetStatus ( PINT\_Type \* *base*, pint\_pmatch\_bslice\_t *bslice* ) [inline], [static]

This function returns the status of selected bit slice.

## Parameters

|               |                                      |
|---------------|--------------------------------------|
| <i>base</i>   | Base address of the PINT peripheral. |
| <i>bslice</i> | Pattern match bit slice number.      |

## Return values

---

## Function Documentation

|               |                                                               |
|---------------|---------------------------------------------------------------|
| <i>status</i> | = 0 Match has not been detected. = 1 Match has been detected. |
|---------------|---------------------------------------------------------------|

### 24.5.19 static uint32\_t PINT\_PatternMatchGetStatusAll ( PINT\_Type \* *base* ) [inline], [static]

This function returns the status of all bit slices.

Parameters

|             |                                      |
|-------------|--------------------------------------|
| <i>base</i> | Base address of the PINT peripheral. |
|-------------|--------------------------------------|

Return values

|               |                                                                                                                                        |
|---------------|----------------------------------------------------------------------------------------------------------------------------------------|
| <i>status</i> | Each bit position indicates the match status of corresponding bit slice. = 0 Match has not been detected. = 1 Match has been detected. |
|---------------|----------------------------------------------------------------------------------------------------------------------------------------|

### 24.5.20 uint32\_t PINT\_PatternMatchResetDetectLogic ( PINT\_Type \* *base* )

This function resets the pattern match detection logic if any of the product term is matching.

Parameters

|             |                                      |
|-------------|--------------------------------------|
| <i>base</i> | Base address of the PINT peripheral. |
|-------------|--------------------------------------|

Return values

|                 |                                                                                                                              |
|-----------------|------------------------------------------------------------------------------------------------------------------------------|
| <i>pmstatus</i> | Each bit position indicates the match status of corresponding bit slice. = 0 Match was detected. = 1 Match was not detected. |
|-----------------|------------------------------------------------------------------------------------------------------------------------------|

### 24.5.21 static void PINT\_PatternMatchEnable ( PINT\_Type \* *base* ) [inline], [static]

This function enables the pattern match function.

Parameters

|             |                                      |
|-------------|--------------------------------------|
| <i>base</i> | Base address of the PINT peripheral. |
|-------------|--------------------------------------|



Return values

|              |  |
|--------------|--|
| <i>None.</i> |  |
|--------------|--|

#### 24.5.22 static void PINT\_PatternMatchDisable ( PINT\_Type \* *base* ) [inline], [static]

This function disables the pattern match function.

Parameters

|             |                                      |
|-------------|--------------------------------------|
| <i>base</i> | Base address of the PINT peripheral. |
|-------------|--------------------------------------|

Return values

|              |  |
|--------------|--|
| <i>None.</i> |  |
|--------------|--|

#### 24.5.23 static void PINT\_PatternMatchEnableRXEV ( PINT\_Type \* *base* ) [inline], [static]

This function enables the pattern match RXEV output.

Parameters

|             |                                      |
|-------------|--------------------------------------|
| <i>base</i> | Base address of the PINT peripheral. |
|-------------|--------------------------------------|

Return values

|              |  |
|--------------|--|
| <i>None.</i> |  |
|--------------|--|

#### 24.5.24 static void PINT\_PatternMatchDisableRXEV ( PINT\_Type \* *base* ) [inline], [static]

This function disables the pattern match RXEV output.

Parameters

|             |                                      |
|-------------|--------------------------------------|
| <i>base</i> | Base address of the PINT peripheral. |
|-------------|--------------------------------------|

## Function Documentation

Return values

|              |  |
|--------------|--|
| <i>None.</i> |  |
|--------------|--|

### 24.5.25 void PINT\_EnableCallback ( PINT\_Type \* *base* )

This function enables the interrupt for the selected PINT peripheral. Although the pin(s) are monitored as soon as they are enabled, the callback function is not enabled until this function is called.

Parameters

|             |                                      |
|-------------|--------------------------------------|
| <i>base</i> | Base address of the PINT peripheral. |
|-------------|--------------------------------------|

Return values

|              |  |
|--------------|--|
| <i>None.</i> |  |
|--------------|--|

### 24.5.26 void PINT\_DisableCallback ( PINT\_Type \* *base* )

This function disables the interrupt for the selected PINT peripheral. Although the pins are still being monitored but the callback function is not called.

Parameters

|             |                                 |
|-------------|---------------------------------|
| <i>base</i> | Base address of the peripheral. |
|-------------|---------------------------------|

Return values

|              |  |
|--------------|--|
| <i>None.</i> |  |
|--------------|--|

### 24.5.27 void PINT\_Deinit ( PINT\_Type \* *base* )

This function disables the PINT clock.

Parameters

|             |                                      |
|-------------|--------------------------------------|
| <i>base</i> | Base address of the PINT peripheral. |
|-------------|--------------------------------------|

Return values

|              |  |
|--------------|--|
| <i>None.</i> |  |
|--------------|--|

#### 24.5.28 void PINT\_EnableCallbackByIndex ( PINT\_Type \* *base*, pint\_pin\_int\_t *pinIdx* )

This function enables callback by pin index instead of enabling all pins.

Parameters

|               |                                 |
|---------------|---------------------------------|
| <i>base</i>   | Base address of the peripheral. |
| <i>pinIdx</i> | pin index.                      |

Return values

|              |  |
|--------------|--|
| <i>None.</i> |  |
|--------------|--|

#### 24.5.29 void PINT\_DisableCallbackByIndex ( PINT\_Type \* *base*, pint\_pin\_int\_t *pinIdx* )

This function disables callback by pin index instead of disabling all pins.

Parameters

|               |                                 |
|---------------|---------------------------------|
| <i>base</i>   | Base address of the peripheral. |
| <i>pinIdx</i> | pin index.                      |

Return values

|              |  |
|--------------|--|
| <i>None.</i> |  |
|--------------|--|



## Chapter 25

### Power: Power driver

#### 25.1 Overview

The MCUXpresso SDK provides a power driver for the MCUXpresso SDK devices.

#### 25.2 Function description

Power driver and library provides these functions:

- Functions to enable and disable power to different peripherals
- Functions to obtain power down config structure with default parameters
- Functions to determine cause of reset
- Functions to get power Library API to return the library version.

##### 25.2.1 Power enable and disable

Power driver provides two API's [POWER\\_EnablePD\(\)](#) and [POWER\\_DisablePD\(\)](#) to enable or disable the PDRUNCFG bits in SYSCON. The PDRUNCFG has an inverted logic i.e. the peripheral is powered on when the bit is cleared and powered off when bit is set. So the API [POWER\\_DisablePD\(\)](#) is used to power on a peripheral and [POWER\\_EnablePD\(\)](#) is used to power off a peripheral. The API's take a parameter of type `pd_bit_t` which organizes the PDRUNCFG bits. The driver also provides two separate API's to power down and power up Flash, [POWER\\_PowerDownFlash\(\)](#) and [POWER\\_PowerUpFlash\(\)](#)

#### Files

- file [fsl\\_power.h](#)

#### Data Structures

- struct [pm\\_bod\\_cfg\\_t](#)  
*BOD config. [More...](#)*
- struct [pm\\_power\\_config\\_t](#)  
*Power config. [More...](#)*

#### Macros

- #define [POWER\\_BOD\\_ENABLE](#) ( 1 << 0 )  
*BODVBAT configuration flag.*
- #define [POWER\\_BOD\\_HIGH](#) ( 1 << 3 )  
*ES2 BOD VBAT only.*
- #define [POWER\\_BOD\\_LVL\\_1\\_75V](#) 9  
*BOD trigger level setting.*

## Function description

- #define **POWER\_BOD\_LVL\_1\_8V** 10  
*BOD trigger level 1.8V.*
- #define **POWER\_BOD\_LVL\_1\_9V** 11  
*BOD trigger level 1.9V.*
- #define **POWER\_BOD\_LVL\_2\_0V** 12  
*BOD trigger level 2.0V.*
- #define **POWER\_BOD\_LVL\_2\_1V** 13  
*BOD trigger level 2.1V.*
- #define **POWER\_BOD\_LVL\_2\_2V** 14  
*BOD trigger level 2.2V.*
- #define **POWER\_BOD\_LVL\_2\_3V** 15  
*BOD trigger level 2.3V.*
- #define **POWER\_BOD\_LVL\_2\_4V** 16  
*BOD trigger level 2.4V.*
- #define **POWER\_BOD\_LVL\_2\_5V** 17  
*BOD trigger level 2.5V.*
- #define **POWER\_BOD\_LVL\_2\_6V** 18  
*BOD trigger level 2.6V.*
- #define **POWER\_BOD\_LVL\_2\_7V** 19  
*BOD trigger level 2.7V.*
- #define **POWER\_BOD\_LVL\_2\_8V** 20  
*BOD trigger level 2.8V.*
- #define **POWER\_BOD\_LVL\_2\_9V** 21  
*BOD trigger level 2.9V.*
- #define **POWER\_BOD\_LVL\_3\_0V** 22  
*BOD trigger level 3.0V.*
- #define **POWER\_BOD\_LVL\_3\_1V** 23  
*BOD trigger level 3.1V.*
- #define **POWER\_BOD\_LVL\_3\_2V** 24  
*BOD trigger level 3.2V.*
- #define **POWER\_BOD\_LVL\_3\_3V** 25  
*BOD trigger level 3.3V.*
- #define **POWER\_BOD\_HYST\_25MV** 0  
*BOD Hysteresis control setting.*
- #define **POWER\_BOD\_HYST\_50MV** 1  
*BOD Hysteresis control 50mV.*
- #define **POWER\_BOD\_HYST\_75MV** 2  
*BOD Hysteresis control 75mV.*
- #define **POWER\_BOD\_HYST\_100MV** 3  
*BOD Hysteresis control 100mV, default at Reset.*
- #define **PM\_CFG\_SRAM\_BANK\_BIT\_BASE** 0  
*SRAM banks definition list for retention in power down modes !*
- #define **PM\_CFG\_SRAM\_BANK0\_RET** (1<<0)  
*On ES1, this bank shall be kept in retention for Warmstart from power down.*
- #define **PM\_CFG\_SRAM\_BANK1\_RET** (1<<1)  
*Bank 1 shall be kept in retention.*
- #define **PM\_CFG\_SRAM\_BANK2\_RET** (1<<2)  
*Bank 2 shall be kept in retention.*
- #define **PM\_CFG\_SRAM\_BANK3\_RET** (1<<3)  
*Bank 3 shall be kept in retention.*
- #define **PM\_CFG\_SRAM\_BANK4\_RET** (1<<4)

- *Bank 4 shall be kept in retention.*  
#define [PM\\_CFG\\_SRAM\\_BANK5\\_RET](#) (1<<5)
- *Bank 5 shall be kept in retention.*  
#define [PM\\_CFG\\_SRAM\\_BANK6\\_RET](#) (1<<6)
- *Bank 6 shall be kept in retention.*  
#define [PM\\_CFG\\_SRAM\\_BANK7\\_RET](#) (1<<7)
- *On ES2, this bank shall be kept in retention for Warmstart.*  
#define [PM\\_CFG\\_SRAM\\_BANK8\\_RET](#) (1<<8)
- *Bank 8 shall be kept in retention.*  
#define [PM\\_CFG\\_SRAM\\_BANK9\\_RET](#) (1<<9)
- *Bank 9 shall be kept in retention.*  
#define [PM\\_CFG\\_SRAM\\_BANK10\\_RET](#) (1<<10)
- *Bank 10 shall be kept in retention.*  
#define [PM\\_CFG\\_SRAM\\_BANK11\\_RET](#) (1<<11)
- *Bank 11 shall be kept in retention.*  
#define [PM\\_CFG\\_SRAM\\_ALL\\_RETENTION](#) 0xFF
- *All banks shall be kept in retention.*  
#define [PM\\_CFG\\_KEEP\\_AO\\_VOLTAGE](#) (1<<15)  
*keep the same voltage on the Always-on power domain - typical used with FRO32K to avoid timebase drift*
- #define [POWER\\_WAKEUPSRC\\_SYSTEM](#) [LOWPOWER\\_WAKEUPSRCINT0\\_SYSTEM\\_IRQ](#)  
*BOD, Watchdog Timer, Flash controller, [DEEP SLEEP] BODVBAT [POWER\_DOWN].*
- #define [POWER\\_WAKEUPSRC\\_DMA](#) [LOWPOWER\\_WAKEUPSRCINT0\\_DMA\\_IRQ](#)  
*[DEEP SLEEP]*
- #define [POWER\\_WAKEUPSRC\\_GINT](#) [LOWPOWER\\_WAKEUPSRCINT0\\_GINT\\_IRQ](#)  
*[DEEP SLEEP]*
- #define [POWER\\_WAKEUPSRC\\_IRBLASTER](#) [LOWPOWER\\_WAKEUPSRCINT0\\_IRBLASTER\\_IRQ](#)  
*[DEEP SLEEP]*
- #define [POWER\\_WAKEUPSRC\\_PINT0](#) [LOWPOWER\\_WAKEUPSRCINT0\\_PINT0\\_IRQ](#)  
*[DEEP SLEEP]*
- #define [POWER\\_WAKEUPSRC\\_PINT1](#) [LOWPOWER\\_WAKEUPSRCINT0\\_PINT1\\_IRQ](#)  
*[DEEP SLEEP]*
- #define [POWER\\_WAKEUPSRC\\_PINT2](#) [LOWPOWER\\_WAKEUPSRCINT0\\_PINT2\\_IRQ](#)  
*[DEEP SLEEP]*
- #define [POWER\\_WAKEUPSRC\\_PINT3](#) [LOWPOWER\\_WAKEUPSRCINT0\\_PINT3\\_IRQ](#)  
*[DEEP SLEEP]*
- #define [POWER\\_WAKEUPSRC\\_SPIFI](#) [LOWPOWER\\_WAKEUPSRCINT0\\_SPIFI\\_IRQ](#)  
*[DEEP SLEEP]*
- #define [POWER\\_WAKEUPSRC\\_TIMER0](#) [LOWPOWER\\_WAKEUPSRCINT0\\_TIMER0\\_IRQ](#)  
*[DEEP SLEEP]*
- #define [POWER\\_WAKEUPSRC\\_TIMER1](#) [LOWPOWER\\_WAKEUPSRCINT0\\_TIMER1\\_IRQ](#)  
*[DEEP SLEEP]*
- #define [POWER\\_WAKEUPSRC\\_USART0](#) [LOWPOWER\\_WAKEUPSRCINT0\\_USART0\\_IRQ](#)  
*[DEEP SLEEP, POWER\_DOWN]*
- #define [POWER\\_WAKEUPSRC\\_USART1](#) [LOWPOWER\\_WAKEUPSRCINT0\\_USART1\\_IRQ](#)  
*[DEEP SLEEP]*
- #define [POWER\\_WAKEUPSRC\\_I2C0](#) [LOWPOWER\\_WAKEUPSRCINT0\\_I2C0\\_IRQ](#)  
*[DEEP SLEEP, POWER\_DOWN]*
- #define [POWER\\_WAKEUPSRC\\_I2C1](#) [LOWPOWER\\_WAKEUPSRCINT0\\_I2C1\\_IRQ](#)  
*[DEEP SLEEP]*
- #define [POWER\\_WAKEUPSRC\\_SPI0](#) [LOWPOWER\\_WAKEUPSRCINT0\\_SPI0\\_IRQ](#)

## Function description

- *[DEEP SLEEP, POWER DOWN]*  
• #define [POWER\\_WAKEUPSRC\\_SPI1](#) [LOWPOWER\\_WAKEUPSRCINT0\\_SPI1\\_IRQ](#)
- *[DEEP SLEEP]*  
• #define [POWER\\_WAKEUPSRC\\_PWM0](#) [LOWPOWER\\_WAKEUPSRCINT0\\_PWM0\\_IRQ](#)
- *[DEEP SLEEP]*  
• #define [POWER\\_WAKEUPSRC\\_PWM1](#) [LOWPOWER\\_WAKEUPSRCINT0\\_PWM1\\_IRQ](#)
- *[DEEP SLEEP]*  
• #define [POWER\\_WAKEUPSRC\\_PWM2](#) [LOWPOWER\\_WAKEUPSRCINT0\\_PWM2\\_IRQ](#)
- *[DEEP SLEEP]*  
• #define [POWER\\_WAKEUPSRC\\_PWM3](#) [LOWPOWER\\_WAKEUPSRCINT0\\_PWM3\\_IRQ](#)
- *[DEEP SLEEP]*  
• #define [POWER\\_WAKEUPSRC\\_PWM4](#) [LOWPOWER\\_WAKEUPSRCINT0\\_PWM4\\_IRQ](#)
- *[DEEP SLEEP]*  
• #define [POWER\\_WAKEUPSRC\\_PWM5](#) [LOWPOWER\\_WAKEUPSRCINT0\\_PWM5\\_IRQ](#)
- *[DEEP SLEEP]*  
• #define [POWER\\_WAKEUPSRC\\_PWM6](#) [LOWPOWER\\_WAKEUPSRCINT0\\_PWM6\\_IRQ](#)
- *[DEEP SLEEP]*  
• #define [POWER\\_WAKEUPSRC\\_PWM7](#) [LOWPOWER\\_WAKEUPSRCINT0\\_PWM7\\_IRQ](#)
- *[DEEP SLEEP]*  
• #define [POWER\\_WAKEUPSRC\\_PWM8](#) [LOWPOWER\\_WAKEUPSRCINT0\\_PWM8\\_IRQ](#)
- *[DEEP SLEEP]*  
• #define [POWER\\_WAKEUPSRC\\_PWM9](#) [LOWPOWER\\_WAKEUPSRCINT0\\_PWM9\\_IRQ](#)
- *[DEEP SLEEP]*  
• #define [POWER\\_WAKEUPSRC\\_PWM10](#) [LOWPOWER\\_WAKEUPSRCINT0\\_PWM10\\_IRQ](#)
- *[DEEP SLEEP]*  
• #define [POWER\\_WAKEUPSRC\\_I2C2](#) [LOWPOWER\\_WAKEUPSRCINT0\\_I2C2\\_IRQ](#)
- *[DEEP SLEEP]*  
• #define [POWER\\_WAKEUPSRC\\_RTC](#) [LOWPOWER\\_WAKEUPSRCINT0\\_RTC\\_IRQ](#)
- *[DEEP SLEEP, POWER DOWN]*  
• #define [POWER\\_WAKEUPSRC\\_NFCTAG](#) [LOWPOWER\\_WAKEUPSRCINT0\\_NFCTAG\\_IRQ](#)
- *[DEEP SLEEP, POWER DOWN (ES2 Only), DEEP DOWN (ES2 only)]*  
• #define [POWER\\_WAKEUPSRC\\_MAILBOX](#) [LOWPOWER\\_WAKEUPSRCINT0\\_MAILBOX\\_IRQ](#)  
*Mailbox, Wake-up from DEEP SLEEP and POWER DOWN low power mode [DEEP SLEEP, POWER DOWN].*
- #define [POWER\\_WAKEUPSRC\\_ADC\\_SEQA](#) ((uint64\_t)[LOWPOWER\\_WAKEUPSRCINT1\\_ADC\\_SEQA\\_IRQ](#) << 32)  
*[DEEP SLEEP]*
- #define [POWER\\_WAKEUPSRC\\_ADC\\_SEQB](#) ((uint64\_t)[LOWPOWER\\_WAKEUPSRCINT1\\_ADC\\_SEQB\\_IRQ](#) << 32)  
*[DEEP SLEEP]*
- #define [POWER\\_WAKEUPSRC\\_ADC\\_THCMP\\_OVR](#) ((uint64\_t)[LOWPOWER\\_WAKEUPSRCINT1\\_ADC\\_THCMP\\_OVR\\_IRQ](#) << 32)  
*[DEEP SLEEP]*
- #define [POWER\\_WAKEUPSRC\\_DMIC](#) ((uint64\_t)[LOWPOWER\\_WAKEUPSRCINT1\\_DMIC\\_IRQ](#) << 32)  
*[DEEP SLEEP]*
- #define [POWER\\_WAKEUPSRC\\_HWVAD](#) ((uint64\_t)[LOWPOWER\\_WAKEUPSRCINT1\\_HWVAD\\_IRQ](#) << 32)  
*[DEEP SLEEP]*



- #define `POWER_WAKEUPSRC_BLE_DP` ((uint64\_t)LOWPOWER\_WAKEUPSRCINT1\_BLE\_DP\_IRQ << 32)  
[DEEP SLEEP]
- #define `POWER_WAKEUPSRC_BLE_DP0` ((uint64\_t)LOWPOWER\_WAKEUPSRCINT1\_BLE\_DP0\_IRQ << 32)  
[DEEP SLEEP]
- #define `POWER_WAKEUPSRC_BLE_DP1` ((uint64\_t)LOWPOWER\_WAKEUPSRCINT1\_BLE\_DP1\_IRQ << 32)  
[DEEP SLEEP]
- #define `POWER_WAKEUPSRC_BLE_DP2` ((uint64\_t)LOWPOWER\_WAKEUPSRCINT1\_BLE\_DP2\_IRQ << 32)  
[DEEP SLEEP]
- #define `POWER_WAKEUPSRC_BLE_LL_ALL` ((uint64\_t)LOWPOWER\_WAKEUPSRCINT1\_BLE\_LL\_ALL\_IRQ << 32)  
[DEEP SLEEP]
- #define `POWER_WAKEUPSRC_ZIGBEE_MAC` ((uint64\_t)LOWPOWER\_WAKEUPSRCINT1\_ZIGBEE\_MAC\_IRQ << 32)  
[DEEP SLEEP]
- #define `POWER_WAKEUPSRC_ZIGBEE_MODEM` ((uint64\_t)LOWPOWER\_WAKEUPSRCINT1\_ZIGBEE\_MODEM\_IRQ << 32)  
[DEEP SLEEP]
- #define `POWER_WAKEUPSRC_RFP_TMU` ((uint64\_t)LOWPOWER\_WAKEUPSRCINT1\_RFP\_TMU\_IRQ << 32)  
[DEEP SLEEP]
- #define `POWER_WAKEUPSRC_RFP_AGC` ((uint64\_t)LOWPOWER\_WAKEUPSRCINT1\_RFP\_AGC\_IRQ << 32)  
[DEEP SLEEP]
- #define `POWER_WAKEUPSRC_ISO7816` ((uint64\_t)LOWPOWER\_WAKEUPSRCINT1\_ISO7816\_IRQ << 32)  
[DEEP SLEEP]
- #define `POWER_WAKEUPSRC_ANA_COMP` ((uint64\_t)LOWPOWER\_WAKEUPSRCINT1\_ANA\_COMP\_IRQ << 32)  
[DEEP SLEEP, POWER DOWN]
- #define `POWER_WAKEUPSRC_WAKE_UP_TIMER0` ((uint64\_t)LOWPOWER\_WAKEUPSRCINT1\_WAKE\_UP\_TIMER0\_IRQ << 32)  
[DEEP SLEEP, POWER DOWN]
- #define `POWER_WAKEUPSRC_WAKE_UP_TIMER1` ((uint64\_t)LOWPOWER\_WAKEUPSRCINT1\_WAKE\_UP\_TIMER1\_IRQ << 32)  
[DEEP SLEEP, POWER DOWN]
- #define `POWER_WAKEUPSRC_BLE_WAKE_TIMER` ((uint64\_t)LOWPOWER\_WAKEUPSRCINT1\_BLE\_WAKE\_TIMER\_IRQ << 32)  
[DEEP SLEEP, POWER DOWN]
- #define `POWER_WAKEUPSRC_BLE_OSC_EN` ((uint64\_t)LOWPOWER\_WAKEUPSRCINT1\_BLE\_OSC\_EN\_IRQ << 32)  
[DEEP SLEEP, POWER DOWN]
- #define `POWER_WAKEUPSRC_IO` ((uint64\_t)LOWPOWER\_WAKEUPSRCINT1\_IO\_IRQ << 32)  
[POWER DOWN, DEEP DOWN]

## Function description

### Enumerations

- enum `pd_bit_t` {  
    `kPDRUNCFG_PD_LDO_ADC_EN` = 22,  
    `kPDRUNCFG_PD_BOD_MEM_EN` = 23,  
    `kPDRUNCFG_PD_BOD_CORE_EN` = 24,  
    `kPDRUNCFG_PD_FRO32K_EN` = 25,  
    `kPDRUNCFG_PD_XTAL32K_EN` = 26,  
    `kPDRUNCFG_PD_BOD_ANA_COMP_EN` = 27 }  
    *PDRUNCFG bits offset.*
- enum `pm_power_mode_t` {  
    `PM_POWER_DOWN`,  
    `PM_DEEP_DOWN` }  
    *Power modes.*
- enum `reset_cause_t` { ,  
    `RESET_POR` = (1 << 0),  
    `RESET_EXT_PIN` = (1 << 1),  
    `RESET_BOR` = (1 << 2),  
    `RESET_SYS_REQ` = (1 << 3),  
    `RESET_WDT` = (1 << 4),  
    `RESET_WAKE_DEEP_PD` = (1 << 5),  
    `RESET_WAKE_PD` = (1 << 6),  
    `RESET_SW_REQ` = (1 << 7) }  
    *Reset Cause definition.*
- enum `pm_ldo_volt_t` {  
    `PM_LDO_VOLT_1_1V_DEFAULT`,  
    `PM_LDO_VOLT_1_0V` }  
    *LDO voltage setting.*

### Power Configuration

- void `POWER_Init` (void)  
    *Initialize the sdk power drivers.*
- void `POWER_SetTrimDefaultActiveVoltage` (void)  
    *Optimize the LDO voltage for power saving Initialize the power domains.*
- void `POWER_BodSetUp` (void)  
    *BODMEM and BODCORE setup.*
- void `POWER_BodActivate` (void)  
    *enable SW reset for the BODCORE*
- static void `POWER_EnablePD` (`pd_bit_t` en)  
    *API to enable PDRUNCFG bit in the Syscon.*
- static void `POWER_DisablePD` (`pd_bit_t` en)  
    *API to disable PDRUNCFG bit in the Syscon.*
- static uint32\_t `POWER_GetIoWakeStatus` (void)  
    *Get IO and Ntag Field detect Wake-up sources from Power Down and Deep Power Down modes.*
- static void `POWER_EnterSleep` (void)  
    *Power API to enter sleep mode (Doze mode)*
- bool `POWER_EnterPowerMode` (`pm_power_mode_t` pm\_power\_mode, `pm_power_config_t` \*pm-  
    \_power\_config)

- *Power Library API to enter different power mode.*
- `reset_cause_t` `POWER_GetResetCause` (void)  
*determine cause of reset*
- `void` `POWER_ClearResetCause` (void)  
*Clear cause of reset.*
- `uint32_t` `POWER_GetLibVersion` (void)  
*Power Library API to return the library version.*
- `void` `POWER_BodVbatGetDefaultConfig` (`pm_bod_cfg_t` \*bod\_cfg\_p)  
*Get default Vbat BOD config parameters, level @ 1.75V, Hysteresis @ 100mV.*
- `bool` `POWER_BodVbatConfig` (`pm_bod_cfg_t` \*bod\_cfg\_p)  
*Configure the VBAT BOD.*
- `void` `POWER_ApplyLdoActiveVoltage` (`pm_ldo_volt_t` ldoVolt)  
*Configure the LDO voltage.*
- `#define` `POWER_ENTER_SLEEP()` `__DSB(); __WFI(); __ISB();`  
*Power API to enter sleep mode (Doze mode)*

## 25.3 Data Structure Documentation

### 25.3.1 struct pm\_bod\_cfg\_t

#### Data Fields

- `uint8_t` `bod_level`  
*BOD trigger level.*
- `uint8_t` `bod_hyst`  
*BOD Hysteresis control.*
- `uint8_t` `bod_cfg`  
*BOD config setting.*

### 25.3.2 struct pm\_power\_config\_t

#### Data Fields

- `pm_wake_source_t` `pm_wakeup_src`  
*Wakeup source select.*
- `uint32_t` `pm_wakeup_io`  
*Wakeup IO.*
- `uint32_t` `pm_config`  
*Power mode config.*

## 25.4 Macro Definition Documentation

### 25.4.1 #define POWER\_BOD\_LVL\_1\_75V 9

Default at Reset , 1.7V on ES1

## Enumeration Type Documentation

### 25.4.2 #define POWER\_BOD\_HYST\_25MV 0

BOD Hysteresis control 25mV

### 25.4.3 #define POWER\_ENTER\_SLEEP( ) \_\_DSB(); \_\_WFI(); \_\_ISB();

Note

: The static inline function has not the expected effect in -O0. In order to force inline this macro is added

Returns

none

## 25.5 Enumeration Type Documentation

### 25.5.1 enum pd\_bit\_t

Enumerator

*kPDRUNCFG\_PD\_LDO\_ADC\_EN* Offset is 22, LDO ADC enabled.  
*kPDRUNCFG\_PD\_BOD\_MEM\_EN* Offset is 23, BOD MEM enabled.  
*kPDRUNCFG\_PD\_BOD\_CORE\_EN* Offset is 24, BOD CORE enabled.  
*kPDRUNCFG\_PD\_FRO32K\_EN* Offset is 25, FRO32K enabled.  
*kPDRUNCFG\_PD\_XTAL32K\_EN* Offset is 26, XTAL32K enabled.  
*kPDRUNCFG\_PD\_BOD\_ANA\_COMP\_EN* Offset is 27, Analog Comparator enabled.

### 25.5.2 enum pm\_power\_mode\_t

Enumerator

*PM\_POWER\_DOWN* Power down mode.  
*PM\_DEEP\_DOWN* Deep power down mode.

### 25.5.3 enum reset\_cause\_t

Enumerator

*RESET\_POR* The last chip reset was caused by a Power On Reset.  
*RESET\_EXT\_PIN* The last chip reset was caused by a Pad Reset.  
*RESET\_BOR* The last chip reset was caused by a Brown Out Detector.

***RESET\_SYS\_REQ*** The last chip reset was caused by a System Reset requested by the ARM CPU.

***RESET\_WDT*** The last chip reset was caused by the Watchdog Timer.

***RESET\_WAKE\_DEEP\_PD*** The last chip reset was caused by a Wake-up I/O (GPIO or internal NTAG FD INT).

***RESET\_WAKE\_PD*** The last CPU reset was caused by a Wake-up from Power down (many sources possible: timer, IO, ...).

***RESET\_SW\_REQ*** The last chip reset was caused by a Software. ES2 Only

#### 25.5.4 enum pm\_ldo\_volt\_t

Enumerator

***PM\_LDO\_VOLT\_1\_1V\_DEFAULT*** LDO voltage 1.1V.

***PM\_LDO\_VOLT\_1\_0V*** not safe at system start/wakeup and CPU clock switch to higher frequency

### 25.6 Function Documentation

#### 25.6.1 void POWER\_Init ( void )

Optimize the LDO voltage for power saving Initialize the power domains

Returns

none

#### 25.6.2 void POWER\_SetTrimDefaultActiveVoltage ( void )

Returns

none

#### 25.6.3 void POWER\_BodSetUp ( void )

Enable the BOD core and BOD mem Disable the analog comparator clock

Returns

none

## Function Documentation

### 25.6.4 void POWER\_BodActivate ( void )

Returns

none

### 25.6.5 static void POWER\_EnablePD ( pd\_bit\_t *en* ) [inline], [static]

Note that enabling the bit powers down the peripheral

Parameters

|           |                                                 |
|-----------|-------------------------------------------------|
| <i>en</i> | peripheral for which to enable the PDRUNCFG bit |
|-----------|-------------------------------------------------|

Returns

none

### 25.6.6 static void POWER\_DisablePD ( pd\_bit\_t *en* ) [inline], [static]

Note that disabling the bit powers up the peripheral

Parameters

|           |                                                  |
|-----------|--------------------------------------------------|
| <i>en</i> | peripheral for which to disable the PDRUNCFG bit |
|-----------|--------------------------------------------------|

Returns

none

### 25.6.7 static uint32\_t POWER\_GetloWakeStatus ( void ) [inline], [static]

Allow to identify the wake-up source when waking up from Power-Down modes or Deep Power Down modes. Status is reset by POR, RSTN, WDT. bit in range from 0 to 21 are for DIO0 to DIO21 bit 22 is NTAG field detect wakeup source

Returns

IO and Field detect Wake-up source

### 25.6.8 static void POWER\_EnterSleep ( void ) [inline], [static]

Note

: If the user desires to program a wakeup timer before going to sleep, it needs to use either the fsl\_wtimer.h API or use the POWER\_SetLowPower() API instead see POWER\_ENTER\_SLEEP

Returns

none

### 25.6.9 bool POWER\_EnterPowerMode ( pm\_power\_mode\_t pm\_power\_mode, pm\_power\_config\_t \* pm\_power\_config )

If requested mode is PM\_POWER\_DOWN, the API will perform the clamping of the DIOs if the PIO register has the bit IO\_CLAMPING set: SYSCON->RETENTIONCTRL.IOCLAMP will be set

Parameters

|                      |             |
|----------------------|-------------|
| <i>pm_power_mode</i> | Power modes |
|----------------------|-------------|

See Also

[pm\\_power\\_mode\\_t](#)

Parameters

|                        |              |
|------------------------|--------------|
| <i>pm_power_config</i> | Power config |
|------------------------|--------------|

See Also

[pm\\_power\\_config\\_t](#)

Returns

false if chip could not go to sleep. Configuration structure is incorrect

### 25.6.10 reset\_cause\_t POWER\_GetResetCause ( void )

Returns

reset\_cause

**25.6.11 uint32\_t POWER\_GetLibVersion ( void )**



## Parameters

|             |  |
|-------------|--|
| <i>none</i> |  |
|-------------|--|

## Returns

version number of the power library

**25.6.12 void POWER\_BodVbatGetDefaultConfig ( pm\_bod\_cfg\_t \* bod\_cfg\_p )**

## Parameters

|                  |            |
|------------------|------------|
| <i>bod_cfg_p</i> | BOD config |
|------------------|------------|

## See Also

[pm\\_bod\\_cfg\\_t](#)

## Returns

none

**25.6.13 bool POWER\_BodVbatConfig ( pm\_bod\_cfg\_t \* bod\_cfg\_p )**

## Parameters

|                  |            |
|------------------|------------|
| <i>bod_cfg_p</i> | BOD config |
|------------------|------------|

## See Also

[pm\\_bod\\_cfg\\_t](#)

## Returns

false if configuration parameters are incorrect

**25.6.14 void POWER\_ApplyLdoActiveVoltage ( pm\_ldo\_volt\_t ldoVolt )**

## Function Documentation

### Parameters

|                |                     |
|----------------|---------------------|
| <i>ldoVolt</i> | LDO voltage setting |
|----------------|---------------------|

### See Also

[pm\\_ldo\\_volt\\_t](#)

### Returns

none

## Chapter 26

# PWM: Pulse Width Modulator

### 26.1 Overview

The SDK provides a driver for the Pulse Width Modulator (PWM).

The function [PWM\\_Init\(\)](#) initializes the PWM module with specified configurations, the function [PWM\\_GetDefaultConfig\(\)](#) could help to get the default configurations. The initialization function configures the module to use the specified clock for PWM operation.

The function [PWM\\_SetupPwm\(\)](#) sets up the PWM channel for PWM output. The function can set up PWM signal property the channel. The PWM has 10 channels: 0 to 9. Each channel has its own period, compare match value, and polarity specified. The settings are applied to the specified channel requesting PWM output. The period and compare match are 16-bit values. At the compare match value, within the period, the PWM output toggles. The period value is loaded to downcounter, which decrements to 0. Once it reaches 0, it reloads the count and starts the signal out again, until the PWM channel is stopped. The function also sets up the channel output level after the channel is disabled. The 11th channel (ChannelAll) is a special channel which outputs the same output signals on other 10 channels (0 to 9) when it is set up and enabled.

The function [PWM\\_ReadPeriodValue\(\)](#) reads the current period (downcounter value) for the PWM channel. The function [PWM\\_ReadCompareValue\(\)](#) reads the compare match value for the PWM channel.

The function [PWM\\_StartTimer\(\)](#) can be used to start the PWM channel. The function [PWM\\_StopTimer\(\)](#) can be used to stop the PWM channel.

Provide functions to get and clear the PWM status.

Provide functions to enable/disable PWM interrupts and get current enabled interrupts.

### 26.2 Typical use case

#### 26.2.1 PWM output

Configures PWM channel to output PWM signal.

```
int main(void)
{
 /* Structure of initialize PWM */
 pwm_config_t pwmConfig;
 pwm_setup_t pwmChan0;
 uint32_t pwmClockFrg;
 uint32_t pwmChan0Clk;

 /* Board pin, clock, debug console initialization */
 BOARD_InitHardware();

 PRINTF("PWM driver example\n");
```

## Typical use case

```
pwmClockFrq = CLOCK_GetFreq(kCLOCK_Pwm);

PWM_GetDefaultConfig(&pwmConfig);

/* Use 32MHz clock */
pwmConfig.clk_sel = kPWM_Osc32Mclk;

/* Initialize PWM */
if (PWM_Init (BOARD_PWM_BASEADDR, &pwmConfig) != kStatus_Success)
{
 PRINTF("PWM initialization failed\n");
 return 1;
}

/* Set up PWM channel 0 to generate PWM pulse of 100 us with 50% duty cycle */
pwmChan0.pol_ctrl = kPWM_SetHighOnMatchLowOnPeriod;
pwmChan0.dis_out_level = kPWM_SetLow;
pwmChan0.prescaler_val = 0;
pwmChan0Clk = pwmClockFrq / (1 + pwmChan0.prescaler_val);
pwmChan0.period_val = USEC_TO_COUNT(100, pwmChan0Clk);
pwmChan0.comp_val = pwmChan0.period_val / 2;
if (PWM_SetupPwm (BOARD_PWM_BASEADDR, kPWM_Pwm0, &pwmChan0) != kStatus_Success)
{
 PRINTF("PWM chan0 setup failed\n");
 return 1;
}

/* Start the PWM generation channel 0 */
PWM_StartTimer (BOARD_PWM_BASEADDR, kPWM_Pwm0);

while (1U)
{
 ;
}
```

## Files

- file [fsl\\_pwm.h](#)

## Data Structures

- struct [pwm\\_config\\_t](#)  
*PWM configuration structure. [More...](#)*
- struct [pwm\\_setup\\_t](#)  
*PWM channel setup structure. [More...](#)*

## Enumerations

- enum `pwm_channels_t` {  
`kPWM_Pwm0` = 0x0,  
`kPWM_Pwm1`,  
`kPWM_Pwm2`,  
`kPWM_Pwm3`,  
`kPWM_Pwm4`,  
`kPWM_Pwm5`,  
`kPWM_Pwm6`,  
`kPWM_Pwm7`,  
`kPWM_Pwm8`,  
`kPWM_Pwm9`,  
`kPWM_PwmAll` }  
*PWM channel selection values.*
- enum `pwm_polarity_control_t` {  
`kPWM_SetHighOnMatchLowOnPeriod` = 0x0,  
`kPWM_SetLowOnMatchHighOnPeriod` }  
*PWM channel polarity control values.*
- enum `pwm_dis_output_level_t` {  
`kPWM_SetLow` = 0x0,  
`kPWM_SetHigh` }  
*PWM channel disable output level values.*
- enum `pwm_interrupt_enable_t` {  
`kPWM_InterruptDisabled` = 0x0,  
`kPWM_InterruptEnabled` }  
*PWM channel interrupt enable flags.*
- enum `pwm_interrupt_status_t` {  
`kPWM_NoInterrupt` = 0x0,  
`kPWM_InterruptPendig` }  
*PWM channel interrupt status flags.*

## Driver version

- #define `FSL_PWM_DRIVER_VERSION` (`MAKE_VERSION(2, 0, 0)`)  
*PWM driver version 2.0.0.*

## Initialization and deinitialization

- void `PWM_GetDefaultConfig` (`pwm_config_t` \*userConfig)  
*Fill in the PWM config struct with the default settings.*
- `status_t` `PWM_Init` (`PWM_Type` \*base, const `pwm_config_t` \*userConfig)  
*Initializes the PWM module.*
- void `PWM_Deinit` (`PWM_Type` \*base)  
*Gate the PWM module clock.*

### PWM module output

- [status\\_t PWM\\_SetupPwm](#) (PWM\_Type \*base, [pwm\\_channels\\_t](#) pwm\_chan, [pwm\\_setup\\_t](#) \*pwm-Setup)  
*Sets up the PWM channel.*

### PWM Interrupts Interface

- static void [PWM\\_EnableInterrupts](#) (PWM\_Type \*base, [pwm\\_channels\\_t](#) pwm\_chan)  
*Enable PWM channel interrupt.*
- static void [PWM\\_DisableInterrupts](#) (PWM\_Type \*base, [pwm\\_channels\\_t](#) pwm\_chan)  
*Disable PWM channel interrupt.*
- static uint32\_t [PWM\\_GetEnabledInterrupts](#) (PWM\_Type \*base, [pwm\\_channels\\_t](#) pwm\_chan)  
*Gets the enabled PWM interrupts.*

### Status Interface

- uint32\_t [PWM\\_GetStatusFlags](#) (PWM\_Type \*base, [pwm\\_channels\\_t](#) pwm\_chan)  
*Gets the PWM status flags.*
- void [PWM\\_ClearStatusFlags](#) (PWM\_Type \*base, [pwm\\_channels\\_t](#) pwm\_chan)  
*Clears the PWM status flags.*

### Timer Start and Stop

- static void [PWM\\_StartTimer](#) (PWM\_Type \*base, [pwm\\_channels\\_t](#) pwm\_chan)  
*Start PWM channel.*
- static void [PWM\\_StopTimer](#) (PWM\_Type \*base, [pwm\\_channels\\_t](#) pwm\_chan)  
*Stop PWM channel.*
- uint16\_t [PWM\\_ReadPeriodValue](#) (PWM\_Type \*base, [pwm\\_channels\\_t](#) pwm\_chan)  
*Read current period value for PWM channel.*
- uint16\_t [PWM\\_ReadCompareValue](#) (PWM\_Type \*base, [pwm\\_channels\\_t](#) pwm\_chan)  
*Read compare match value for PWM channel.*

## 26.3 Data Structure Documentation

### 26.3.1 struct pwm\_config\_t

#### Data Fields

- [pwm\\_clock\\_source\\_t clk\\_sel](#)  
*PWM clock select value.*

### 26.3.2 struct pwm\_setup\_t

#### Data Fields

- [pwm\\_polarity\\_control\\_t pol\\_ctrl](#)  
*Channel polarity control.*

- [pwm\\_dis\\_output\\_level\\_t dis\\_out\\_level](#)  
*Channel disable output level.*
- [uint16\\_t prescaler\\_val](#)  
*Channel Prescaler value.*
- [uint16\\_t period\\_val](#)  
*Channel PWM period value.*
- [uint16\\_t comp\\_val](#)  
*Channel compare match value.*

### 26.3.2.0.0.27 Field Documentation

#### 26.3.2.0.0.27.1 `uint16_t pwm_setup_t::prescaler_val`

- 10 bit value

## 26.4 Macro Definition Documentation

### 26.4.1 `#define FSL_PWM_DRIVER_VERSION (MAKE_VERSION(2, 0, 0))`

## 26.5 Enumeration Type Documentation

### 26.5.1 `enum pwm_channels_t`

Enumerator

- kPWM\_Pwm0*** Channel 0.
- kPWM\_Pwm1*** Channel 1.
- kPWM\_Pwm2*** Channel 2.
- kPWM\_Pwm3*** Channel 3.
- kPWM\_Pwm4*** Channel 4.
- kPWM\_Pwm5*** Channel 5.
- kPWM\_Pwm6*** Channel 6.
- kPWM\_Pwm7*** Channel 7.
- kPWM\_Pwm8*** Channel 8.
- kPWM\_Pwm9*** Channel 9.
- kPWM\_PwmAll*** Channel 10 - All the channels will output same output programmed in this channel.

### 26.5.2 `enum pwm_polarity_control_t`

Enumerator

- kPWM\_SetHighOnMatchLowOnPeriod*** Set high on compare match, set low at end of PWM period.
- kPWM\_SetLowOnMatchHighOnPeriod*** Set low on compare match, set high at end of PWM period.

## Function Documentation

### 26.5.3 enum pwm\_dis\_output\_level\_t

Enumerator

*kPWM\_SetLow* Set to Low level.  
*kPWM\_SetHigh* Set to High level.

### 26.5.4 enum pwm\_interrupt\_enable\_t

Enumerator

*kPWM\_InterruptDisabled* PWM channel interrupt disabled.  
*kPWM\_InterruptEnabled* PWM channel interrupt enabled.

### 26.5.5 enum pwm\_interrupt\_status\_t

Enumerator

*kPWM\_NoInterrupt* PWM channel interrupt not occurred.  
*kPWM\_InterruptPending* PWM channel interrupt pending.

## 26.6 Function Documentation

### 26.6.1 void PWM\_GetDefaultConfig ( pwm\_config\_t \* userConfig )

The default values are:

```
* userConfig->clk_sel = kPWM_Osc32Mclk;
*
```

Parameters

|                   |                                         |
|-------------------|-----------------------------------------|
| <i>userConfig</i> | Pointer to user's PWM config structure. |
|-------------------|-----------------------------------------|

### 26.6.2 status\_t PWM\_Init ( PWM\_Type \* base, const pwm\_config\_t \* userConfig )

Call this API to ungate the PWM clock and configure the PWM HW.



## Note

This API should be called at the beginning of the application to use the PWM driver, or any operation to the PWM module could cause hard fault because PWM module clock is not enabled. The configuration structure can be filled by user from scratch, or be set with default values by [PWM\\_GetDefaultConfig\(\)](#). After calling this API, the application can configure PWM channels to generate PWM outputs. Example:

```
* pwm_config_t userConfig = {
* .clk_sel = kPWM_Fro48Mclk,
* };
* PWM_Init(PWM, &userConfig);
*
```

## Parameters

|                   |                                         |
|-------------------|-----------------------------------------|
| <i>base</i>       | PWM base address                        |
| <i>userConfig</i> | pointer to user configuration structure |

## Returns

kStatus\_Success - Success  
kStatus\_InvalidArgument - Invalid input parameter

### 26.6.3 void PWM\_Deinit ( PWM\_Type \* *base* )

## Parameters

|             |                  |
|-------------|------------------|
| <i>base</i> | PWM base address |
|-------------|------------------|

### 26.6.4 status\_t PWM\_SetupPwm ( PWM\_Type \* *base*, pwm\_channels\_t *pwm\_chan*, pwm\_setup\_t \* *pwmSetup* )

The function initializes the PWM channel according to the parameters passed in by the user. The function sets up the PWM compare match register & period registers.

## Parameters

## Function Documentation

|                 |                                     |
|-----------------|-------------------------------------|
| <i>base</i>     | PWM base address                    |
| <i>pwm_chan</i> | PWM channel select value            |
| <i>pwmSetup</i> | Pointer to PWM user setup structure |

### Returns

kStatus\_Success - Success

kStatus\_InvalidArgument - Invalid input parameter

### 26.6.5 static void PWM\_EnableInterrupts ( PWM\_Type \* *base*, pwm\_channels\_t *pwm\_chan* ) [inline], [static]

This function enables the interrupt for the specified PWM channel.

#### Parameters

|                 |                          |
|-----------------|--------------------------|
| <i>base</i>     | PWM base address         |
| <i>pwm_chan</i> | PWM channel select value |

### 26.6.6 static void PWM\_DisableInterrupts ( PWM\_Type \* *base*, pwm\_channels\_t *pwm\_chan* ) [inline], [static]

This function disables the interrupt for the specified PWM channel.

#### Parameters

|                 |                          |
|-----------------|--------------------------|
| <i>base</i>     | PWM base address         |
| <i>pwm_chan</i> | PWM channel select value |

### 26.6.7 static uint32\_t PWM\_GetEnabledInterrupts ( PWM\_Type \* *base*, pwm\_channels\_t *pwm\_chan* ) [inline], [static]

#### Parameters

---

|                 |                          |
|-----------------|--------------------------|
| <i>base</i>     | PWM base address         |
| <i>pwm_chan</i> | PWM channel select value |

Returns

PWM interrupt enabled status. This is the one of the values specified in enumeration [pwm\\_interrupt\\_enable\\_t](#)

### 26.6.8 uint32\_t PWM\_GetStatusFlags ( PWM\_Type \* *base*, pwm\_channels\_t *pwm\_chan* )

Parameters

|                 |                          |
|-----------------|--------------------------|
| <i>base</i>     | PWM base address         |
| <i>pwm_chan</i> | PWM channel select value |

Returns

The status flags. This is the one of the value of members of the enumeration [pwm\\_interrupt\\_status\\_t](#)

### 26.6.9 void PWM\_ClearStatusFlags ( PWM\_Type \* *base*, pwm\_channels\_t *pwm\_chan* )

Parameters

|                 |                          |
|-----------------|--------------------------|
| <i>base</i>     | PWM base address         |
| <i>pwm_chan</i> | PWM channel select value |

### 26.6.10 static void PWM\_StartTimer ( PWM\_Type \* *base*, pwm\_channels\_t *pwm\_chan* ) [inline], [static]

The API will start PWM channel output on the pin. Before calling this API, make sure that the PWM channel is set up using [PWM\\_SetupPwm\(\)](#) API.

## Function Documentation

### Parameters

|                 |                          |
|-----------------|--------------------------|
| <i>base</i>     | PWM base address         |
| <i>pwm_chan</i> | PWM channel select value |

### 26.6.11 static void PWM\_StopTimer ( PWM\_Type \* *base*, pwm\_channels\_t *pwm\_chan* ) [inline], [static]

The API will stop PWM channel output on the pin.

### Parameters

|                 |                          |
|-----------------|--------------------------|
| <i>base</i>     | PWM base address         |
| <i>pwm_chan</i> | PWM channel select value |

### 26.6.12 uint16\_t PWM\_ReadPeriodValue ( PWM\_Type \* *base*, pwm\_channels\_t *pwm\_chan* )

The API will read the current period value set for the PWM channel.

### Parameters

|                 |                          |
|-----------------|--------------------------|
| <i>base</i>     | PWM base address         |
| <i>pwm_chan</i> | PWM channel select value |

### Returns

16-bit period value

### 26.6.13 uint16\_t PWM\_ReadCompareValue ( PWM\_Type \* *base*, pwm\_channels\_t *pwm\_chan* )

The API will read the compare match value set for the PWM channel.

### Parameters

|                 |                          |
|-----------------|--------------------------|
| <i>base</i>     | PWM base address         |
| <i>pwm_chan</i> | PWM channel select value |

Returns

16-bit period value



## Chapter 27

### RNG: Random Number generator

#### 27.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Random Number generator module of MCU-Xpresso SDK devices.

#### Files

- file [fsl\\_rng.h](#)

#### Enumerations

- enum [trng\\_mode\\_t](#) {  
    [trng\\_UpdateOnce](#) = 0x1,  
    [trng\\_FreeRunning](#) = 0x2 }

#### Functions

- [status\\_t TRNG\\_GetDefaultConfig](#) ([trng\\_config\\_t](#) \*userConfig)  
    *Gets Default config of TRNG.*
- [status\\_t TRNG\\_Init](#) ([RNG\\_Type](#) \*base, const [trng\\_config\\_t](#) \*userConfig)  
    *Initializes the TRNG.*
- void [TRNG\\_Deinit](#) ([RNG\\_Type](#) \*base)  
    *Shuts down the TRNG.*
- [status\\_t TRNG\\_GetRandomData](#) ([RNG\\_Type](#) \*base, void \*data, [size\\_t](#) data\_size)  
    *Gets random data.*

#### 27.2 Enumeration Type Documentation

##### 27.2.1 enum [trng\\_mode\\_t](#)

RNG return status types RNG operating modes

Enumerator

*[trng\\_UpdateOnce](#)* TRNG update once & disable.  
*[trng\\_FreeRunning](#)* TRNG updates continuously.

#### 27.3 Function Documentation

##### 27.3.1 [status\\_t TRNG\\_GetDefaultConfig](#) ( [trng\\_config\\_t](#) \* *userConfig* )

This function initializes the TRNG configuration structure.

## Function Documentation

### Parameters

|                   |                                         |
|-------------------|-----------------------------------------|
| <i>userConfig</i> | Pointer to TRNG configuration structure |
|-------------------|-----------------------------------------|

### 27.3.2 status\_t TRNG\_Init ( RNG\_Type \* *base*, const trng\_config\_t \* *userConfig* )

This function initializes the TRNG.

### Parameters

|                   |                           |
|-------------------|---------------------------|
| <i>base</i>       | TRNG base address         |
| <i>userConfig</i> | The configuration of TRNG |

### Returns

kStatus\_Success - Success kStatus\_InvalidArgument - Invalid parameter

### 27.3.3 void TRNG\_Deinit ( RNG\_Type \* *base* )

This function shuts down the TRNG.

### Parameters

|             |                   |
|-------------|-------------------|
| <i>base</i> | TRNG base address |
|-------------|-------------------|

### 27.3.4 status\_t TRNG\_GetRandomData ( RNG\_Type \* *base*, void \* *data*, size\_t *data\_size* )

This function gets random data from the TRNG.

### Parameters

|             |                                                    |
|-------------|----------------------------------------------------|
| <i>base</i> | TRNG base address                                  |
| <i>data</i> | pointer to user buffer to be filled by random data |



|                  |                       |
|------------------|-----------------------|
| <i>data_size</i> | size of data in bytes |
|------------------|-----------------------|

Returns

TRNG status



## Chapter 28

### RTC: Real Time Clock

#### 28.1 Overview

The MCUXpresso SDK provides a driver for the Real Time Clock (RTC).

#### 28.2 Function groups

The RTC driver supports operating the module as a time counter.

##### 28.2.1 Initialization and deinitialization

The function [RTC\\_Init\(\)](#) initializes the RTC with specified configurations. The function [RTC\\_GetDefaultConfig\(\)](#) gets the default configurations.

The function [RTC\\_Deinit\(\)](#) disables the RTC timer and disables the module clock.

##### 28.2.2 Set & Get Datetime

The function [RTC\\_SetDatetime\(\)](#) sets the timer period in seconds. User passes in the details in date & time format by using the below data structure.

Refer to the driver examples codes located at `<SDK_ROOT>/boards/<BOARD>/driver_examples/rtc`  
The function [RTC\\_GetDatetime\(\)](#) reads the current timer value in seconds, converts it to date & time format and stores it into a datetime structure passed in by the user.

##### 28.2.3 Set & Get Alarm

The function [RTC\\_SetAlarm\(\)](#) sets the alarm time period in seconds. User passes in the details in date & time format by using the datetime data structure.

The function [RTC\\_GetAlarm\(\)](#) reads the alarm time in seconds, converts it to date & time format and stores it into a datetime structure passed in by the user.

##### 28.2.4 Start & Stop timer

The function [RTC\\_StartTimer\(\)](#) starts the RTC time counter.

The function [RTC\\_StopTimer\(\)](#) stops the RTC time counter.

## Typical use case

### 28.2.5 Status

Provides functions to get and clear the RTC status.

### 28.2.6 Interrupt

Provides functions to enable/disable RTC interrupts and get current enabled interrupts.

### 28.2.7 High resolution timer

Provides functions to enable high resolution timer and set and get the wake time.

## 28.3 Typical use case

### 28.3.1 RTC tick example

Example to set the RTC current time and trigger an alarm. Refer to the driver examples codes located at `<SDK_ROOT>/boards/<BOARD>/driver_examples/rtc`

## Files

- file [fsl\\_rtc.h](#)

## Data Structures

- struct [rtc\\_datetime\\_t](#)  
*Structure is used to hold the date and time. [More...](#)*

## Enumerations

- enum [rtc\\_interrupt\\_enable\\_t](#) {  
    [kRTC\\_AlarmInterruptEnable](#) = RTC\_CTRL\_ALARMDPD\_EN\_MASK,  
    [kRTC\\_WakeupInterruptEnable](#) = RTC\_CTRL\_WAKEDPD\_EN\_MASK }  
*List of RTC interrupts.*
- enum [rtc\\_status\\_flags\\_t](#) {  
    [kRTC\\_AlarmFlag](#) = RTC\_CTRL\_ALARM1HZ\_MASK,  
    [kRTC\\_WakeupFlag](#) = RTC\_CTRL\_WAKE1KHZ\_MASK }  
*List of RTC flags.*

## Functions

- static void [RTC\\_SetWakeupCount](#) (RTC\_Type \*base, uint16\_t wakeupValue)  
*Enable the RTC high resolution timer and set the wake-up time.*
- static uint16\_t [RTC\\_GetWakeupCount](#) (RTC\_Type \*base)  
*Read actual RTC counter value.*

- static void [RTC\\_Reset](#) (RTC\_Type \*base)  
*Performs a software reset on the RTC module.*

## Driver version

- #define [FSL\\_RTC\\_DRIVER\\_VERSION](#) ([MAKE\\_VERSION](#)(2, 0, 0))  
*Version 2.0.0.*

## Initialization and deinitialization

- void [RTC\\_Init](#) (RTC\_Type \*base)  
*Ungates the RTC clock and enables the RTC oscillator.*
- static void [RTC\\_Deinit](#) (RTC\_Type \*base)  
*Stop the timer and gate the RTC clock.*

## Current Time & Alarm

- [status\\_t RTC\\_SetDatetime](#) (RTC\_Type \*base, const [rtc\\_datetime\\_t](#) \*datetime)  
*Sets the RTC date and time according to the given time structure.*
- void [RTC\\_GetDatetime](#) (RTC\_Type \*base, [rtc\\_datetime\\_t](#) \*datetime)  
*Gets the RTC time and stores it in the given time structure.*
- [status\\_t RTC\\_SetAlarm](#) (RTC\_Type \*base, const [rtc\\_datetime\\_t](#) \*alarmTime)  
*Sets the RTC alarm time.*
- void [RTC\\_GetAlarm](#) (RTC\_Type \*base, [rtc\\_datetime\\_t](#) \*datetime)  
*Returns the RTC alarm time.*

## Interrupt Interface

- static void [RTC\\_EnableInterrupts](#) (RTC\_Type \*base, uint32\_t mask)  
*Enables the selected RTC interrupts.*
- static void [RTC\\_DisableInterrupts](#) (RTC\_Type \*base, uint32\_t mask)  
*Disables the selected RTC interrupts.*
- static uint32\_t [RTC\\_GetEnabledInterrupts](#) (RTC\_Type \*base)  
*Gets the enabled RTC interrupts.*

## Status Interface

- static uint32\_t [RTC\\_GetStatusFlags](#) (RTC\_Type \*base)  
*Gets the RTC status flags.*
- static void [RTC\\_ClearStatusFlags](#) (RTC\_Type \*base, uint32\_t mask)  
*Clears the RTC status flags.*

## Timer Start and Stop

- static void [RTC\\_StartTimer](#) (RTC\_Type \*base)  
*Starts the RTC time counter.*
- static void [RTC\\_StopTimer](#) (RTC\_Type \*base)  
*Stops the RTC time counter.*

## Enumeration Type Documentation

### 28.4 Data Structure Documentation

#### 28.4.1 struct rtc\_datetime\_t

##### Data Fields

- uint16\_t [year](#)  
*Range from 1970 to 2099.*
- uint8\_t [month](#)  
*Range from 1 to 12.*
- uint8\_t [day](#)  
*Range from 1 to 31 (depending on month).*
- uint8\_t [hour](#)  
*Range from 0 to 23.*
- uint8\_t [minute](#)  
*Range from 0 to 59.*
- uint8\_t [second](#)  
*Range from 0 to 59.*

##### 28.4.1.0.0.28 Field Documentation

28.4.1.0.0.28.1 uint16\_t rtc\_datetime\_t::year

28.4.1.0.0.28.2 uint8\_t rtc\_datetime\_t::month

28.4.1.0.0.28.3 uint8\_t rtc\_datetime\_t::day

28.4.1.0.0.28.4 uint8\_t rtc\_datetime\_t::hour

28.4.1.0.0.28.5 uint8\_t rtc\_datetime\_t::minute

28.4.1.0.0.28.6 uint8\_t rtc\_datetime\_t::second

### 28.5 Enumeration Type Documentation

#### 28.5.1 enum rtc\_interrupt\_enable\_t

Enumerator

*kRTC\_AlarmInterruptEnable* Alarm interrupt.

*kRTC\_WakeupInterruptEnable* Wake-up interrupt.

#### 28.5.2 enum rtc\_status\_flags\_t

Enumerator

*kRTC\_AlarmFlag* Alarm flag.

*kRTC\_WakeupFlag* 1kHz wake-up timer flag

## 28.6 Function Documentation

### 28.6.1 void RTC\_Init ( RTC\_Type \* *base* )

Note

This API should be called at the beginning of the application using the RTC driver.

Parameters

|             |                             |
|-------------|-----------------------------|
| <i>base</i> | RTC peripheral base address |
|-------------|-----------------------------|

### 28.6.2 static void RTC\_Deinit ( RTC\_Type \* *base* ) [inline], [static]

Parameters

|             |                             |
|-------------|-----------------------------|
| <i>base</i> | RTC peripheral base address |
|-------------|-----------------------------|

### 28.6.3 status\_t RTC\_SetDatetime ( RTC\_Type \* *base*, const rtc\_datetime\_t \* *datetime* )

The RTC counter must be stopped prior to calling this function as writes to the RTC seconds register will fail if the RTC counter is running.

Parameters

|                 |                                                                        |
|-----------------|------------------------------------------------------------------------|
| <i>base</i>     | RTC peripheral base address                                            |
| <i>datetime</i> | Pointer to structure where the date and time details to set are stored |

Returns

kStatus\_Success: Success in setting the time and starting the RTC  
 kStatus\_InvalidArgument: Error because the datetime format is incorrect

### 28.6.4 void RTC\_GetDatetime ( RTC\_Type \* *base*, rtc\_datetime\_t \* *datetime* )

## Function Documentation

### Parameters

|                 |                                                                  |
|-----------------|------------------------------------------------------------------|
| <i>base</i>     | RTC peripheral base address                                      |
| <i>datetime</i> | Pointer to structure where the date and time details are stored. |

### 28.6.5 **status\_t RTC\_SetAlarm ( RTC\_Type \* *base*, const rtc\_datetime\_t \* *alarmTime* )**

The function checks whether the specified alarm time is greater than the present time. If not, the function does not set the alarm and returns an error.

### Parameters

|                  |                                                      |
|------------------|------------------------------------------------------|
| <i>base</i>      | RTC peripheral base address                          |
| <i>alarmTime</i> | Pointer to structure where the alarm time is stored. |

### Returns

kStatus\_Success: success in setting the RTC alarm  
kStatus\_InvalidArgument: Error because the alarm datetime format is incorrect  
kStatus\_Fail: Error because the alarm time has already passed

### 28.6.6 **void RTC\_GetAlarm ( RTC\_Type \* *base*, rtc\_datetime\_t \* *datetime* )**

### Parameters

|                 |                                                                        |
|-----------------|------------------------------------------------------------------------|
| <i>base</i>     | RTC peripheral base address                                            |
| <i>datetime</i> | Pointer to structure where the alarm date and time details are stored. |

### 28.6.7 **static void RTC\_SetWakeupCount ( RTC\_Type \* *base*, uint16\_t *wakeupValue* ) [inline], [static]**

### Parameters



|                    |                                                   |
|--------------------|---------------------------------------------------|
| <i>base</i>        | RTC peripheral base address                       |
| <i>wakeupValue</i> | The value to be loaded into the RTC WAKE register |

**28.6.8 static uint16\_t RTC\_GetWakeupCount ( RTC\_Type \* *base* ) [inline], [static]**

Parameters

|             |                             |
|-------------|-----------------------------|
| <i>base</i> | RTC peripheral base address |
|-------------|-----------------------------|

**28.6.9 static void RTC\_EnableInterrupts ( RTC\_Type \* *base*, uint32\_t *mask* ) [inline], [static]**

Parameters

|             |                                                                                                                     |
|-------------|---------------------------------------------------------------------------------------------------------------------|
| <i>base</i> | RTC peripheral base address                                                                                         |
| <i>mask</i> | The interrupts to enable. This is a logical OR of members of the enumeration <a href="#">rtc_interrupt_enable_t</a> |

**28.6.10 static void RTC\_DisableInterrupts ( RTC\_Type \* *base*, uint32\_t *mask* ) [inline], [static]**

Parameters

|             |                                                                                                                     |
|-------------|---------------------------------------------------------------------------------------------------------------------|
| <i>base</i> | RTC peripheral base address                                                                                         |
| <i>mask</i> | The interrupts to enable. This is a logical OR of members of the enumeration <a href="#">rtc_interrupt_enable_t</a> |

**28.6.11 static uint32\_t RTC\_GetEnabledInterrupts ( RTC\_Type \* *base* ) [inline], [static]**

## Function Documentation

### Parameters

|             |                             |
|-------------|-----------------------------|
| <i>base</i> | RTC peripheral base address |
|-------------|-----------------------------|

### Returns

The enabled interrupts. This is the logical OR of members of the enumeration [rtc\\_interrupt\\_enable\\_t](#)

### 28.6.12 static uint32\_t RTC\_GetStatusFlags ( RTC\_Type \* *base* ) [inline], [static]

### Parameters

|             |                             |
|-------------|-----------------------------|
| <i>base</i> | RTC peripheral base address |
|-------------|-----------------------------|

### Returns

The status flags. This is the logical OR of members of the enumeration [rtc\\_status\\_flags\\_t](#)

### 28.6.13 static void RTC\_ClearStatusFlags ( RTC\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

### Parameters

|             |                                                                                                                  |
|-------------|------------------------------------------------------------------------------------------------------------------|
| <i>base</i> | RTC peripheral base address                                                                                      |
| <i>mask</i> | The status flags to clear. This is a logical OR of members of the enumeration <a href="#">rtc_status_flags_t</a> |

### 28.6.14 static void RTC\_StartTimer ( RTC\_Type \* *base* ) [inline], [static]

After calling this function, the timer counter increments once a second provided SR[TOF] or SR[TIF] are not set.

### Parameters

---

|             |                             |
|-------------|-----------------------------|
| <i>base</i> | RTC peripheral base address |
|-------------|-----------------------------|

#### 28.6.15 static void RTC\_StopTimer ( RTC\_Type \* *base* ) [inline], [static]

RTC's seconds register can be written to only when the timer is stopped.

Parameters

|             |                             |
|-------------|-----------------------------|
| <i>base</i> | RTC peripheral base address |
|-------------|-----------------------------|

#### 28.6.16 static void RTC\_Reset ( RTC\_Type \* *base* ) [inline], [static]

This resets all RTC registers to their reset value. The bit is cleared by software explicitly clearing it.

Parameters

|             |                             |
|-------------|-----------------------------|
| <i>base</i> | RTC peripheral base address |
|-------------|-----------------------------|



## Chapter 29

### SPIFI: SPIFI flash interface driver

#### 29.1 Overview

##### Modules

- [SPIFI DMA Driver](#)
- [SPIFI Driver](#)

##### Data Structures

- struct [spifi\\_command\\_t](#)  
*SPIFI command structure. [More...](#)*
- struct [spifi\\_config\\_t](#)  
*SPIFI region configuration structure. [More...](#)*
- struct [spifi\\_transfer\\_t](#)  
*Transfer structure for SPIFI. [More...](#)*
- struct [spifi\\_dma\\_handle\\_t](#)  
*SPIFI DMA transfer handle, users should not touch the content of the handle. [More...](#)*

##### Typedefs

- typedef void(\* [spifi\\_dma\\_callback\\_t](#))(SPIFI\_Type \*base, spifi\_dma\_handle\_t \*handle, [status\\_t](#) status, void \*userData)  
*SPIFI DMA transfer callback function for finish and error.*

##### Enumerations

- enum [\\_status\\_t](#) {  
    [kStatus\\_SPIFI\\_Idle](#) = MAKE\_STATUS(kStatusGroup\_SPIFI, 0),  
    [kStatus\\_SPIFI\\_Busy](#) = MAKE\_STATUS(kStatusGroup\_SPIFI, 1),  
    [kStatus\\_SPIFI\\_Error](#) = MAKE\_STATUS(kStatusGroup\_SPIFI, 2) }  
*Status structure of SPIFI.*
- enum [spifi\\_interrupt\\_enable\\_t](#) { [kSPIFI\\_CommandFinishInterruptEnable](#) = SPIFI\_CTRL\_INTEN-  
    \_MASK }  
*SPIFI interrupt source.*
- enum [spifi\\_spi\\_mode\\_t](#) {  
    [kSPIFI\\_SPISckLow](#) = 0x0U,  
    [kSPIFI\\_SPISckHigh](#) = 0x1U }  
*SPIFI SPI mode select.*
- enum [spifi\\_dual\\_mode\\_t](#) {  
    [kSPIFI\\_QuadMode](#) = 0x0U,  
    [kSPIFI\\_DualMode](#) = 0x1U }  
*SPIFI dual mode select.*

## Overview

- enum `spifi_data_direction_t` {  
    `kSPIFI_DataInput` = 0x0U,  
    `kSPIFI_DataOutput` = 0x1U }  
    *SPIFI data direction.*
- enum `spifi_command_format_t` {  
    `kSPIFI_CommandAllSerial` = 0x0,  
    `kSPIFI_CommandDataQuad` = 0x1U,  
    `kSPIFI_CommandOpcodeSerial` = 0x2U,  
    `kSPIFI_CommandAllQuad` = 0x3U }  
    *SPIFI command opcode format.*
- enum `spifi_command_type_t` {  
    `kSPIFI_CommandOpcodeOnly` = 0x1U,  
    `kSPIFI_CommandOpcodeAddrOneByte` = 0x2U,  
    `kSPIFI_CommandOpcodeAddrTwoBytes` = 0x3U,  
    `kSPIFI_CommandOpcodeAddrThreeBytes` = 0x4U,  
    `kSPIFI_CommandOpcodeAddrFourBytes` = 0x5U,  
    `kSPIFI_CommandNoOpcodeAddrThreeBytes` = 0x6U,  
    `kSPIFI_CommandNoOpcodeAddrFourBytes` = 0x7U }  
    *SPIFI command type.*
- enum `_spifi_status_flags` {  
    `kSPIFI_MemoryCommandWriteFinished` = `SPIFI_STAT_MCINIT_MASK`,  
    `kSPIFI_CommandWriteFinished` = `SPIFI_STAT_CMD_MASK`,  
    `kSPIFI_InterruptRequest` = `SPIFI_STAT_INTRQ_MASK` }  
    *SPIFI status flags.*

## Functions

- static void `SPIFI_EnableDMA` (`SPIFI_Type *base`, bool enable)  
    *Enable or disable DMA request for SPIFI.*
- static uint32\_t `SPIFI_GetDataRegisterAddress` (`SPIFI_Type *base`)  
    *Gets the SPIFI data register address.*
- static void `SPIFI_WriteData` (`SPIFI_Type *base`, uint32\_t data)  
    *Write a word data in address of SPIFI.*
- void `SPIFI_WriteBuffer` (`SPIFI_Type *base`, uint8\_t \*buf, size\_t size\_to\_write)  
    *Write a buffer worth of data to SPIFI.*
- static void `SPIFI_WriteDataByte` (`SPIFI_Type *base`, uint8\_t data)  
    *Write a byte data in address of SPIFI.*
- void `SPIFI_WriteDataHalfword` (`SPIFI_Type *base`, uint16\_t data)  
    *Write a halfword data in address of SPIFI.*
- static uint32\_t `SPIFI_ReadData` (`SPIFI_Type *base`)  
    *Read data from serial flash.*
- static uint8\_t `SPIFI_ReadDataByte` (`SPIFI_Type *base`)  
    *Read a byte data from serial flash.*
- uint16\_t `SPIFI_ReadDataHalfword` (`SPIFI_Type *base`)  
    *Read a halfword data from serial flash.*

## Driver version

- #define `FSL_SPIFI_DRIVER_VERSION` (`MAKE_VERSION(2, 0, 2)`)

*SPIFI driver version 2.0.2.*

## Initialization and deinitialization

- uint32\_t [SPIFI\\_GetInstance](#) (SPIFI\_Type \*base)  
*Get the SPIFI instance from peripheral base address.*
- void [SPIFI\\_Init](#) (SPIFI\_Type \*base, const [spifi\\_config\\_t](#) \*config)  
*Initializes the SPIFI with the user configuration structure.*
- void [SPIFI\\_GetDefaultConfig](#) ([spifi\\_config\\_t](#) \*config)  
*Get SPIFI default configure settings.*
- void [SPIFI\\_Deinit](#) (SPIFI\_Type \*base)  
*Deinitializes the SPIFI regions.*

## Basic Control Operations

- void [SPIFI\\_SetCommand](#) (SPIFI\_Type \*base, const [spifi\\_command\\_t](#) \*cmd)  
*Set SPIFI flash command.*
- static void [SPIFI\\_SetCommandAddress](#) (SPIFI\_Type \*base, uint32\_t addr)  
*Set SPIFI command address.*
- static void [SPIFI\\_SetIntermediateData](#) (SPIFI\_Type \*base, uint32\_t val)  
*Set SPIFI intermediate data.*
- static void [SPIFI\\_SetCacheLimit](#) (SPIFI\_Type \*base, uint32\_t val)  
*Set SPIFI Cache limit value.*
- static void [SPIFI\\_ResetCommand](#) (SPIFI\_Type \*base)  
*Reset the command field of SPIFI.*
- void [SPIFI\\_SetMemoryCommand](#) (SPIFI\_Type \*base, const [spifi\\_command\\_t](#) \*cmd)  
*Set SPIFI flash AHB read command.*
- static void [SPIFI\\_EnableInterrupt](#) (SPIFI\_Type \*base, uint32\_t mask)  
*Enable SPIFI interrupt.*
- static void [SPIFI\\_DisableInterrupt](#) (SPIFI\_Type \*base, uint32\_t mask)  
*Disable SPIFI interrupt.*

## Status

- static uint32\_t [SPIFI\\_GetStatusFlag](#) (SPIFI\_Type \*base)  
*Get the status of all interrupt flags for SPIFI.*

## Driver version

- #define [FSL\\_SPIFI\\_DMA\\_DRIVER\\_VERSION](#) (MAKE\_VERSION(2, 0, 2))  
*SPIFI DMA driver version 2.0.2.*

## DMA Transactional

- void [SPIFI\\_TransferTxCreateHandleDMA](#) (SPIFI\_Type \*base, spifi\_dma\_handle\_t \*handle, [spifi\\_dma\\_callback\\_t](#) callback, void \*userData, [dma\\_handle\\_t](#) \*dmaHandle)  
*Initializes the SPIFI handle for send which is used in transactional functions and set the callback.*
- void [SPIFI\\_TransferRxCreateHandleDMA](#) (SPIFI\_Type \*base, spifi\_dma\_handle\_t \*handle, [spifi\\_dma\\_callback\\_t](#) callback, void \*userData, [dma\\_handle\\_t](#) \*dmaHandle)  
*Initializes the SPIFI handle for receive which is used in transactional functions and set the callback.*

## Data Structure Documentation

- [status\\_t SPIFI\\_TransferSendDMA](#) (SPIFI\_Type \*base, spifi\_dma\_handle\_t \*handle, [spifi\\_transfer\\_t](#) \*xfer)  
*Transfers SPIFI data using an DMA non-blocking method.*
- [status\\_t SPIFI\\_TransferReceiveDMA](#) (SPIFI\_Type \*base, spifi\_dma\_handle\_t \*handle, [spifi\\_transfer\\_t](#) \*xfer)  
*Receives data using an DMA non-blocking method.*
- void [SPIFI\\_TransferAbortSendDMA](#) (SPIFI\_Type \*base, spifi\_dma\_handle\_t \*handle)  
*Aborts the sent data using DMA.*
- void [SPIFI\\_TransferAbortReceiveDMA](#) (SPIFI\_Type \*base, spifi\_dma\_handle\_t \*handle)  
*Aborts the receive data using DMA.*
- [status\\_t SPIFI\\_TransferGetSendCountDMA](#) (SPIFI\_Type \*base, spifi\_dma\_handle\_t \*handle, size\_t \*count)  
*Gets the transferred counts of send.*
- [status\\_t SPIFI\\_TransferGetReceiveCountDMA](#) (SPIFI\_Type \*base, spifi\_dma\_handle\_t \*handle, size\_t \*count)  
*Gets the status of the receive transfer.*

## 29.2 Data Structure Documentation

### 29.2.1 struct spifi\_command\_t

#### Data Fields

- uint16\_t [dataLen](#)  
*How many data bytes are needed in this command.*
- bool [isPollMode](#)  
*For command need to read data from serial flash.*
- [spifi\\_data\\_direction\\_t](#) [direction](#)  
*Data direction of this command.*
- uint8\_t [intermediateBytes](#)  
*How many intermediate bytes needed.*
- [spifi\\_command\\_format\\_t](#) [format](#)  
*Command format.*
- [spifi\\_command\\_type\\_t](#) [type](#)  
*Command type.*
- uint8\_t [opcode](#)  
*Command opcode value.*

#### 29.2.1.0.0.29 Field Documentation

##### 29.2.1.0.0.29.1 uint16\_t spifi\_command\_t::dataLen

##### 29.2.1.0.0.29.2 spifi\_data\_direction\_t spifi\_command\_t::direction

### 29.2.2 struct spifi\_config\_t

#### Data Fields

- uint16\_t [timeout](#)



- `uint8_t csHighTime`  
*SPI transfer timeout, the unit is SCK cycles.*
- `bool disablePrefetch`  
*CS high time cycles.*
- `bool disableCachePrefech`  
*True means SPIFI will not attempt a speculative prefetch.*
- `bool isFeedbackClock`  
*Disable prefetch of cache line.*
- `bool isFeedbackClock`  
*Is data sample uses feedback clock.*
- `spifi_spi_mode_t spiMode`  
*SPIFI spi mode select.*
- `bool isReadFullClockCycle`  
*If enable read full clock cycle.*
- `spifi_dual_mode_t dualMode`  
*SPIFI dual mode, dual or quad.*

### 29.2.2.0.0.30 Field Documentation

29.2.2.0.0.30.1 `bool spifi_config_t::disablePrefetch`

29.2.2.0.0.30.2 `bool spifi_config_t::isFeedbackClock`

29.2.2.0.0.30.3 `bool spifi_config_t::isReadFullClockCycle`

29.2.2.0.0.30.4 `spifi_dual_mode_t spifi_config_t::dualMode`

## 29.2.3 struct spifi\_transfer\_t

### Data Fields

- `uint8_t * data`  
*Pointer to data to transmit.*
- `size_t dataSize`  
*Bytes to be transmit.*

## 29.2.4 struct \_spifi\_dma\_handle

### Data Fields

- `dma_handle_t * dmaHandle`  
*DMA handler for SPIFI send.*
- `size_t transferSize`  
*Bytes need to transfer.*
- `uint32_t state`  
*Internal state for SPIFI DMA transfer.*
- `spifi_dma_callback_t callback`  
*Callback for users while transfer finish or error occurred.*
- `void * userData`

## Enumeration Type Documentation

*User callback parameter.*

### 29.2.4.0.0.31 Field Documentation

29.2.4.0.0.31.1 `size_t spifi_dma_handle_t::transferSize`

## 29.3 Macro Definition Documentation

29.3.1 `#define FSL_SPIFI_DRIVER_VERSION (MAKE_VERSION(2, 0, 2))`

29.3.2 `#define FSL_SPIFI_DMA_DRIVER_VERSION (MAKE_VERSION(2, 0, 2))`

## 29.4 Enumeration Type Documentation

### 29.4.1 `enum _status_t`

Enumerator

*kStatus\_SPIFI\_Idle* SPIFI is in idle state.

*kStatus\_SPIFI\_Busy* SPIFI is busy.

*kStatus\_SPIFI\_Error* Error occurred during SPIFI transfer.

### 29.4.2 `enum spifi_interrupt_enable_t`

Enumerator

*kSPIFI\_CommandFinishInterruptEnable* Interrupt while command finished.

### 29.4.3 `enum spifi_spi_mode_t`

Enumerator

*kSPIFI\_SPISckLow* SCK low after last bit of command, keeps low while CS high.

*kSPIFI\_SPISckHigh* SCK high after last bit of command and while CS high.

### 29.4.4 `enum spifi_dual_mode_t`

Enumerator

*kSPIFI\_QuadMode* SPIFI uses IO3:0.

*kSPIFI\_DualMode* SPIFI uses IO1:0.

### 29.4.5 enum spifi\_data\_direction\_t

Enumerator

***kSPIFI\_DataInput*** Data input from serial flash.

***kSPIFI\_DataOutput*** Data output to serial flash.

### 29.4.6 enum spifi\_command\_format\_t

Enumerator

***kSPIFI\_CommandAllSerial*** All fields of command are serial.

***kSPIFI\_CommandDataQuad*** Only data field is dual/quad, others are serial.

***kSPIFI\_CommandOpcodeSerial*** Only opcode field is serial, others are quad/dual.

***kSPIFI\_CommandAllQuad*** All fields of command are dual/quad mode.

### 29.4.7 enum spifi\_command\_type\_t

Enumerator

***kSPIFI\_CommandOpcodeOnly*** Command only have opcode, no address field.

***kSPIFI\_CommandOpcodeAddrOneByte*** Command have opcode and also one byte address field.

***kSPIFI\_CommandOpcodeAddrTwoBytes*** Command have opcode and also two bytes address field.

***kSPIFI\_CommandOpcodeAddrThreeBytes*** Command have opcode and also three bytes address field.

***kSPIFI\_CommandOpcodeAddrFourBytes*** Command have opcode and also four bytes address field.

***kSPIFI\_CommandNoOpcodeAddrThreeBytes*** Command have no opcode and three bytes address field.

***kSPIFI\_CommandNoOpcodeAddrFourBytes*** Command have no opcode and four bytes address field.

### 29.4.8 enum \_spifi\_status\_flags

Enumerator

***kSPIFI\_MemoryCommandWriteFinished*** Memory command write finished.

***kSPIFI\_CommandWriteFinished*** Command write finished.

***kSPIFI\_InterruptRequest*** CMD flag from 1 to 0, means command execute finished.

## 29.5 Function Documentation

### 29.5.1 uint32\_t SPIFI\_GetInstance ( SPIFI\_Type \* *base* )

## Parameters

|             |                                |
|-------------|--------------------------------|
| <i>base</i> | SPIFI peripheral base address. |
|-------------|--------------------------------|

## Returns

SPIFI instance.

### 29.5.2 void SPIFI\_Init ( SPIFI\_Type \* *base*, const spifi\_config\_t \* *config* )

This function configures the SPIFI module with the user-defined configuration.

## Parameters

|               |                                             |
|---------------|---------------------------------------------|
| <i>base</i>   | SPIFI peripheral base address.              |
| <i>config</i> | The pointer to the configuration structure. |

### 29.5.3 void SPIFI\_GetDefaultConfig ( spifi\_config\_t \* *config* )

## Parameters

|               |                                 |
|---------------|---------------------------------|
| <i>config</i> | SPIFI config structure pointer. |
|---------------|---------------------------------|

### 29.5.4 void SPIFI\_Deinit ( SPIFI\_Type \* *base* )

## Parameters

|             |                                |
|-------------|--------------------------------|
| <i>base</i> | SPIFI peripheral base address. |
|-------------|--------------------------------|

### 29.5.5 void SPIFI\_SetCommand ( SPIFI\_Type \* *base*, const spifi\_command\_t \* *cmd* )

## Function Documentation

### Parameters

|             |                                  |
|-------------|----------------------------------|
| <i>base</i> | SPIFI peripheral base address.   |
| <i>cmd</i>  | SPIFI command structure pointer. |

**29.5.6 static void SPIFI\_SetCommandAddress ( SPIFI\_Type \* *base*, uint32\_t *addr* ) [inline], [static]**

### Parameters

|             |                                |
|-------------|--------------------------------|
| <i>base</i> | SPIFI peripheral base address. |
| <i>addr</i> | Address value for the command. |

**29.5.7 static void SPIFI\_SetIntermediateData ( SPIFI\_Type \* *base*, uint32\_t *val* ) [inline], [static]**

Before writing a command which needs specific intermediate value, users shall call this function to write it. The main use of this function for current serial flash is to select no-opcode mode and cancelling this mode. As dummy cycle do not care about the value, no need to call this function.

### Parameters

|             |                                |
|-------------|--------------------------------|
| <i>base</i> | SPIFI peripheral base address. |
| <i>val</i>  | Intermediate data.             |

**29.5.8 static void SPIFI\_SetCacheLimit ( SPIFI\_Type \* *base*, uint32\_t *val* ) [inline], [static]**

SPIFI includes caching of previously-accessed data to improve performance. Software can write an address to this function, to prevent such caching at and above the address.

### Parameters

|             |                                |
|-------------|--------------------------------|
| <i>base</i> | SPIFI peripheral base address. |
|-------------|--------------------------------|

|            |                                             |
|------------|---------------------------------------------|
| <i>val</i> | Zero-based upper limit of cacheable memory. |
|------------|---------------------------------------------|

### 29.5.9 static void SPIFI\_ResetCommand ( SPIFI\_Type \* *base* ) [inline], [static]

This function is used to abort the current command or memory mode.

Parameters

|             |                                |
|-------------|--------------------------------|
| <i>base</i> | SPIFI peripheral base address. |
|-------------|--------------------------------|

### 29.5.10 void SPIFI\_SetMemoryCommand ( SPIFI\_Type \* *base*, const spifi\_command\_t \* *cmd* )

Call this function means SPIFI enters to memory mode, while users need to use command, a SPIFI\_Reset-Command shall be called.

Parameters

|             |                                  |
|-------------|----------------------------------|
| <i>base</i> | SPIFI peripheral base address.   |
| <i>cmd</i>  | SPIFI command structure pointer. |

### 29.5.11 static void SPIFI\_EnableInterrupt ( SPIFI\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

The interrupt is triggered only in command mode, and it means the command now is finished.

Parameters

|             |                                                                                                      |
|-------------|------------------------------------------------------------------------------------------------------|
| <i>base</i> | SPIFI peripheral base address.                                                                       |
| <i>mask</i> | SPIFI interrupt enable mask. It is a logic OR of members the enumeration :: spifi_interrupt_enable_t |

### 29.5.12 static void SPIFI\_DisableInterrupt ( SPIFI\_Type \* *base*, uint32\_t *mask* ) [inline], [static]

The interrupt is triggered only in command mode, and it means the command now is finished.

## Function Documentation

### Parameters

|             |                                                                                                      |
|-------------|------------------------------------------------------------------------------------------------------|
| <i>base</i> | SPIFI peripheral base address.                                                                       |
| <i>mask</i> | SPIFI interrupt enable mask. It is a logic OR of members the enumeration :: spifi_interrupt_enable_t |

### 29.5.13 static uint32\_t SPIFI\_GetStatusFlag ( SPIFI\_Type \* *base* ) [inline], [static]

### Parameters

|             |                                |
|-------------|--------------------------------|
| <i>base</i> | SPIFI peripheral base address. |
|-------------|--------------------------------|

### Returns

SPIFI flag status

### 29.5.14 static void SPIFI\_EnableDMA ( SPIFI\_Type \* *base*, bool *enable* ) [inline], [static]

### Parameters

|               |                                                    |
|---------------|----------------------------------------------------|
| <i>base</i>   | SPIFI peripheral base address.                     |
| <i>enable</i> | True means enable DMA and false means disable DMA. |

### 29.5.15 static uint32\_t SPIFI\_GetDataRegisterAddress ( SPIFI\_Type \* *base* ) [inline], [static]

This API is used to provide a transfer address for the SPIFI DMA transfer configuration.

### Parameters

|             |                    |
|-------------|--------------------|
| <i>base</i> | SPIFI base pointer |
|-------------|--------------------|

### Returns

data register address



### 29.5.16 static void SPIFI\_WriteData ( SPIFI\_Type \* *base*, uint32\_t *data* ) [inline], [static]

Users can write a page or at least a word data into SPIFI address. Beware: certain SPIFI implementations (such as that of JN5189/QN9090/K32W061) require that the data do not exceed the actual size of the command, so cannot call SPIFI\_WriteData when less than 32 bits are expected.

Parameters

|             |                                |
|-------------|--------------------------------|
| <i>base</i> | SPIFI peripheral base address. |
| <i>data</i> | Data that need to be written.  |

### 29.5.17 void SPIFI\_WriteBuffer ( SPIFI\_Type \* *base*, uint8\_t \* *buf*, size\_t *size\_to\_write* )

Used for transaction requiring less than 32 bits of data

Parameters

|                      |                                                     |
|----------------------|-----------------------------------------------------|
| <i>base</i>          | SPIFI peripheral base address.                      |
| <i>buf</i>           | pointer on octet buffer to be written to the SPIFI. |
| <i>size_to_write</i> | size of buffer.                                     |

### 29.5.18 static void SPIFI\_WriteDataByte ( SPIFI\_Type \* *base*, uint8\_t *data* ) [inline], [static]

Users can write a byte data into SPIFI address.

Parameters

|             |                                |
|-------------|--------------------------------|
| <i>base</i> | SPIFI peripheral base address. |
| <i>data</i> | Data need be write.            |

### 29.5.19 void SPIFI\_WriteDataHalfword ( SPIFI\_Type \* *base*, uint16\_t *data* )

Users can write a halfword data into SPIFI address.

## Function Documentation

### Parameters

|             |                                |
|-------------|--------------------------------|
| <i>base</i> | SPIFI peripheral base address. |
| <i>data</i> | Data need be write.            |

### 29.5.20 static uint32\_t SPIFI\_ReadData ( SPIFI\_Type \* *base* ) [inline], [static]

Users should notice before call this function, the data length field in command register shall be larger than 4, otherwise a hard fault will happen.

### Parameters

|             |                                |
|-------------|--------------------------------|
| <i>base</i> | SPIFI peripheral base address. |
|-------------|--------------------------------|

### Returns

Data input from flash.

### 29.5.21 static uint8\_t SPIFI\_ReadDataByte ( SPIFI\_Type \* *base* ) [inline], [static]

### Parameters

|             |                                |
|-------------|--------------------------------|
| <i>base</i> | SPIFI peripheral base address. |
|-------------|--------------------------------|

### Returns

Data input from flash.

### 29.5.22 uint16\_t SPIFI\_ReadDataHalfword ( SPIFI\_Type \* *base* )

### Parameters

---

|             |                                |
|-------------|--------------------------------|
| <i>base</i> | SPIFI peripheral base address. |
|-------------|--------------------------------|

Returns

Data input from flash.

**29.5.23 void SPIFI\_TransferTxCreateHandleDMA ( SPIFI\_Type \* *base*,  
spifi\_dma\_handle\_t \* *handle*, spifi\_dma\_callback\_t *callback*, void \*  
*userData*, dma\_handle\_t \* *dmaHandle* )**

Parameters

|                    |                                            |
|--------------------|--------------------------------------------|
| <i>base</i>        | SPIFI peripheral base address              |
| <i>handle</i>      | Pointer to spifi_dma_handle_t structure    |
| <i>callback</i>    | SPIFI callback, NULL means no callback.    |
| <i>userData</i>    | User callback function data.               |
| <i>rxDmaHandle</i> | User requested DMA handle for DMA transfer |

**29.5.24 void SPIFI\_TransferRxCreateHandleDMA ( SPIFI\_Type \* *base*,  
spifi\_dma\_handle\_t \* *handle*, spifi\_dma\_callback\_t *callback*, void \*  
*userData*, dma\_handle\_t \* *dmaHandle* )**

Parameters

|                    |                                            |
|--------------------|--------------------------------------------|
| <i>base</i>        | SPIFI peripheral base address              |
| <i>handle</i>      | Pointer to spifi_dma_handle_t structure    |
| <i>callback</i>    | SPIFI callback, NULL means no callback.    |
| <i>userData</i>    | User callback function data.               |
| <i>rxDmaHandle</i> | User requested DMA handle for DMA transfer |

**29.5.25 status\_t SPIFI\_TransferSendDMA ( SPIFI\_Type \* *base*, spifi\_dma\_handle\_t  
\* *handle*, spifi\_transfer\_t \* *xfer* )**

This function writes data to the SPIFI transmit FIFO. This function is non-blocking.

## Function Documentation

### Parameters

|               |                                         |
|---------------|-----------------------------------------|
| <i>base</i>   | Pointer to QuadSPI Type.                |
| <i>handle</i> | Pointer to spifi_dma_handle_t structure |
| <i>xfer</i>   | SPIFI transfer structure.               |

### 29.5.26 status\_t SPIFI\_TransferReceiveDMA ( SPIFI\_Type \* *base*, spifi\_dma\_handle\_t \* *handle*, spifi\_transfer\_t \* *xfer* )

This function receive data from the SPIFI receive buffer/FIFO. This function is non-blocking.

### Parameters

|               |                                         |
|---------------|-----------------------------------------|
| <i>base</i>   | Pointer to QuadSPI Type.                |
| <i>handle</i> | Pointer to spifi_dma_handle_t structure |
| <i>xfer</i>   | SPIFI transfer structure.               |

### 29.5.27 void SPIFI\_TransferAbortSendDMA ( SPIFI\_Type \* *base*, spifi\_dma\_handle\_t \* *handle* )

This function aborts the sent data using DMA.

### Parameters

|               |                                         |
|---------------|-----------------------------------------|
| <i>base</i>   | SPIFI peripheral base address.          |
| <i>handle</i> | Pointer to spifi_dma_handle_t structure |

### 29.5.28 void SPIFI\_TransferAbortReceiveDMA ( SPIFI\_Type \* *base*, spifi\_dma\_handle\_t \* *handle* )

This function abort receive data which using DMA.

### Parameters

|               |                                         |
|---------------|-----------------------------------------|
| <i>base</i>   | SPIFI peripheral base address.          |
| <i>handle</i> | Pointer to spifi_dma_handle_t structure |

**29.5.29** `status_t SPIFI_TransferGetSendCountDMA ( SPIFI_Type * base,  
spifi_dma_handle_t * handle, size_t * count )`

## Function Documentation

### Parameters

|               |                                          |
|---------------|------------------------------------------|
| <i>base</i>   | Pointer to QuadSPI Type.                 |
| <i>handle</i> | Pointer to spifi_dma_handle_t structure. |
| <i>count</i>  | Bytes sent.                              |

### Return values

|                                     |                                                                |
|-------------------------------------|----------------------------------------------------------------|
| <i>kStatus_Success</i>              | Succeed get the transfer count.                                |
| <i>kStatus_NoTransferInProgress</i> | There is not a non-blocking transaction currently in progress. |

### 29.5.30 status\_t SPIFI\_TransferGetReceiveCountDMA ( SPIFI\_Type \* *base*, spifi\_dma\_handle\_t \* *handle*, size\_t \* *count* )

### Parameters

|               |                                         |
|---------------|-----------------------------------------|
| <i>base</i>   | Pointer to QuadSPI Type.                |
| <i>handle</i> | Pointer to spifi_dma_handle_t structure |
| <i>count</i>  | Bytes received.                         |

### Return values

|                                     |                                                                |
|-------------------------------------|----------------------------------------------------------------|
| <i>kStatus_Success</i>              | Succeed get the transfer count.                                |
| <i>kStatus_NoTransferInProgress</i> | There is not a non-blocking transaction currently in progress. |

## 29.6 SPIFI Driver

SPIFI driver includes functional APIs.

Functional APIs are feature/property target low level APIs. Functional APIs can be used for SPIFI initialization/configuration/operation for optimization/customization purpose. Using the functional API requires the knowledge of the SPIFI peripheral and how to organize functional APIs to meet the application requirements. All functional API use the peripheral base address as the first parameter. SPIFI functional operation groups provide the functional API set.

### 29.6.1 Typical use case

#### 29.6.1.1 SPIFI transfer using an polling method

Refer to the driver examples codes located at `<SDK_ROOT>/boards/<BOARD>/driver_examples/spifi`

### 29.7 SPIFI DMA Driver

This chapter describes the programming interface of the SPIFI DMA driver. SPIFI DMA driver includes transactional APIs.

Transactional APIs are transaction target high level APIs. Transactional APIs can be used to enable the peripheral and in the application if the code size and performance of transactional APIs satisfy the requirements. If the code size and performance are a critical requirement, see the transactional API implementation and write a custom code. All transactional APIs use the `spifi_handle_t` as the first parameter. Initialize the handle by calling the `SPIFI_TransferCreateHandleDMA()` API.

#### 29.7.1 Typical use case

##### 29.7.1.1 SPIFI Send/receive using a DMA method

Refer to the driver examples codes located at `<SDK_ROOT>/boards/<BOARD>/driver_examples/spifi`



## Chapter 30

# WWDT: Windowed Watchdog Timer Driver

### 30.1 Overview

The MCUXpresso SDK provides a peripheral driver for the Watchdog module (WDOG) of MCUXpresso SDK devices.

### 30.2 Function groups

#### 30.2.1 Initialization and deinitialization

The function [WWDT\\_Init\(\)](#) initializes the watchdog timer with specified configurations. The configurations include timeout value and whether to enable watchdog after init. The function [WWDT\\_GetDefaultConfig\(\)](#) gets the default configurations.

The function [WWDT\\_Deinit\(\)](#) disables the watchdog and the module clock.

#### 30.2.2 Status

Provides functions to get and clear the WWDT status.

#### 30.2.3 Interrupt

Provides functions to enable/disable WWDT interrupts and get current enabled interrupts.

#### 30.2.4 Watch dog Refresh

The function [WWDT\\_Refresh\(\)](#) feeds the WWDT.

### 30.3 Typical use case

Refer to the driver examples codes located at <SDK\_ROOT>/boards/<BOARD>/driver\_examples/wwdt

#### Files

- file [fsl\\_wwdt.h](#)

#### Data Structures

- struct [wwdt\\_config\\_t](#)  
*Describes WWDT configuration structure. [More...](#)*

## Typical use case

## Enumerations

- enum `_wwdt_status_flags_t` {  
    `kWWDT_TimeoutFlag` = `WWDT_MOD_WDTOF_MASK`,  
    `kWWDT_WarningFlag` = `WWDT_MOD_WDINT_MASK` }  
    *WWDT status flags.*

## Driver version

- #define `FSL_WWDT_DRIVER_VERSION` (`MAKE_VERSION(2, 1, 3)`)  
    *Defines WWDT driver version 2.1.3.*

## Refresh sequence

- #define `WWDT_FIRST_WORD_OF_REFRESH` (`0xAAU`)  
    *First word of refresh sequence.*
- #define `WWDT_SECOND_WORD_OF_REFRESH` (`0x55U`)  
    *Second word of refresh sequence.*

## WWDT Initialization and De-initialization

- void `WWDT_GetDefaultConfig` (`wwdt_config_t *config`)  
    *Initializes WWDT configure structure.*
- void `WWDT_Init` (`WWDT_Type *base`, const `wwdt_config_t *config`)  
    *Initializes the WWDT.*
- void `WWDT_Deinit` (`WWDT_Type *base`)  
    *Shuts down the WWDT.*

## WWDT Functional Operation

- static void `WWDT_Enable` (`WWDT_Type *base`)  
    *Enables the WWDT module.*
- static void `WWDT_Disable` (`WWDT_Type *base`)  
    *Disables the WWDT module.*
- static uint32\_t `WWDT_GetStatusFlags` (`WWDT_Type *base`)  
    *Gets all WWDT status flags.*
- void `WWDT_ClearStatusFlags` (`WWDT_Type *base`, uint32\_t mask)  
    *Clear WWDT flag.*
- static void `WWDT_SetWarningValue` (`WWDT_Type *base`, uint32\_t warningValue)  
    *Set the WWDT warning value.*
- static void `WWDT_SetTimeoutValue` (`WWDT_Type *base`, uint32\_t timeoutCount)  
    *Set the WWDT timeout value.*
- static void `WWDT_SetWindowValue` (`WWDT_Type *base`, uint32\_t windowValue)  
    *Sets the WWDT window value.*
- void `WWDT_Refresh` (`WWDT_Type *base`)  
    *Refreshes the WWDT timer.*

## 30.4 Data Structure Documentation

### 30.4.1 struct wwdt\_config\_t

#### Data Fields

- bool [enableWwdt](#)  
*Enables or disables WWDT.*
- bool [enableWatchdogReset](#)  
*true: Watchdog timeout will cause a chip reset false: Watchdog timeout will not cause a chip reset*
- bool [enableWatchdogProtect](#)  
*true: Enable watchdog protect i.e timeout value can only be changed after counter is below warning & window values false: Disable watchdog protect; timeout value can be changed at any time*
- bool [enableLockOscillator](#)  
*true: Disabling or powering down the watchdog oscillator is prevented Once set, this bit can only be cleared by a reset false: Do not lock oscillator*
- uint32\_t [windowValue](#)  
*Window value, set this to 0xFFFFF if windowing is not in effect.*
- uint32\_t [timeoutValue](#)  
*Timeout value.*
- uint32\_t [warningValue](#)  
*Watchdog time counter value that will generate a warning interrupt.*
- uint32\_t [clockFreq\\_Hz](#)  
*Watchdog clock source frequency.*

#### 30.4.1.0.0.32 Field Documentation

##### 30.4.1.0.0.32.1 uint32\_t wwdt\_config\_t::warningValue

Set this to 0 for no warning

##### 30.4.1.0.0.32.2 uint32\_t wwdt\_config\_t::clockFreq\_Hz

## 30.5 Macro Definition Documentation

### 30.5.1 #define FSL\_WWDT\_DRIVER\_VERSION (MAKE\_VERSION(2, 1, 3))

## 30.6 Enumeration Type Documentation

### 30.6.1 enum \_wwdt\_status\_flags\_t

This structure contains the WWDT status flags for use in the WWDT functions.

Enumerator

***kWWDT\_TimeoutFlag*** Time-out flag, set when the timer times out.

***kWWDT\_WarningFlag*** Warning interrupt flag, set when timer is below the value WDWARNINT.

## Function Documentation

### 30.7 Function Documentation

#### 30.7.1 void WWDT\_GetDefaultConfig ( wwdt\_config\_t \* *config* )

This function initializes the WWDT configure structure to default value. The default value are:

```
* config->enableWwdt = true;
* config->enableWatchdogReset = false;
* config->enableWatchdogProtect = false;
* config->enableLockOscillator = false;
* config->windowValue = 0xFFFFFU;
* config->timeoutValue = 0xFFFFFU;
* config->warningValue = 0;
*
```

##### Parameters

|               |                                   |
|---------------|-----------------------------------|
| <i>config</i> | Pointer to WWDT config structure. |
|---------------|-----------------------------------|

See Also

[wwdt\\_config\\_t](#)

#### 30.7.2 void WWDT\_Init ( WWDT\_Type \* *base*, const wwdt\_config\_t \* *config* )

This function initializes the WWDT. When called, the WWDT runs according to the configuration.

Example:

```
* wwdt_config_t config;
* WWDT_GetDefaultConfig(&config);
* config.timeoutValue = 0x7ffU;
* WWDT_Init(wwdt_base, &config);
*
```

##### Parameters

|               |                              |
|---------------|------------------------------|
| <i>base</i>   | WWDT peripheral base address |
| <i>config</i> | The configuration of WWDT    |

#### 30.7.3 void WWDT\_Deinit ( WWDT\_Type \* *base* )

This function shuts down the WWDT.

## Parameters

|             |                              |
|-------------|------------------------------|
| <i>base</i> | WWDT peripheral base address |
|-------------|------------------------------|

**30.7.4 static void WWDT\_Enable ( WWDT\_Type \* *base* ) [inline], [static]**

This function write value into WWDT\_MOD register to enable the WWDT, it is a write-once bit; once this bit is set to one and a watchdog feed is performed, the watchdog timer will run permanently.

## Parameters

|             |                              |
|-------------|------------------------------|
| <i>base</i> | WWDT peripheral base address |
|-------------|------------------------------|

**30.7.5 static void WWDT\_Disable ( WWDT\_Type \* *base* ) [inline], [static]**

This function write value into WWDT\_MOD register to disable the WWDT.

## Parameters

|             |                              |
|-------------|------------------------------|
| <i>base</i> | WWDT peripheral base address |
|-------------|------------------------------|

**30.7.6 static uint32\_t WWDT\_GetStatusFlags ( WWDT\_Type \* *base* ) [inline], [static]**

This function gets all status flags.

Example for getting Timeout Flag:

```
* uint32_t status;
* status = WWDT_GetStatusFlags(wwdt_base) &
* kWWDTimeoutFlag;
*
```

## Parameters

|             |                              |
|-------------|------------------------------|
| <i>base</i> | WWDT peripheral base address |
|-------------|------------------------------|

## Returns

The status flags. This is the logical OR of members of the enumeration [\\_wwdt\\_status\\_flags\\_t](#)

## Function Documentation

### 30.7.7 void WWDT\_ClearStatusFlags ( WWDT\_Type \* *base*, uint32\_t *mask* )

This function clears WWDT status flag.

Example for clearing warning flag:

```
* WWDT_ClearStatusFlags(wwdt_base, kWWDT_WarningFlag);
*
```

Parameters

|             |                                                                                                                     |
|-------------|---------------------------------------------------------------------------------------------------------------------|
| <i>base</i> | WWDT peripheral base address                                                                                        |
| <i>mask</i> | The status flags to clear. This is a logical OR of members of the enumeration <a href="#">_wwdt-_status_flags_t</a> |

### 30.7.8 static void WWDT\_SetWarningValue ( WWDT\_Type \* *base*, uint32\_t *warningValue* ) [inline], [static]

The WDWARNINT register determines the watchdog timer counter value that will generate a watchdog interrupt. When the watchdog timer counter is no longer greater than the value defined by WARNINT, an interrupt will be generated after the subsequent WDCLK.

Parameters

|                     |                              |
|---------------------|------------------------------|
| <i>base</i>         | WWDT peripheral base address |
| <i>warningValue</i> | WWDT warning value.          |

### 30.7.9 static void WWDT\_SetTimeoutValue ( WWDT\_Type \* *base*, uint32\_t *timeoutCount* ) [inline], [static]

This function sets the timeout value. Every time a feed sequence occurs the value in the TC register is loaded into the Watchdog timer. Writing a value below 0xFF will cause 0xFF to be loaded into the TC register. Thus the minimum time-out interval is TWDCLK\*256\*4. If enableWatchdogProtect flag is true in [wwdt\\_config\\_t](#) config structure, any attempt to change the timeout value before the watchdog counter is below the warning and window values will cause a watchdog reset and set the WDTOF flag.

Parameters

|                     |                                               |
|---------------------|-----------------------------------------------|
| <i>base</i>         | WWDT peripheral base address                  |
| <i>timeoutCount</i> | WWDT timeout value, count of WWDT clock tick. |

### 30.7.10 static void WWDT\_SetWindowValue ( WWDT\_Type \* *base*, uint32\_t *windowValue* ) [inline], [static]

The WINDOW register determines the highest TV value allowed when a watchdog feed is performed. If a feed sequence occurs when timer value is greater than the value in WINDOW, a watchdog event will occur. To disable windowing, set *windowValue* to 0xFFFFFFFF (maximum possible timer value) so windowing is not in effect.

Parameters

|                    |                              |
|--------------------|------------------------------|
| <i>base</i>        | WWDT peripheral base address |
| <i>windowValue</i> | WWDT window value.           |

### 30.7.11 void WWDT\_Refresh ( WWDT\_Type \* *base* )

This function feeds the WWDT. This function should be called before WWDT timer is in timeout. Otherwise, a reset is asserted.

Parameters

|             |                              |
|-------------|------------------------------|
| <i>base</i> | WWDT peripheral base address |
|-------------|------------------------------|





## Chapter 31

### CMP: Comparator driver

#### 31.1 Overview

The MCUXpresso SDK provides a peripheral driver for the cmp driver module of MCUXpresso SDK devices.

#### Data Structures

- struct `cmp_config_t`  
*cmp configurataions [More...](#)*

#### Macros

- #define `CMP_INT_POL_SHIFT_VALUE` (1U)  
*cmp level shift value definition*

#### Enumerations

- enum `_cmp_status` {  
    `kCMP_In0BiggerThanIn1` = 1U,  
    `kCMP_In1BiggerThanIn0` = 0U }  
*cmp status*
- enum `cmp_interrupt_mask_t` {  
    `kCMP_EdgeRising` = 0U << `CMP_INT_POL_SHIFT_VALUE`,  
    `kCMP_EdgeFalling` = 1U << `CMP_INT_POL_SHIFT_VALUE`,  
    `kCMP_EdgeRisingFalling` = 3U << `CMP_INT_POL_SHIFT_VALUE`,  
    `kCMP_LevelLow` = (0U << `CMP_INT_POL_SHIFT_VALUE`) | 1U,  
    `kCMP_LevelHigh` = (2U << `CMP_INT_POL_SHIFT_VALUE`) | 1U }  
*cmp interrupt*
- enum `cmp_mode_t` {  
    `kCMP_FastMode` = 0U,  
    `kCMP_LowpowerMode` = 1U }  
*cmp work mode*
- enum `cmp_input_t` {  
    `kCMP_InputAllExternal` = 0U,  
    `kCMP_InputOneExternalOneInternal` }  
*cmp input source*

#### Driver version

- #define `FSL_CMP_DRIVER_VERSION` (`MAKE_VERSION`(2U, 0U, 1U))  
*Driver version 2.0.1.*

### Cmp Initialization and deinitialization

- void [CMP\\_Init](#) ([cmp\\_config\\_t](#) \*config)  
*CMP initialization.*
- void [CMP\\_Deinit](#) (void)  
*CMP deinitialization.*

### cmp functionality

- static void [CMP\\_SwapExtInput](#) (void)  
*Swap the external input channel.*
- static void [CMP\\_EnableLowPowerMode](#) (bool enable)  
*switch cmp work mode.*
- static void [CMP\\_EnableInnerInput](#) (bool enable)  
*switch input source.*
- static void [CMP\\_EnableLowHysteresis](#) (bool enable)  
*cmp enable low hysteresis.*
- static uint32\_t [CMP\\_GetOutput](#) (void)  
*cmp output status.*

### cmp interrupt

- void [CMP\\_SetInterruptConfig](#) ([cmp\\_interrupt\\_mask\\_t](#) mask)  
*cmp set interrupt configurations.*
- static void [CMP\\_EnableInterrupt](#) (void)  
*cmp enable interrupt.*
- static void [CMP\\_DisableInterrupt](#) (void)  
*cmp disable interrupt.*
- static bool [CMP\\_GetStatus](#) (void)  
*cmp get status.*
- static void [CMP\\_ClearStatus](#) (void)  
*cmp clear interrupt status.*
- static bool [CMP\\_GetInterruptStatus](#) (void)  
*cmp get interrupt status.*

## 31.2 Data Structure Documentation

### 31.2.1 struct [cmp\\_config\\_t](#)

#### Data Fields

- bool [enLowHysteris](#)  
*low hysteresis*
- [cmp\\_input\\_t](#) [src](#)  
*input source select*
- [cmp\\_mode\\_t](#) [mode](#)  
*cmp work mode*

### 31.3 Macro Definition Documentation

#### 31.3.1 #define FSL\_CMP\_DRIVER\_VERSION (MAKE\_VERSION(2U, 0U, 1U))

### 31.4 Enumeration Type Documentation

#### 31.4.1 enum \_cmp\_status

Enumerator

*kCMP\_In0BiggerThanIn1* comparator input 0 is bigger than input 1

*kCMP\_In1BiggerThanIn0* comparator input 1 is bigger than input 0

#### 31.4.2 enum cmp\_interrupt\_mask\_t

Enumerator

*kCMP\_EdgeRising* Edge sensitive, falling edge.

*kCMP\_EdgeFalling* Edge sensitive, rising edge.

*kCMP\_EdgeRisingFalling* Edge sensitive, rising and falling edge.

*kCMP\_LevelLow* Level sensitive, low level.

*kCMP\_LevelHigh* Level sensitive, high level.

#### 31.4.3 enum cmp\_mode\_t

Enumerator

*kCMP\_FastMode* Used in an active or deep sleep mode, this mode requires PMU bias enabled.

*kCMP\_LowpowerMode* Used for all power mode, doesn't require PMU bias enabled.

#### 31.4.4 enum cmp\_input\_t

Enumerator

*kCMP\_InputAllExternal* Cmp input from two external source.

*kCMP\_InputOneExternalOneInternal* Cmp input from one external input and one internal voltage reference 0.8V.

### 31.5 Function Documentation

#### 31.5.1 void CMP\_Init ( cmp\_config\_t \* config )

Note: The cmp initial function not responsible for cmp power, application shall handle it.

## Function Documentation

### Parameters

|               |                      |
|---------------|----------------------|
| <i>config</i> | init configurations. |
|---------------|----------------------|

### 31.5.2 void CMP\_Deinit ( void )

Note: The cmp deinit function not responsible for cmp power, application shall handle it.

### 31.5.3 static void CMP\_SwapExtInput ( void ) [inline], [static]

### Parameters

|             |                   |
|-------------|-------------------|
| <i>base</i> | CMP base address. |
|-------------|-------------------|

### 31.5.4 static void CMP\_EnableLowePowerMode ( bool *enable* ) [inline], [static]

### Parameters

|               |                                                        |
|---------------|--------------------------------------------------------|
| <i>enable</i> | true is enter low power mode, false is enter fast mode |
|---------------|--------------------------------------------------------|

### 31.5.5 static void CMP\_EnableInnerInput ( bool *enable* ) [inline], [static]

### Parameters

|               |                                                               |
|---------------|---------------------------------------------------------------|
| <i>enable</i> | true is one external and one internal, false is all external. |
|---------------|---------------------------------------------------------------|

### 31.5.6 static uint32\_t CMP\_GetOutput ( void ) [inline], [static]

### Returns

0 is kCMP\_In1BiggerThanIn0, 1 is kCMP\_In0BiggerThanIn1.

### 31.5.7 void CMP\_SetInterruptConfig ( cmp\_interrupt\_mask\_t *mask* )

## Parameters

|             |                 |
|-------------|-----------------|
| <i>mask</i> | interrupt mask. |
|-------------|-----------------|

**31.5.8 static bool CMP\_GetStatus ( void ) [inline], [static]**

## Returns

true is interrupt pending, false is no interrupt pending.

**31.5.9 static void CMP\_ClearStatus ( void ) [inline], [static]**

## Returns

true is interrupt pending, false is no interrupt pending.

**31.5.10 static bool CMP\_GetInterruptStatus ( void ) [inline], [static]**

## Returns

true is interrupt pending, false is no interrupt pending.



## Chapter 32

# FLASHIAP: Flash In Application Programming Driver

### 32.1 Overview

The MCUXpresso SDK provides a driver for the Flash In Application Programming (FLASHIAP).

It provides a set of functions to call the on-chip in application flash programming interface. User code executing from on-chip flash or RAM can call these function to erase and write the flash memory.

### 32.2 GFlash In Application Programming operation

[FLASHIAP\\_PrepareSectorForWrite\(\)](#) prepares a sector for write or erase operation.

[FLASHIAP\\_CopyRamToFlash\(\)](#) function programs the flash memory.

[FLASHIAP\\_EraseSector\(\)](#) function erase a flash sector. A sector must be erased before write operation.

### 32.3 Typical use case

Refer to the driver examples codes located at <SDK\_ROOT>/boards/<BOARD>/driver\_examples/flashiap

#### Files

- file [fsl\\_flashiap.h](#)

#### Typedefs

- typedef void(\* [IAP\\_ENTRY\\_T](#) )(uint32\_t cmd[5], uint32\_t stat[4])  
*IAP\_ENTRY API function type.*

## Enumerations

- enum `_flashiap_status` {  
    `kStatus_FLASHIAP_Success` = `kStatus_Success`,  
    `kStatus_FLASHIAP_InvalidCommand` = `MAKE_STATUS(kStatusGroup_FLASHIAP, 1U)`,  
    `kStatus_FLASHIAP_SrcAddrError`,  
    `kStatus_FLASHIAP_DstAddrError`,  
    `kStatus_FLASHIAP_SrcAddrNotMapped`,  
    `kStatus_FLASHIAP_DstAddrNotMapped`,  
    `kStatus_FLASHIAP_CountError`,  
    `kStatus_FLASHIAP_InvalidSector`,  
    `kStatus_FLASHIAP_SectorNotblank` = `MAKE_STATUS(kStatusGroup_FLASHIAP, 8U)`,  
    `kStatus_FLASHIAP_NotPrepared`,  
    `kStatus_FLASHIAP_CompareError`,  
    `kStatus_FLASHIAP_Busy`,  
    `kStatus_FLASHIAP_ParamError`,  
    `kStatus_FLASHIAP_AddrError` = `MAKE_STATUS(kStatusGroup_FLASHIAP, 13U)`,  
    `kStatus_FLASHIAP_AddrNotMapped`,  
    `kStatus_FLASHIAP_NoPower` = `MAKE_STATUS(kStatusGroup_FLASHIAP, 24U)`,  
    `kStatus_FLASHIAP_NoClock` }

*Flashiap status codes.*

- enum `_flashiap_commands` {  
    `kIapCmd_FLASHIAP_PrepareSectorforWrite` = 50U,  
    `kIapCmd_FLASHIAP_CopyRamToFlash` = 51U,  
    `kIapCmd_FLASHIAP_EraseSector` = 52U,  
    `kIapCmd_FLASHIAP_BlankCheckSector` = 53U,  
    `kIapCmd_FLASHIAP_ReadPartId` = 54U,  
    `kIapCmd_FLASHIAP_Read_BootromVersion` = 55U,  
    `kIapCmd_FLASHIAP_Compare` = 56U,  
    `kIapCmd_FLASHIAP_ReinvokeISP` = 57U,  
    `kIapCmd_FLASHIAP_ReadUid` = 58U,  
    `kIapCmd_FLASHIAP_ErasePage` = 59U,  
    `kIapCmd_FLASHIAP_ReadMisr` = 70U,  
    `kIapCmd_FLASHIAP_ReinvokeI2cSpiISP` = 71U }

*Flashiap command codes.*

## Functions

- static void `iap_entry` (`uint32_t` \*cmd\_param, `uint32_t` \*status\_result)  
*IAP\_ENTRY API function type.*
- `status_t FLASHIAP_PrepareSectorForWrite` (`uint32_t` startSector, `uint32_t` endSector)  
*Prepare sector for write operation.*
- `status_t FLASHIAP_CopyRamToFlash` (`uint32_t` dstAddr, `uint32_t` \*srcAddr, `uint32_t` numOfBytes, `uint32_t` systemCoreClock)  
*Copy RAM to flash.*
- `status_t FLASHIAP_EraseSector` (`uint32_t` startSector, `uint32_t` endSector, `uint32_t` systemCoreClock)



- *Erase sector.*  
**status\_t FLASHIAP\_ErasePage** (uint32\_t startPage, uint32\_t endPage, uint32\_t systemCoreClock)  
*This function erases page(s).*
- **status\_t FLASHIAP\_BlankCheckSector** (uint32\_t startSector, uint32\_t endSector)  
*Blank check sector(s)*
- **status\_t FLASHIAP\_Compare** (uint32\_t dstAddr, uint32\_t \*srcAddr, uint32\_t numOfBytes)  
*Compare memory contents of flash with ram.*

## Driver version

- #define **FSL\_FLASHIAP\_DRIVER\_VERSION** (MAKE\_VERSION(2, 0, 3))  
*Version 2.0.3.*

## 32.4 Macro Definition Documentation

### 32.4.1 #define FSL\_FLASHIAP\_DRIVER\_VERSION (MAKE\_VERSION(2, 0, 3))

## 32.5 Enumeration Type Documentation

### 32.5.1 enum \_flashiap\_status

Enumerator

- kStatus\_FLASHIAP\_Success** Api is executed successfully.
- kStatus\_FLASHIAP\_InvalidCommand** Invalid command.
- kStatus\_FLASHIAP\_SrcAddrError** Source address is not on word boundary.
- kStatus\_FLASHIAP\_DstAddrError** Destination address is not on a correct boundary.
- kStatus\_FLASHIAP\_SrcAddrNotMapped** Source address is not mapped in the memory map.
- kStatus\_FLASHIAP\_DstAddrNotMapped** Destination address is not mapped in the memory map.
- kStatus\_FLASHIAP\_CountError** Byte count is not multiple of 4 or is not a permitted value.
- kStatus\_FLASHIAP\_InvalidSector** Sector number is invalid or end sector number is greater than start sector number.
- kStatus\_FLASHIAP\_SectorNotblank** One or more sectors are not blank.
- kStatus\_FLASHIAP\_NotPrepared** Command to prepare sector for write operation was not executed.
- kStatus\_FLASHIAP\_CompareError** Destination and source memory contents do not match.
- kStatus\_FLASHIAP\_Busy** Flash programming hardware interface is busy.
- kStatus\_FLASHIAP\_ParamError** Insufficient number of parameters or invalid parameter.
- kStatus\_FLASHIAP\_AddrError** Address is not on word boundary.
- kStatus\_FLASHIAP\_AddrNotMapped** Address is not mapped in the memory map.
- kStatus\_FLASHIAP\_NoPower** Flash memory block is powered down.
- kStatus\_FLASHIAP\_NoClock** Flash memory block or controller is not clocked.

## Function Documentation

### 32.5.2 enum \_flashiap\_commands

Enumerator

*kIapCmd\_FLASHIAP\_PrepareSectorforWrite* Prepare Sector for write.  
*kIapCmd\_FLASHIAP\_CopyRamToFlash* Copy RAM to flash.  
*kIapCmd\_FLASHIAP\_EraseSector* Erase Sector.  
*kIapCmd\_FLASHIAP\_BlankCheckSector* Blank check sector.  
*kIapCmd\_FLASHIAP\_ReadPartId* Read part id.  
*kIapCmd\_FLASHIAP\_Read\_BootromVersion* Read bootrom version.  
*kIapCmd\_FLASHIAP\_Compare* Compare.  
*kIapCmd\_FLASHIAP\_ReinvokeISP* Reinvoke ISP.  
*kIapCmd\_FLASHIAP\_ReadUid* Read Uid isp.  
*kIapCmd\_FLASHIAP\_ErasePage* Erase Page.  
*kIapCmd\_FLASHIAP\_ReadMisr* Read Misr.  
*kIapCmd\_FLASHIAP\_ReinvokeI2cSpiISP* Reinvoke I2C/SPI isp.

## 32.6 Function Documentation

### 32.6.1 static void iap\_entry ( uint32\_t \* cmd\_param, uint32\_t \* status\_result ) [inline], [static]

Wrapper for rom iap call

Parameters

|                      |                                           |
|----------------------|-------------------------------------------|
| <i>cmd_param</i>     | IAP command and relevant parameter array. |
| <i>status_result</i> | IAP status result array.                  |

Return values

|              |                                                    |
|--------------|----------------------------------------------------|
| <i>None.</i> | Status/Result is returned via status_result array. |
|--------------|----------------------------------------------------|

### 32.6.2 status\_t FLASHIAP\_PrepareSectorForWrite ( uint32\_t startSector, uint32\_t endSector )

This function prepares sector(s) for write/erase operation. This function must be called before calling the [FLASHIAP\\_CopyRamToFlash\(\)](#) or [FLASHIAP\\_EraseSector\(\)](#) or [FLASHIAP\\_ErasePage\(\)](#) function. The end sector must be greater than or equal to start sector number.

## Parameters

|                    |                      |
|--------------------|----------------------|
| <i>startSector</i> | Start sector number. |
| <i>endSector</i>   | End sector number.   |

## Return values

|                                       |                                                                                    |
|---------------------------------------|------------------------------------------------------------------------------------|
| <i>kStatus_FLASHIAP_Success</i>       | Api was executed successfully.                                                     |
| <i>kStatus_FLASHIAP_NoPower</i>       | Flash memory block is powered down.                                                |
| <i>kStatus_FLASHIAP_NoClock</i>       | Flash memory block or controller is not clocked.                                   |
| <i>kStatus_FLASHIAP_InvalidSector</i> | Sector number is invalid or end sector number is greater than start sector number. |
| <i>kStatus_FLASHIAP_Busy</i>          | Flash programming hardware interface is busy.                                      |

### 32.6.3 status\_t FLASHIAP\_CopyRamToFlash ( uint32\_t dstAddr, uint32\_t \* srcAddr, uint32\_t numOfBytes, uint32\_t systemCoreClock )

This function programs the flash memory. Corresponding sectors must be prepared via FLASHIAP\_PrepSectorForWrite before calling this function. The addresses should be a 256 byte boundary and the number of bytes should be 256 | 512 | 1024 | 4096.

## Parameters

|                        |                                                                                    |
|------------------------|------------------------------------------------------------------------------------|
| <i>dstAddr</i>         | Destination flash address where data bytes are to be written.                      |
| <i>srcAddr</i>         | Source ram address from where data bytes are to be read.                           |
| <i>numOfBytes</i>      | Number of bytes to be written.                                                     |
| <i>systemCoreClock</i> | SystemCoreClock in Hz. It is converted to KHz before calling the rom IAP function. |

## Return values

|                                 |                                |
|---------------------------------|--------------------------------|
| <i>kStatus_FLASHIAP_Success</i> | Api was executed successfully. |
|---------------------------------|--------------------------------|

## Function Documentation

|                                          |                                                                 |
|------------------------------------------|-----------------------------------------------------------------|
| <i>kStatus_FLASHIAP_NoPower</i>          | Flash memory block is powered down.                             |
| <i>kStatus_FLASHIAP_NoClock</i>          | Flash memory block or controller is not clocked.                |
| <i>kStatus_FLASHIAP_SrcAddrError</i>     | Source address is not on word boundary.                         |
| <i>kStatus_FLASHIAP_DstAddrError</i>     | Destination address is not on a correct boundary.               |
| <i>kStatus_FLASHIAP_SrcAddrNotMapped</i> | Source address is not mapped in the memory map.                 |
| <i>kStatus_FLASHIAP_DstAddrNotMapped</i> | Destination address is not mapped in the memory map.            |
| <i>kStatus_FLASHIAP_CountError</i>       | Byte count is not multiple of 4 or is not a permitted value.    |
| <i>kStatus_FLASHIAP_NotPrepared</i>      | Command to prepare sector for write operation was not executed. |
| <i>kStatus_FLASHIAP_Busy</i>             | Flash programming hardware interface is busy.                   |

### 32.6.4 **status\_t FLASHIAP\_EraseSector ( uint32\_t startSector, uint32\_t endSector, uint32\_t systemCoreClock )**

This function erases sector(s). The end sector must be greater than or equal to start sector number. FLASHIAP\_PrepareSectorForWrite must be called before calling this function.

#### Parameters

|                        |                                                                                    |
|------------------------|------------------------------------------------------------------------------------|
| <i>startSector</i>     | Start sector number.                                                               |
| <i>endSector</i>       | End sector number.                                                                 |
| <i>systemCoreClock</i> | SystemCoreClock in Hz. It is converted to KHz before calling the rom IAP function. |

#### Return values

|                                 |                                |
|---------------------------------|--------------------------------|
| <i>kStatus_FLASHIAP_Success</i> | Api was executed successfully. |
|---------------------------------|--------------------------------|

|                                       |                                                                                    |
|---------------------------------------|------------------------------------------------------------------------------------|
| <i>kStatus_FLASHIAP_NoPower</i>       | Flash memory block is powered down.                                                |
| <i>kStatus_FLASHIAP_NoClock</i>       | Flash memory block or controller is not clocked.                                   |
| <i>kStatus_FLASHIAP_InvalidSector</i> | Sector number is invalid or end sector number is greater than start sector number. |
| <i>kStatus_FLASHIAP_NotPrepared</i>   | Command to prepare sector for write operation was not executed.                    |
| <i>kStatus_FLASHIAP_Busy</i>          | Flash programming hardware interface is busy.                                      |

### 32.6.5 status\_t FLASHIAP\_ErasePage ( uint32\_t startPage, uint32\_t endPage, uint32\_t systemCoreClock )

The end page must be greater than or equal to start page number. Corresponding sectors must be prepared via FLASHIAP\_PrepareSectorForWrite before calling this function.

Parameters

|                        |                                                                                    |
|------------------------|------------------------------------------------------------------------------------|
| <i>startPage</i>       | Start page number                                                                  |
| <i>endPage</i>         | End page number                                                                    |
| <i>systemCoreClock</i> | SystemCoreClock in Hz. It is converted to KHz before calling the rom IAP function. |

Return values

|                                       |                                                                             |
|---------------------------------------|-----------------------------------------------------------------------------|
| <i>kStatus_FLASHIAP_Success</i>       | Api was executed successfully.                                              |
| <i>kStatus_FLASHIAP_NoPower</i>       | Flash memory block is powered down.                                         |
| <i>kStatus_FLASHIAP_NoClock</i>       | Flash memory block or controller is not clocked.                            |
| <i>kStatus_FLASHIAP_InvalidSector</i> | Page number is invalid or end page number is greater than start page number |

## Function Documentation

|                                      |                                                                 |
|--------------------------------------|-----------------------------------------------------------------|
| <i>kStatus_FLASHIAP_Not-Prepared</i> | Command to prepare sector for write operation was not executed. |
| <i>kStatus_FLASHIAP_Busy</i>         | Flash programming hardware interface is busy.                   |

### 32.6.6 **status\_t FLASHIAP\_BlankCheckSector ( uint32\_t startSector, uint32\_t endSector )**

Blank check single or multiples sectors of flash memory. The end sector must be greater than or equal to start sector number. It can be used to verify the sector eraseure after FLASHIAP\_EraseSector call.

Parameters

|                    |                                                                             |
|--------------------|-----------------------------------------------------------------------------|
| <i>startSector</i> | : Start sector number. Must be greater than or equal to start sector number |
| <i>endSector</i>   | : End sector number                                                         |

Return values

|                                        |                                                  |
|----------------------------------------|--------------------------------------------------|
| <i>kStatus_FLASHIAP_Success</i>        | One or more sectors are in erased state.         |
| <i>kStatus_FLASHIAP_No-Power</i>       | Flash memory block is powered down.              |
| <i>kStatus_FLASHIAP_No-Clock</i>       | Flash memory block or controller is not clocked. |
| <i>kStatus_FLASHIAP_SectorNotblank</i> | One or more sectors are not blank.               |

### 32.6.7 **status\_t FLASHIAP\_Compare ( uint32\_t dstAddr, uint32\_t \* srcAddr, uint32\_t numBytes )**

This function compares the contents of flash and ram. It can be used to verify the flash memory contents after FLASHIAP\_CopyRamToFlash call.

Parameters

|                |                            |
|----------------|----------------------------|
| <i>dstAddr</i> | Destination flash address. |
|----------------|----------------------------|

|                   |                                 |
|-------------------|---------------------------------|
| <i>srcAddr</i>    | Source ram address.             |
| <i>numOfBytes</i> | Number of bytes to be compared. |

## Return values

|                                       |                                                              |
|---------------------------------------|--------------------------------------------------------------|
| <i>kStatus_FLASHIAP_Success</i>       | Contents of flash and ram match.                             |
| <i>kStatus_FLASHIAP_NoPower</i>       | Flash memory block is powered down.                          |
| <i>kStatus_FLASHIAP_NoClock</i>       | Flash memory block or controller is not clocked.             |
| <i>kStatus_FLASHIAP_AddrError</i>     | Address is not on word boundary.                             |
| <i>kStatus_FLASHIAP_AddrNotMapped</i> | Address is not mapped in the memory map.                     |
| <i>kStatus_FLASHIAP_CountError</i>    | Byte count is not multiple of 4 or is not a permitted value. |
| <i>kStatus_FLASHIAP_CompareError</i>  | Destination and source memory contents do not match.         |





## Chapter 33

# SHA: SHA encryption decryption driver

### 33.1 Overview

The MCUXpresso SDK provides a peripheral driver for the SHA module in MCUXpresso SDK devices.

The driver provides blocking synchronous APIs. The SHA operations are complete (and results are made available for further usage) when a function returns. When called, these functions do not return until an S-HA operation is complete. These functions use main CPU for simple polling loops to determine operation complete or error status and data movements. The driver functions are not re-entrant. These functions provide typical interface to upper layer or application software.

### 33.2 SHA Driver Initialization and Configuration

Clock to the SHA module has to be enabled before using the driver API.

### 33.3 Comments about API usage in RTOS

SHA operations provided by this driver are not re-entrant. Therefore, the application software should ensure the SHA module operation is not requested from different tasks or interrupt service routines while an operation is in progress.

### 33.4 SHA Driver Example

Typical use case Refer to the driver examples codes located at `<SDK_ROOT>/boards/<BOARD>/driver_examples/sha`

#### Modules

- [Sha\\_algorithm\\_level\\_api](#)

#### Files

- file [fsl\\_sha.h](#)

#### Data Structures

- struct [sha\\_ctx\\_t](#)  
*Storage type used to save hash context. [More...](#)*

#### Macros

- #define [SHA\\_CTX\\_SIZE](#) 20  
*SHA Context size.*

## Enumeration Type Documentation

### Enumerations

- enum `sha_algo_t` {  
    `kSHA_Sha1`,  
    `kSHA_Sha256` }

*Supported cryptographic block cipher functions for HASH creation.*

### Driver version

- #define `FSL_SHA_DRIVER_VERSION` (`MAKE_VERSION`(2, 1, 0))  
*Defines LPC SHA driver version 2.1.0.*

## 33.5 Data Structure Documentation

### 33.5.1 struct `sha_ctx_t`

## 33.6 Macro Definition Documentation

### 33.6.1 #define `FSL_SHA_DRIVER_VERSION` (`MAKE_VERSION`(2, 1, 0))

### 33.6.2 #define `SHA_CTX_SIZE` 20

## 33.7 Enumeration Type Documentation

### 33.7.1 enum `sha_algo_t`

Enumerator

*kSHA\_Sha1*   SHA\_1.  
*kSHA\_Sha256*   SHA\_256.

## Chapter 34 Serial Manager

### 34.1 Overview

This chapter describes the programming interface of the serial manager component.

The serial manager component provides a series of APIs to operate different serial port types. The port types it supports are UART, USB CDC and SWO.

### Modules

- [Serial Port SWO](#)
- [Serial Port USB](#)
- [Serial Port Uart](#)
- [Serial Port Virtual USB](#)

### Data Structures

- struct [serial\\_manager\\_config\\_t](#)  
*serial manager config structure [More...](#)*
- struct [serial\\_manager\\_callback\\_message\\_t](#)  
*Callback message structure. [More...](#)*

### Macros

- #define [SERIAL\\_PORT\\_TYPE\\_UART](#) (1U)  
*Enable or disable uart port (1 - enable, 0 - disable)*
- #define [SERIAL\\_PORT\\_TYPE\\_USBCDC](#) (0U)  
*Enable or disable USB CDC port (1 - enable, 0 - disable)*
- #define [SERIAL\\_PORT\\_TYPE\\_SWO](#) (0U)  
*Enable or disable SWO port (1 - enable, 0 - disable)*
- #define [SERIAL\\_PORT\\_TYPE\\_USBCDC\\_VIRTUAL](#) (0U)  
*Enable or disable USB CDC virtual port (1 - enable, 0 - disable)*
- #define [SERIAL\\_MANAGER\\_WRITE\\_HANDLE\\_SIZE](#) (4U)  
*Set serial manager write handle size.*
- #define [SERIAL\\_MANAGER\\_HANDLE\\_SIZE](#) (SERIAL\_MANAGER\_HANDLE\_SIZE\_TEMP + 12U)  
*SERIAL\_PORT\_UART\_HANDLE\_SIZE/SERIAL\_PORT\_USB\_CDC\_HANDLE\_SIZE + serial manager dedicated size.*

### Typedefs

- typedef void(\* [serial\\_manager\\_callback\\_t](#) )(void \*callbackParam, [serial\\_manager\\_callback\\_message\\_t](#) \*message, [serial\\_manager\\_status\\_t](#) status)  
*callback function*

## Enumerations

- enum `serial_port_type_t` {  
    `kSerialPort_None` = 0U,  
    `kSerialPort_Uart` = 1U,  
    `kSerialPort_Uart` = 1U,  
    `kSerialPort_UsbCdc`,  
    `kSerialPort_Swo`,  
    `kSerialPort_UsbCdcVirtual` }  
    *serial port type*
- enum `serial_manager_status_t` {  
    `kStatus_SerialManager_Success` = `kStatus_Success`,  
    `kStatus_SerialManager_Error` = `MAKE_STATUS(kStatusGroup_SERIALMANAGER, 1)`,  
    `kStatus_SerialManager_Busy` = `MAKE_STATUS(kStatusGroup_SERIALMANAGER, 2)`,  
    `kStatus_SerialManager_Notify` = `MAKE_STATUS(kStatusGroup_SERIALMANAGER, 3)`,  
    `kStatus_SerialManager_Canceled`,  
    `kStatus_SerialManager_HandleConflict` = `MAKE_STATUS(kStatusGroup_SERIALMANAGER, 5)`,  
    `kStatus_SerialManager_RingBufferOverflow` }  
    *serial manager error code*

## Functions

- `serial_manager_status_t SerialManager_Init` (`serial_handle_t serialHandle`, `serial_manager_config_t *config`)  
    *Initializes a serial manager module with the serial manager handle and the user configuration structure.*
- `serial_manager_status_t SerialManager_Deinit` (`serial_handle_t serialHandle`)  
    *De-initializes the serial manager module instance.*
- `serial_manager_status_t SerialManager_OpenWriteHandle` (`serial_handle_t serialHandle`, `serial_write_handle_t writeHandle`)  
    *Opens a writing handle for the serial manager module.*
- `serial_manager_status_t SerialManager_CloseWriteHandle` (`serial_write_handle_t writeHandle`)  
    *Closes a writing handle for the serial manager module.*
- `serial_manager_status_t SerialManager_OpenReadHandle` (`serial_handle_t serialHandle`, `serial_read_handle_t readHandle`)  
    *Opens a reading handle for the serial manager module.*
- `serial_manager_status_t SerialManager_CloseReadHandle` (`serial_read_handle_t readHandle`)  
    *Closes a reading for the serial manager module.*
- `serial_manager_status_t SerialManager_WriteBlocking` (`serial_write_handle_t writeHandle`, `uint8_t *buffer`, `uint32_t length`)  
    *Transmits data with the blocking mode.*
- `serial_manager_status_t SerialManager_ReadBlocking` (`serial_read_handle_t readHandle`, `uint8_t *buffer`, `uint32_t length`)  
    *Reads data with the blocking mode.*
- `serial_manager_status_t SerialManager_EnterLowpower` (`serial_handle_t serialHandle`)  
    *Prepares to enter low power consumption.*
- `serial_manager_status_t SerialManager_ExitLowpower` (`serial_handle_t serialHandle`)  
    *Restores from low power consumption.*

## 34.2 Data Structure Documentation

### 34.2.1 struct serial\_manager\_config\_t

#### Data Fields

- uint8\_t \* [ringBuffer](#)  
*Ring buffer address, it is used to buffer data received by the hardware.*
- uint32\_t [ringBufferSize](#)  
*The size of the ring buffer.*
- [serial\\_port\\_type\\_t](#) type  
*Serial port type.*
- void \* [portConfig](#)  
*Serial port configuration.*

#### 34.2.1.0.0.33 Field Documentation

##### 34.2.1.0.0.33.1 uint8\_t\* serial\_manager\_config\_t::ringBuffer

Besides, the memory space cannot be free during the lifetime of the serial manager module.

### 34.2.2 struct serial\_manager\_callback\_message\_t

#### Data Fields

- uint8\_t \* [buffer](#)  
*Transferred buffer.*
- uint32\_t [length](#)  
*Transferred data length.*

## 34.3 Enumeration Type Documentation

### 34.3.1 enum serial\_port\_type\_t

Enumerator

***kSerialPort\_None*** Serial port is none.  
***kSerialPort\_Uart*** Serial port UART.  
***kSerialPort\_Uart*** Serial port UART.  
***kSerialPort\_UsbCdc*** Serial port USB CDC.  
***kSerialPort\_Swo*** Serial port SWO.  
***kSerialPort\_UsbCdcVirtual*** Serial port USB CDC Virtual.

## Function Documentation

### 34.3.2 enum serial\_manager\_status\_t

Enumerator

*kStatus\_SerialManager\_Success* Success.  
*kStatus\_SerialManager\_Error* Failed.  
*kStatus\_SerialManager\_Busy* Busy.  
*kStatus\_SerialManager\_Notify* Ring buffer is not empty.  
*kStatus\_SerialManager\_Canceled* the non-blocking request is canceled  
*kStatus\_SerialManager\_HandleConflict* The handle is opened.  
*kStatus\_SerialManager\_RingBufferOverflow* The ring buffer is overflowed.

## 34.4 Function Documentation

### 34.4.1 serial\_manager\_status\_t SerialManager\_Init ( serial\_handle\_t serialHandle, serial\_manager\_config\_t \* config )

This function configures the Serial Manager module with user-defined settings. The user can configure the configuration structure. The parameter serialHandle is a pointer to point to a memory space of size [SERIAL\\_MANAGER\\_HANDLE\\_SIZE](#) allocated by the caller. The Serial Manager module supports two types of serial port, UART (includes UART, USART, LPSCI, LPUART, etc) and USB CDC. Please refer to [serial\\_port\\_type\\_t](#) for serial port setting. These two types can be set by using [serial\\_manager\\_config\\_t](#).

Example below shows how to use this API to configure the Serial Manager. For UART,

```
* #define SERIAL_MANAGER_RING_BUFFER_SIZE (256U)
* static uint8_t s_serialHandleBuffer[SERIAL_MANAGER_HANDLE_SIZE];
* static serial_handle_t s_serialHandle = &s_serialHandleBuffer[0];
* static uint8_t s_ringBuffer[SERIAL_MANAGER_RING_BUFFER_SIZE];
*
* serial_manager_config_t config;
* serial_port_uart_config_t uartConfig;
* config.type = kSerialPort_Uart;
* config.ringBuffer = &s_ringBuffer[0];
* config.ringBufferSize = SERIAL_MANAGER_RING_BUFFER_SIZE;
* uartConfig.instance = 0;
* uartConfig.clockRate = 24000000;
* uartConfig.baudRate = 115200;
* uartConfig.parityMode = kSerialManager_UartParityDisabled;
* uartConfig.stopBitCount = kSerialManager_UartOneStopBit;
* uartConfig.enableRx = 1;
* uartConfig.enableTx = 1;
* config.portConfig = &uartConfig;
* SerialManager_Init(s_serialHandle, &config);
*
```

For USB CDC,

```
* #define SERIAL_MANAGER_RING_BUFFER_SIZE (256U)
* static uint8_t s_serialHandleBuffer[SERIAL_MANAGER_HANDLE_SIZE];
* static serial_handle_t s_serialHandle = &s_serialHandleBuffer[0];
* static uint8_t s_ringBuffer[SERIAL_MANAGER_RING_BUFFER_SIZE];
*
* serial_manager_config_t config;
```

```

* serial_port_usb_cdc_config_t usbCdcConfig;
* config.type = kSerialPort_UsbCdc;
* config.ringBuffer = &s_ringBuffer[0];
* config.ringBufferSize = SERIAL_MANAGER_RING_BUFFER_SIZE;
* usbCdcConfig.controllerIndex =
* kSerialManager_UsbControllerKhci0;
* config.portConfig = &usbCdcConfig;
* SerialManager_Init(s_serialHandle, &config);
*

```

## Parameters

|                     |                                                                                                                |
|---------------------|----------------------------------------------------------------------------------------------------------------|
| <i>serialHandle</i> | Pointer to point to a memory space of size <a href="#">SERIAL_MANAGER_HANDLE_SIZE</a> allocated by the caller. |
| <i>config</i>       | Pointer to user-defined configuration structure.                                                               |

## Return values

|                                      |                                                   |
|--------------------------------------|---------------------------------------------------|
| <i>kStatus_SerialManager_Error</i>   | An error occurred.                                |
| <i>kStatus_SerialManager_Success</i> | The Serial Manager module initialization succeed. |

### 34.4.2 serial\_manager\_status\_t SerialManager\_Deinit ( serial\_handle\_t serialHandle )

This function de-initializes the serial manager module instance. If the opened writing or reading handle is not closed, the function will return `kStatus_SerialManager_Busy`.

## Parameters

|                     |                                           |
|---------------------|-------------------------------------------|
| <i>serialHandle</i> | The serial manager module handle pointer. |
|---------------------|-------------------------------------------|

## Return values

|                                      |                                                 |
|--------------------------------------|-------------------------------------------------|
| <i>kStatus_SerialManager_Success</i> | The serial manager de-initialization succeed.   |
| <i>kStatus_SerialManager_Busy</i>    | Opened reading or writing handle is not closed. |

## Function Documentation

### 34.4.3 serial\_manager\_status\_t SerialManager\_OpenWriteHandle ( serial\_handle\_t serialHandle, serial\_write\_handle\_t writeHandle )

This function Opens a writing handle for the serial manager module. If the serial manager needs to be used in different tasks, the task should open a dedicated write handle for itself by calling [SerialManager\\_OpenWriteHandle](#). Since there can only one buffer for transmission for the writing handle at the same time, multiple writing handles need to be opened when the multiple transmission is needed for a task.

Parameters

|                     |                                                   |
|---------------------|---------------------------------------------------|
| <i>serialHandle</i> | The serial manager module handle pointer.         |
| <i>writeHandle</i>  | The serial manager module writing handle pointer. |

Return values

|                                             |                                |
|---------------------------------------------|--------------------------------|
| <i>kStatus_SerialManager_Error</i>          | An error occurred.             |
| <i>kStatus_SerialManager_HandleConflict</i> | The writing handle was opened. |
| <i>kStatus_SerialManager_Success</i>        | The writing handle is opened.  |

Example below shows how to use this API to write data. For task 1,

```
* static uint8_t s_serialWriteHandleBuffer1[SERIAL_MANAGER_WRITE_HANDLE_SIZE
*];
* static serial_write_handle_t s_serialWriteHandle1 = &s_serialWriteHandleBuffer1[0];
* static uint8_t s_nonBlockingWelcome1[] = "This is non-blocking writing log for task1!\r\n";
* SerialManager_OpenWriteHandle(serialHandle, s_serialWriteHandle1);
* SerialManager_InstallTxCallback(s_serialWriteHandle1, Task1_SerialManagerTxCallback,
* s_serialWriteHandle1);
* SerialManager_WriteNonBlocking(s_serialWriteHandle1, s_nonBlockingWelcome1, sizeof(
* s_nonBlockingWelcome1) - 1);
*
```

For task 2,

```
* static uint8_t s_serialWriteHandleBuffer2[SERIAL_MANAGER_WRITE_HANDLE_SIZE
*];
* static serial_write_handle_t s_serialWriteHandle2 = &s_serialWriteHandleBuffer2[0];
* static uint8_t s_nonBlockingWelcome2[] = "This is non-blocking writing log for task2!\r\n";
* SerialManager_OpenWriteHandle(serialHandle, s_serialWriteHandle2);
* SerialManager_InstallTxCallback(s_serialWriteHandle2, Task2_SerialManagerTxCallback,
* s_serialWriteHandle2);
* SerialManager_WriteNonBlocking(s_serialWriteHandle2, s_nonBlockingWelcome2, sizeof(
* s_nonBlockingWelcome2) - 1);
*
```



#### 34.4.4 serial\_manager\_status\_t SerialManager\_CloseWriteHandle ( serial\_write\_handle\_t *writeHandle* )

This function Closes a writing handle for the serial manager module.

## Function Documentation

### Parameters

|                    |                                                   |
|--------------------|---------------------------------------------------|
| <i>writeHandle</i> | The serial manager module writing handle pointer. |
|--------------------|---------------------------------------------------|

### Return values

|                                      |                               |
|--------------------------------------|-------------------------------|
| <i>kStatus_SerialManager_Success</i> | The writing handle is closed. |
|--------------------------------------|-------------------------------|

### 34.4.5 serial\_manager\_status\_t SerialManager\_OpenReadHandle ( serial\_handle\_t serialHandle, serial\_read\_handle\_t readHandle )

This function Opens a reading handle for the serial manager module. The reading handle can not be opened multiple at the same time. The error code `kStatus_SerialManager_Busy` would be returned when the previous reading handle is not closed. And There can only be one buffer for receiving for the reading handle at the same time.

### Parameters

|                     |                                                   |
|---------------------|---------------------------------------------------|
| <i>serialHandle</i> | The serial manager module handle pointer.         |
| <i>readHandle</i>   | The serial manager module reading handle pointer. |

### Return values

|                                      |                                        |
|--------------------------------------|----------------------------------------|
| <i>kStatus_SerialManager_Error</i>   | An error occurred.                     |
| <i>kStatus_SerialManager_Success</i> | The reading handle is opened.          |
| <i>kStatus_SerialManager_Busy</i>    | Previous reading handle is not closed. |

Example below shows how to use this API to read data.

```
* static uint8_t s_serialReadHandleBuffer[SERIAL_MANAGER_READ_HANDLE_SIZE];
* static serial_read_handle_t s_serialReadHandle = &s_serialReadHandleBuffer[0];
* SerialManager_OpenReadHandle(serialHandle, s_serialReadHandle);
* static uint8_t s_nonBlockingBuffer[64];
* SerialManager_InstallRxCallback(s_serialReadHandle, APP_SerialManagerRxCallback, s_serialReadHandle);
* SerialManager_ReadNonBlocking(s_serialReadHandle, s_nonBlockingBuffer, sizeof(s_nonBlockingBuffer));
*
```

### 34.4.6 serial\_manager\_status\_t SerialManager\_CloseReadHandle ( serial\_read\_handle\_t readHandle )

This function Closes a reading for the serial manager module.

## Parameters

|                   |                                                   |
|-------------------|---------------------------------------------------|
| <i>readHandle</i> | The serial manager module reading handle pointer. |
|-------------------|---------------------------------------------------|

## Return values

|                                      |                               |
|--------------------------------------|-------------------------------|
| <i>kStatus_SerialManager_Success</i> | The reading handle is closed. |
|--------------------------------------|-------------------------------|

### 34.4.7 serial\_manager\_status\_t SerialManager\_WriteBlocking ( serial\_write\_handle\_t writeHandle, uint8\_t \* buffer, uint32\_t length )

This is a blocking function, which polls the sending queue, waits for the sending queue to be empty. This function sends data using an interrupt method. The interrupt of the hardware could not be disabled. And There can only one buffer for transmission for the writing handle at the same time.

## Note

The function [SerialManager\\_WriteBlocking](#) and the function `#SerialManager_WriteNonBlocking` cannot be used at the same time. And, the function `#SerialManager_CancelWriting` cannot be used to abort the transmission of this function.

## Parameters

|                    |                                           |
|--------------------|-------------------------------------------|
| <i>writeHandle</i> | The serial manager module handle pointer. |
| <i>buffer</i>      | Start address of the data to write.       |
| <i>length</i>      | Length of the data to write.              |

## Return values

|                                      |                                                                  |
|--------------------------------------|------------------------------------------------------------------|
| <i>kStatus_SerialManager_Success</i> | Successfully sent all data.                                      |
| <i>kStatus_SerialManager_Busy</i>    | Previous transmission still not finished; data not all sent yet. |

## Function Documentation

|                                     |                    |
|-------------------------------------|--------------------|
| <i>kStatus_SerialManager_-Error</i> | An error occurred. |
|-------------------------------------|--------------------|

### 34.4.8 serial\_manager\_status\_t SerialManager\_ReadBlocking ( serial\_read\_handle\_t readHandle, uint8\_t \* buffer, uint32\_t length )

This is a blocking function, which polls the receiving buffer, waits for the receiving buffer to be full. This function receives data using an interrupt method. The interrupt of the hardware could not be disabled. And There can only one buffer for receiving for the reading handle at the same time.

#### Note

The function [SerialManager\\_ReadBlocking](#) and the function `#SerialManager_ReadNonBlocking` cannot be used at the same time. And, the function `#SerialManager_CancelReading` cannot be used to abort the transmission of this function.

#### Parameters

|                   |                                                       |
|-------------------|-------------------------------------------------------|
| <i>readHandle</i> | The serial manager module handle pointer.             |
| <i>buffer</i>     | Start address of the data to store the received data. |
| <i>length</i>     | The length of the data to be received.                |

#### Return values

|                                       |                                                                      |
|---------------------------------------|----------------------------------------------------------------------|
| <i>kStatus_SerialManager_-Success</i> | Successfully received all data.                                      |
| <i>kStatus_SerialManager_-Busy</i>    | Previous transmission still not finished; data not all received yet. |
| <i>kStatus_SerialManager_-Error</i>   | An error occurred.                                                   |

### 34.4.9 serial\_manager\_status\_t SerialManager\_EnterLowpower ( serial\_handle\_t serialHandle )

This function is used to prepare to enter low power consumption.

## Parameters

|                     |                                           |
|---------------------|-------------------------------------------|
| <i>serialHandle</i> | The serial manager module handle pointer. |
|---------------------|-------------------------------------------|

## Return values

|                                      |                       |
|--------------------------------------|-----------------------|
| <i>kStatus_SerialManager_Success</i> | Successful operation. |
|--------------------------------------|-----------------------|

#### 34.4.10 **serial\_manager\_status\_t** SerialManager\_ExitLowpower ( **serial\_handle\_t** *serialHandle* )

This function is used to restore from low power consumption.

## Parameters

|                     |                                           |
|---------------------|-------------------------------------------|
| <i>serialHandle</i> | The serial manager module handle pointer. |
|---------------------|-------------------------------------------|

## Return values

|                                      |                       |
|--------------------------------------|-----------------------|
| <i>kStatus_SerialManager_Success</i> | Successful operation. |
|--------------------------------------|-----------------------|

### 34.5 Serial Port Uart

#### 34.5.1 Overview

#### Data Structures

- struct [serial\\_port\\_uart\\_config\\_t](#)  
*serial port uart config struct [More...](#)*

#### Macros

- #define [SERIAL\\_PORT\\_UART\\_HANDLE\\_SIZE](#) (4U)  
*serial port uart handle size*

#### Enumerations

- enum [serial\\_port\\_uart\\_parity\\_mode\\_t](#) {  
    [kSerialManager\\_UartParityDisabled](#) = 0x0U,  
    [kSerialManager\\_UartParityEven](#) = 0x1U,  
    [kSerialManager\\_UartParityOdd](#) = 0x2U }  
*serial port uart parity mode*
- enum [serial\\_port\\_uart\\_stop\\_bit\\_count\\_t](#) {  
    [kSerialManager\\_UartOneStopBit](#) = 0U,  
    [kSerialManager\\_UartTwoStopBit](#) = 1U }  
*serial port uart stop bit count*

#### 34.5.2 Data Structure Documentation

##### 34.5.2.1 struct serial\_port\_uart\_config\_t

#### Data Fields

- uint32\_t [clockRate](#)  
*clock rate*
- uint32\_t [baudRate](#)  
*baud rate*
- [serial\\_port\\_uart\\_parity\\_mode\\_t](#) [parityMode](#)  
*Parity mode, disabled (default), even, odd.*
- [serial\\_port\\_uart\\_stop\\_bit\\_count\\_t](#) [stopBitCount](#)  
*Number of stop bits, 1 stop bit (default) or 2 stop bits.*
- uint8\_t [instance](#)  
*Instance (0 - UART0, 1 - UART1, ...), detail information please refer to the SOC corresponding RM.*
- uint8\_t [enableRx](#)  
*Enable RX.*
- uint8\_t [enableTx](#)

*Enable TX.*

#### 34.5.2.1.0.34 Field Documentation

34.5.2.1.0.34.1 `uint8_t serial_port_uart_config_t::instance`

### 34.5.3 Enumeration Type Documentation

#### 34.5.3.1 `enum serial_port_uart_parity_mode_t`

Enumerator

*kSerialManager\_UartParityDisabled* Parity disabled.  
*kSerialManager\_UartParityEven* Parity even enabled.  
*kSerialManager\_UartParityOdd* Parity odd enabled.

#### 34.5.3.2 `enum serial_port_uart_stop_bit_count_t`

Enumerator

*kSerialManager\_UartOneStopBit* One stop bit.  
*kSerialManager\_UartTwoStopBit* Two stop bits.

### 34.6 Serial Port USB

#### 34.6.1 Overview

##### Modules

- [USB Device Configuration](#)

##### Data Structures

- struct [serial\\_port\\_usb\\_cdc\\_config\\_t](#)  
*serial port usb config struct [More...](#)*

##### Macros

- #define [SERIAL\\_PORT\\_USB\\_CDC\\_HANDLE\\_SIZE](#) (72)  
*serial port usb handle size*
- #define [USB\\_DEVICE\\_INTERRUPT\\_PRIORITY](#) (3U)  
*USB interrupt priority.*

##### Enumerations

- enum [serial\\_port\\_usb\\_cdc\\_controller\\_index\\_t](#) {  
    [kSerialManager\\_UsbControllerKhci0](#) = 0U,  
    [kSerialManager\\_UsbControllerKhci1](#) = 1U,  
    [kSerialManager\\_UsbControllerEhci0](#) = 2U,  
    [kSerialManager\\_UsbControllerEhci1](#) = 3U,  
    [kSerialManager\\_UsbControllerLpcIp3511Fs0](#) = 4U,  
    [kSerialManager\\_UsbControllerLpcIp3511Fs1](#) = 5U,  
    [kSerialManager\\_UsbControllerLpcIp3511Hs0](#) = 6U,  
    [kSerialManager\\_UsbControllerLpcIp3511Hs1](#) = 7U,  
    [kSerialManager\\_UsbControllerOhci0](#) = 8U,  
    [kSerialManager\\_UsbControllerOhci1](#) = 9U,  
    [kSerialManager\\_UsbControllerIp3516Hs0](#) = 10U,  
    [kSerialManager\\_UsbControllerIp3516Hs1](#) = 11U }  
*USB controller ID.*



## 34.6.2 Data Structure Documentation

### 34.6.2.1 struct serial\_port\_usb\_cdc\_config\_t

#### Data Fields

- [serial\\_port\\_usb\\_cdc\\_controller\\_index\\_t](#) controllerIndex  
controller index

## 34.6.3 Enumeration Type Documentation

### 34.6.3.1 enum serial\_port\_usb\_cdc\_controller\_index\_t

#### Enumerator

**kSerialManager\_UsbControllerKhci0** KHCI 0U.

**kSerialManager\_UsbControllerKhci1** KHCI 1U, Currently, there are no platforms which have two KHCI IPs, this is reserved to be used in the future.

**kSerialManager\_UsbControllerEhci0** EHCI 0U.

**kSerialManager\_UsbControllerEhci1** EHCI 1U, Currently, there are no platforms which have two EHCI IPs, this is reserved to be used in the future.

**kSerialManager\_UsbControllerLpcIp3511Fs0** LPC USB IP3511 FS controller 0.

**kSerialManager\_UsbControllerLpcIp3511Fs1** LPC USB IP3511 FS controller 1, there are no platforms which have two IP3511 IPs, this is reserved to be used in the future.

**kSerialManager\_UsbControllerLpcIp3511Hs0** LPC USB IP3511 HS controller 0.

**kSerialManager\_UsbControllerLpcIp3511Hs1** LPC USB IP3511 HS controller 1, there are no platforms which have two IP3511 IPs, this is reserved to be used in the future.

**kSerialManager\_UsbControllerOhci0** OHCI 0U.

**kSerialManager\_UsbControllerOhci1** OHCI 1U, Currently, there are no platforms which have two OHCI IPs, this is reserved to be used in the future.

**kSerialManager\_UsbControllerIp3516Hs0** IP3516HS 0U.

**kSerialManager\_UsbControllerIp3516Hs1** IP3516HS 1U, Currently, there are no platforms which have two IP3516HS IPs, this is reserved to be used in the future.

### 34.6.4 USB Device Configuration

#### 34.6.4.1 Overview

##### Macros

- #define [USB\\_DEVICE\\_CONFIG\\_SELF\\_POWER](#) (1U)  
*Whether device is self power.*
- #define [USB\\_DEVICE\\_CONFIG\\_ENDPOINTS](#) (4U)  
*How many endpoints are supported in the stack.*
- #define [USB\\_DEVICE\\_CONFIG\\_USE\\_TASK](#) (0U)  
*Whether the device task is enabled.*
- #define [USB\\_DEVICE\\_CONFIG\\_MAX\\_MESSAGES](#) (8U)  
*How many the notification message are supported when the device task is enabled.*
- #define [USB\\_DEVICE\\_CONFIG\\_USB20\\_TEST\\_MODE](#) (0U)  
*Whether test mode enabled.*
- #define [USB\\_DEVICE\\_CONFIG\\_CV\\_TEST](#) (0U)  
*Whether device CV test is enabled.*
- #define [USB\\_DEVICE\\_CONFIG\\_COMPLIANCE\\_TEST](#) (0U)  
*Whether device compliance test is enabled.*
- #define [USB\\_DEVICE\\_CONFIG\\_KEEP\\_ALIVE\\_MODE](#) (0U)  
*Whether the keep alive feature enabled.*
- #define [USB\\_DEVICE\\_CONFIG\\_BUFFER\\_PROPERTY\\_CACHEABLE](#) (0U)  
*Whether the transfer buffer is cache-enabled or not.*
- #define [USB\\_DEVICE\\_CONFIG\\_LOW\\_POWER\\_MODE](#) (0U)  
*Whether the low power mode is enabled or not.*
- #define [USB\\_DEVICE\\_CONFIG\\_REMOTE\\_WAKEUP](#) (0U)  
*The device remote wakeup is unsupported.*
- #define [USB\\_DEVICE\\_CONFIG\\_DETACH\\_ENABLE](#) (0U)  
*Whether the device detached feature is enabled or not.*
- #define [USB\\_DEVICE\\_CONFIG\\_ERROR\\_HANDLING](#) (0U)  
*Whether handle the USB bus error.*
- #define [USB\\_DEVICE\\_CHARGER\\_DETECT\\_ENABLE](#) (0U)  
*Whether the device charger detect feature is enabled or not.*

##### class instance define

- #define [USB\\_DEVICE\\_CONFIG\\_HID](#) (0U)  
*HID instance count.*
- #define [USB\\_DEVICE\\_CONFIG\\_CDC\\_ACM](#) (1U)  
*CDC ACM instance count.*
- #define [USB\\_DEVICE\\_CONFIG\\_MSC](#) (0U)  
*MSC instance count.*
- #define [USB\\_DEVICE\\_CONFIG\\_AUDIO](#) (0U)  
*Audio instance count.*
- #define [USB\\_DEVICE\\_CONFIG\\_PHDC](#) (0U)  
*PHDC instance count.*
- #define [USB\\_DEVICE\\_CONFIG\\_VIDEO](#) (0U)  
*Video instance count.*
- #define [USB\\_DEVICE\\_CONFIG\\_CCID](#) (0U)

- *CCID instance count.*  
• #define **USB\_DEVICE\_CONFIG\_PRINTER** (0U)
- *Printer instance count.*  
• #define **USB\_DEVICE\_CONFIG\_DFU** (0U)
- *DFU instance count.*

### 34.6.4.2 Macro Definition Documentation

#### 34.6.4.2.1 #define USB\_DEVICE\_CONFIG\_SELF\_POWER (1U)

1U supported, 0U not supported

#### 34.6.4.2.2 #define USB\_DEVICE\_CONFIG\_ENDPOINTS (4U)

#### 34.6.4.2.3 #define USB\_DEVICE\_CONFIG\_USE\_TASK (0U)

#### 34.6.4.2.4 #define USB\_DEVICE\_CONFIG\_MAX\_MESSAGES (8U)

#### 34.6.4.2.5 #define USB\_DEVICE\_CONFIG\_USB20\_TEST\_MODE (0U)

#### 34.6.4.2.6 #define USB\_DEVICE\_CONFIG\_CV\_TEST (0U)

#### 34.6.4.2.7 #define USB\_DEVICE\_CONFIG\_COMPLIANCE\_TEST (0U)

If the macro is enabled, the test mode and CV test macroses will be set.

#### 34.6.4.2.8 #define USB\_DEVICE\_CONFIG\_KEEP\_ALIVE\_MODE (0U)

#### 34.6.4.2.9 #define USB\_DEVICE\_CONFIG\_BUFFER\_PROPERTY\_CACHEABLE (0U)

#### 34.6.4.2.10 #define USB\_DEVICE\_CONFIG\_LOW\_POWER\_MODE (0U)

#### 34.6.4.2.11 #define USB\_DEVICE\_CONFIG\_REMOTE\_WAKEUP (0U)

#### 34.6.4.2.12 #define USB\_DEVICE\_CONFIG\_DETACH\_ENABLE (0U)

#### 34.6.4.2.13 #define USB\_DEVICE\_CONFIG\_ERROR\_HANDLING (0U)

#### 34.6.4.2.14 #define USB\_DEVICE\_CHARGER\_DETECT\_ENABLE (0U)

## Serial Port SWO

### 34.7 Serial Port SWO

#### 34.7.1 Overview

#### Data Structures

- struct [serial\\_port\\_swo\\_config\\_t](#)  
*serial port swo config struct [More...](#)*

#### Macros

- #define [SERIAL\\_PORT\\_SWO\\_HANDLE\\_SIZE](#) (12U)  
*serial port swo handle size*

#### Enumerations

- enum [serial\\_port\\_swo\\_protocol\\_t](#) {  
    [kSerialManager\\_SwoProtocolManchester](#) = 1U,  
    [kSerialManager\\_SwoProtocolNrz](#) = 2U }  
*serial port swo protocol*

#### 34.7.2 Data Structure Documentation

##### 34.7.2.1 struct serial\_port\_swo\_config\_t

#### Data Fields

- uint32\_t [clockRate](#)  
*clock rate*
- uint32\_t [baudRate](#)  
*baud rate*
- uint32\_t [port](#)  
*Port used to transfer data.*
- [serial\\_port\\_swo\\_protocol\\_t](#) [protocol](#)  
*SWO protocol.*

#### 34.7.3 Enumeration Type Documentation

##### 34.7.3.1 enum serial\_port\_swo\_protocol\_t

Enumerator

***kSerialManager\_SwoProtocolManchester*** SWO Manchester protocol.  
***kSerialManager\_SwoProtocolNrz*** SWO UART/NRZ protocol.

## 34.8 Serial Port Virtual USB

### 34.8.1 Overview

This chapter describes how to redirect the serial manager stream to application CDC. The weak functions can be implemented by application to redirect the serial manager stream. The weak functions are following,

USB\_DeviceVcomInit - Initialize the cdc vcom.

USB\_DeviceVcomDeinit - De-initialize the cdc vcom.

USB\_DeviceVcomWrite - Write data with non-blocking mode. After data is sent, the installed TX callback should be called with the result.

USB\_DeviceVcomRead - Read data with non-blocking mode. After data is received, the installed RX callback should be called with the result.

USB\_DeviceVcomCancelWrite - Cancel write request.

USB\_DeviceVcomInstallTxCallback - Install TX callback.

USB\_DeviceVcomInstallRxCallback - Install RX callback.

USB\_DeviceVcomIsrFunction - The hardware ISR function.

### Data Structures

- struct [serial\\_port\\_usb\\_cdc\\_virtual\\_config\\_t](#)  
*serial port usb config struct [More...](#)*

### Macros

- #define [SERIAL\\_PORT\\_USB\\_VIRTUAL\\_HANDLE\\_SIZE](#) (40U)  
*serial port USB handle size*

### Enumerations

- enum `serial_port_usb_cdc_virtual_controller_index_t` {  
    `kSerialManager_UsbVirtualControllerKhci0` = 0U,  
    `kSerialManager_UsbVirtualControllerKhci1` = 1U,  
    `kSerialManager_UsbVirtualControllerEhci0` = 2U,  
    `kSerialManager_UsbVirtualControllerEhci1` = 3U,  
    `kSerialManager_UsbVirtualControllerLpcIp3511Fs0` = 4U,  
    `kSerialManager_UsbVirtualControllerLpcIp3511Fs1`,  
    `kSerialManager_UsbVirtualControllerLpcIp3511Hs0` = 6U,  
    `kSerialManager_UsbVirtualControllerLpcIp3511Hs1`,  
    `kSerialManager_UsbVirtualControllerOhci0` = 8U,  
    `kSerialManager_UsbVirtualControllerOhci1` = 9U,  
    `kSerialManager_UsbVirtualControllerIp3516Hs0` = 10U,  
    `kSerialManager_UsbVirtualControllerIp3516Hs1` = 11U }

*USB controller ID.*

### Variables

- `serial_port_usb_cdc_virtual_controller_index_t serial_port_usb_cdc_virtual_config_t::controllerIndex`  
*controller index*

## 34.8.2 Data Structure Documentation

### 34.8.2.1 struct serial\_port\_usb\_cdc\_virtual\_config\_t

#### Data Fields

- `serial_port_usb_cdc_virtual_controller_index_t controllerIndex`  
*controller index*

## 34.8.3 Enumeration Type Documentation

### 34.8.3.1 enum serial\_port\_usb\_cdc\_virtual\_controller\_index\_t

Enumerator

***kSerialManager\_UsbVirtualControllerKhci0*** KHCI 0U.

***kSerialManager\_UsbVirtualControllerKhci1*** KHCI 1U, Currently, there are no platforms which have two KHCI IPs, this is reserved to be used in the future.

***kSerialManager\_UsbVirtualControllerEhci0*** EHCI 0U.

***kSerialManager\_UsbVirtualControllerEhci1*** EHCI 1U, Currently, there are no platforms which have two EHCI IPs, this is reserved to be used in the future.

***kSerialManager\_UsbVirtualControllerLpcIp3511Fs0*** LPC USB IP3511 FS controller 0.

***kSerialManager\_UsbVirtualControllerLpcIp3511Fs1*** LPC USB IP3511 FS controller 1, there are no platforms which have two IP3511 IPs, this is reserved to be used in the future.

***kSerialManager\_UsbVirtualControllerLpcIp3511Hs0*** LPC USB IP3511 HS controller 0.

***kSerialManager\_UsbVirtualControllerLpcIp3511Hs1*** LPC USB IP3511 HS controller 1, there are no platforms which have two IP3511 IPs, this is reserved to be used in the future.

***kSerialManager\_UsbVirtualControllerOhci0*** OHCI 0U.

***kSerialManager\_UsbVirtualControllerOhci1*** OHCI 1U, Currently, there are no platforms which have two OHCI IPs, this is reserved to be used in the future.

***kSerialManager\_UsbVirtualControllerIp3516Hs0*** IP3516HS 0U.

***kSerialManager\_UsbVirtualControllerIp3516Hs1*** IP3516HS 1U, Currently, there are no platforms which have two IP3516HS IPs, this is reserved to be used in the future.





## Chapter 35

# NTAG: integrated NTAG

### 35.1 Overview

The MCUXpresso SDK provides a peripheral driver for the integrated NTAG module of MCUXpresso SDK devices.

### Data Structures

- struct [ntag\\_config\\_t](#)  
*NTAG user configuration. [More...](#)*

### Macros

- #define [RFT1503](#)  
*FD polling implementation options.*
- #define [NTAG\\_IRQ](#) [NFCTag\\_IRQn](#)  
*NTAG FD interrupt line.*

### Typedefs

- typedef void(\* [ntag\\_field\\_detect\\_callback\\_t](#))([ntag\\_field\\_detect\\_t](#) fd, void \*userData)  
*NTAG Field Detect callback typedef.*

### Enumerations

- enum [ntag\\_state\\_t](#) {  
    [kNTAG\\_StateActive](#),  
    [kNTAG\\_StateInactive](#) }  
*ntag operating state*
- enum [ntag\\_field\\_detect\\_t](#) {  
    [kNTAG\\_FieldDetectIn](#),  
    [kNTAG\\_FieldDetectOut](#) }  
*Field Detect line state.*

### Functions

- void [NTAG\\_GetDefaultConfig](#) ([ntag\\_config\\_t](#) \*config)  
*Sets the NTAG configuration structure to default values.*
- void [NTAG\\_Init](#) (const [ntag\\_config\\_t](#) \*config)  
*Initialize the internal NTAG peripheral.*
- [ntag\\_field\\_detect\\_t](#) [NTAG\\_PollFieldDetect](#) (void)  
*Poll state of Field Detect line.*
- void [NTAG\\_SetState](#) ([ntag\\_state\\_t](#) state)  
*Configure NTAG operating state.*

## Function Documentation

### 35.2 Data Structure Documentation

#### 35.2.1 struct ntag\_config\_t

##### Data Fields

- [ntag\\_field\\_detect\\_callback\\_t callback](#)  
*A callback function called at the transfer event.*
- void \* [userData](#)  
*A callback parameter passed to the callback function.*

##### 35.2.1.0.0.1 Field Documentation

35.2.1.0.0.1.1 ntag\_field\_detect\_callback\_t ntag\_config\_t::callback

35.2.1.0.0.1.2 void\* ntag\_config\_t::userData

### 35.3 Macro Definition Documentation

#### 35.3.1 #define RFT1503

### 35.4 Typedef Documentation

35.4.1 typedef void(\* ntag\_field\_detect\_callback\_t)(ntag\_field\_detect\_t fd, void \*userData)

### 35.5 Enumeration Type Documentation

#### 35.5.1 enum ntag\_state\_t

##### Enumerator

*kNTAG\_StateActive* NTAG powered on (ready for I2C communication)  
*kNTAG\_StateInactive* NTAG powered off or MCU in low power state.

#### 35.5.2 enum ntag\_field\_detect\_t

##### Enumerator

*kNTAG\_FieldDetectIn* NTAG is in field.  
*kNTAG\_FieldDetectOut* NTAG is out of field.

### 35.6 Function Documentation

35.6.1 void NTAG\_GetDefaultConfig ( ntag\_config\_t \* config )

Parameters

|               |                                           |
|---------------|-------------------------------------------|
| <i>config</i> | A pointer to the configuration structure. |
|---------------|-------------------------------------------|

### 35.6.2 void NTAG\_Init ( const ntag\_config\_t \* *config* )

Parameters

|               |                                               |
|---------------|-----------------------------------------------|
| <i>config</i> | A pointer to the NTAG configuration structure |
|---------------|-----------------------------------------------|

### 35.6.3 ntag\_field\_detect\_t NTAG\_PollFieldDetect ( void )

### 35.6.4 void NTAG\_SetState ( ntag\_state\_t *state* )

Parameters

|              |                      |
|--------------|----------------------|
| <i>state</i> | NTAG operating state |
|--------------|----------------------|



## Chapter 36

### Wtimer

#### 36.1 Overview

##### Files

- file [fsl\\_wtimer.h](#)

##### Driver version

Wake timers provide wakeup capabilities in sleep modes where 32KHz clock is kept active. Wake timer 0 is a 48bit based counter while wake timer 1 is 32bit based counter. A special API functions [WTIMER\\_StartTimerLarge\(0](#) and [WTIMER\\_StartTimerLarge\(\)](#) are provided to access the 48bit counter. The Wake timer 1 is to be used by the PWRM framework. It shall not be used by the Application directly. API provides the capability to enable and disable interrupts. The application shall implement the Wake timer ISR on its side. The wake timer ISR prototypes are : [void WAKE\\_UP\\_TIMER0\\_IRQHandler\(void\)](#); and [void WAKE\\_UP\\_TIMER1\\_IRQHandler\(void\)](#); The Application shall correctly the 32KHz source among the FRO32 or Crystal 32KHz using [CLOCK\\_EnableClock\(\)](#) API in [fsl\\_clock.h](#) The API provides the capability to calibrate the 32KHz clock versus a high reference clock (32MHz crystal).

- #define [FSL\\_WTIMER\\_DRIVER\\_VERSION](#) ([MAKE\\_VERSION](#)(2, 0, 0))  
*Version 2.0.0.*

##### Initialization and deinitialization

- void [WTIMER\\_Init](#) (void)  
*Enable the clocks to the peripheral (functional clock and AHB clock)*
- void [WTIMER\\_DeInit](#) (void)  
*Disable the clocks to the peripheral (functional clock and AHB clock)*
- void [WTIMER\\_EnableInterrupts](#) (WTIMER\_timer\_id\_t timer\_id)  
*Enable the selected Timer interrupts.*
- WTIMER\_status\_t [WTIMER\\_GetStatusFlags](#) (WTIMER\_timer\_id\_t timer\_id)  
*Gets the Timer status flags.*
- void [WTIMER\\_ClearStatusFlags](#) (WTIMER\_timer\_id\_t timer\_id)  
*Clears the Timer status flags if expired and clear the pending interrupt if active it needs to be called in ISR.*
- void [WTIMER\\_StartTimer](#) (WTIMER\_timer\_id\_t timer\_id, uint32\_t count)  
*Starts the Timer counter.*
- void [WTIMER\\_StopTimer](#) (WTIMER\_timer\_id\_t timer\_id)  
*Stops the Timer counter.*
- uint32\_t [WTIMER\\_CalibrateTimer](#) (void)  
*Calibrate the 32KHz clock to be used by the wake timer versus the 32MHz crystal clock source The Application shall switch OFF the 32MHz clock if no longer used by the chip using [CLOCK\\_DisableClock\(\)](#) in [fsl\\_clock.h](#).*
- uint32\_t [WTIMER\\_ReadTimer](#) (WTIMER\_timer\_id\_t timer\_id)  
*Read the LSB counter of the wake timer This API is unsafe.*

## Function Documentation

- uint32\_t [WTIMER\\_ReadTimerSafe](#) (WTIMER\_timer\_id\_t timer\_id)  
*Read the LSB counter of the wake timer API checks the next counter update (next 32KHz clock edge) so the value is up to date. Important note : The counter shall be running otherwise, the API gets locked and never return.*

## 36.2 Function Documentation

### 36.2.1 void WTIMER\_Init ( void )

#### Note

This function does not reset the wake timer peripheral. Wake timer reset is done in PWRM\_vCold-Start() from the PWRM framework module if integrated. If PWRM framework module is integrated, [WTIMER\\_Init\(\)](#) is called in PWRM\_vInit() for power modes with Oscillator ON.

### 36.2.2 void WTIMER\_DeInit ( void )

#### Note

This function does not reset the wake timer peripheral.

### 36.2.3 void WTIMER\_EnableInterrupts ( WTIMER\_timer\_id\_t timer\_id )

The application shall implement the Wake timer ISR

#### Parameters

|                 |           |
|-----------------|-----------|
| <i>timer_id</i> | Wtimer Id |
|-----------------|-----------|

### 36.2.4 WTIMER\_status\_t WTIMER\_GetStatusFlags ( WTIMER\_timer\_id\_t timer\_id )

#### Parameters

|                 |           |
|-----------------|-----------|
| <i>timer_id</i> | Wtimer Id |
|-----------------|-----------|

#### Returns

The status flags.

### 36.2.5 void WTIMER\_ClearStatusFlags ( WTIMER\_timer\_id\_t timer\_id )

Parameters

|                 |           |
|-----------------|-----------|
| <i>timer_id</i> | Wtimer Id |
|-----------------|-----------|

### 36.2.6 void WTIMER\_StartTimer ( WTIMER\_timer\_id\_t *timer\_id*, uint32\_t *count* )

The function performs: -stop the timer if running, clear the status and interrupt flag if set ([WTIMER\\_ClearStatusFlags\(\)](#)) -set the counter value -start the timer

Parameters

|                 |                                                 |
|-----------------|-------------------------------------------------|
| <i>timer_id</i> | Wtimer Id                                       |
| <i>count</i>    | number of 32KHz clock periods before expiration |

### 36.2.7 void WTIMER\_StopTimer ( WTIMER\_timer\_id\_t *timer\_id* )

Parameters

|                 |           |
|-----------------|-----------|
| <i>timer_id</i> | Wtimer Id |
|-----------------|-----------|

### 36.2.8 uint32\_t WTIMER\_CalibrateTimer ( void )

Returns

32KHz clock frequency (number of 32KHz clock in one sec) - expect to have 32768

### 36.2.9 uint32\_t WTIMER\_ReadTimer ( WTIMER\_timer\_id\_t *timer\_id* )

If the counter has just been started, the counter value may not be up to date until the next 32KHz clock edge. Use [WTIMER\\_ReadTimerSafe\(\)](#) instead

Parameters

|                 |           |
|-----------------|-----------|
| <i>timer_id</i> | Wtimer Id |
|-----------------|-----------|

Returns

counter value - number of ticks before expiration if running

**36.2.10**    `uint32_t` **WTIMER\_ReadTimerSafe** ( `WTIMER_timer_id_t` *timer\_id* )



## Parameters

|                 |           |
|-----------------|-----------|
| <i>timer_id</i> | Wtimer Id |
|-----------------|-----------|

## Returns

32KHz clock frequency (number of 32KHz clock in one sec) - expect to have 32768



## Chapter 37

### ROM\_API

#### 37.1 Overview

##### Files

- file [rom\\_aes.h](#)
- file [rom\\_api.h](#)
- file [rom\\_isp.h](#)
- file [rom\\_lowpower.h](#)
- file [rom\\_mpu.h](#)
- file [rom\\_pmc.h](#)
- file [rom\\_psector.h](#)
- file [rom\\_secure.h](#)

##### Data Structures

- struct [IMAGE\\_DATA\\_T](#)  
*IMAGE\_DATA\_T image node : element of single link chained list of images found in flash. [More...](#)*
- struct [ISP\\_MEM\\_FUNC\\_T](#)  
*ISP\_MEM\_FUNC\_T structure of ops method pointers instantiated per memory type. [More...](#)*
- struct [ISP\\_MEM\\_INFO\\_T](#)  
*ISP\_MEM\_INFO\_T structure of memory characteristics. [More...](#)*
- struct [ISP\\_ENC\\_STATE\\_T](#)  
*ISP\_ENC\_STATE\_T ISP structure for ciphering options : TODO check poorly tested should we advertise this ? [More...](#)*
- struct [ISP\\_STATE\\_T](#)  
*ISP\_STATE\_T structure holding the context the the curent ISP command. [More...](#)*
- struct [LPC\\_LOWPOWER\\_T](#)  
*Low Power Main Structure. [More...](#)*
- struct [LPC\\_LOWPOWER\\_LDOVOLTAGE\\_T](#)  
*Low Power Main Structure. [More...](#)*
- struct [MPU\\_Settings\\_t](#)  
*MPU\_Settings\_t structure in RAM to retrieve current MPU configuration see . [More...](#)*
- struct [image\\_directory\\_entry\\_t](#)  
*image\_directory\_entry\_t image directory found in PAGE0 (PSECT) when SSBL is involved in the loading process [More...](#)*
- struct [psector\\_header\\_t](#)  
*psector\_header\_t psector header. [More...](#)*
- struct [IMAGE\\_CERT\\_T](#)  
*IMAGE\_CERT\_T structure. [More...](#)*

##### Macros

- #define [AES\\_INB\\_FSEL](#)(n) ((n) << 16)  
*n->1=Input Text, n->2=Holding, n->3=Input Text XOR Holding*
- #define [AES\\_HOLD\\_FSEL](#)(n) ((n) << 20)

## Overview

- n->0=Counter, n->1=Input Text, n->2=Output Block, n->3=Input Text XOR Output Block*
- #define [AES\\_OUTT\\_FSEL\(n\)](#) ((n) << 24)  
*n->0=OUTT, n->1=Output Block XOR Input Text, n->2=Output Block XOR Holding*
- #define [ISP\\_INVALID\\_EXTENSION](#) (0)  
*Each ISP extension function invalid : 0 corresponds to a NULL pointer.*
- #define [ISP\\_FLAG\\_HAS\\_CRC32](#) (1 << 0)  
*Each ISP command is preceded by a 'flag' byte that tell how to verify the command.*
- #define [ISP\\_FLAG\\_SIGNED](#) (1 << 1)  
*tells that command is RSA signed and authentication is checked, if unset, the SHA256 is computed and compared against the one held in the message, which guarantees integrity*
- #define [ISP\\_FLAG\\_HAS\\_NEXT\\_HASH](#) (1 << 2)  
*tells to hold the computed hash*
- #define [LOWPOWER\\_CFG\\_MODE\\_ACTIVE](#) 0  
*ACTIVE mode.*
- #define [LOWPOWER\\_CFG\\_MODE\\_DEEPSLEEP](#) 1  
*DEEP SLEEP mode.*
- #define [LOWPOWER\\_CFG\\_MODE\\_POWERDOWN](#) 2  
*POWER DOWN mode.*
- #define [LOWPOWER\\_CFG\\_MODE\\_DEEPPowerDOWN](#) 3  
*DEEP POWER DOWN mode.*
- #define [LOWPOWER\\_CFG\\_XTAL32MSTART\\_DISABLE](#) 0  
*Disable Crystal 32 MHz automatic start when waking up from POWER DOWN and DEEP POWER DOWN modes.*
- #define [LOWPOWER\\_CFG\\_XTAL32MSTART\\_ENABLE](#) 1  
*Enable Crystal 32 MHz automatic start when waking up from POWER DOWN and DEEP POWER DOWN modes.*
- #define [LOWPOWER\\_CFG\\_FLASHPWDNMODE\\_FLASHPWDN](#) 0  
*Power down the Flash only (send CMD\_POWERDOWN to Flash controller).*
- #define [LOWPOWER\\_CFG\\_FLASHPWDNMODE\\_LDOSHUTOFF](#) 1  
*Power down the Flash ((send CMD\_POWERDOWN to Flash controller) and shutoff both Flash LDOs (Core and NV))\ (only valid in DEEP SLEEP mode)*
- #define [LOWPOWER\\_PMUPWDN\\_DCDC](#) (1UL << 0)  
*Analog Power Domains (analog components in Power Management Unit) Low Power Modes control.*
- #define [LOWPOWER\\_PMUPWDN\\_BIAS](#) (1UL << 1)  
*Power Down all Bias and references.*
- #define [LOWPOWER\\_PMUPWDN\\_LDOMEM](#) (1UL << 2)  
*Power Down Memories LDO.*
- #define [LOWPOWER\\_PMUPWDN\\_BODVBAT](#) (1UL << 3)  
*Power Down VBAT Brown Out Detector.*
- #define [LOWPOWER\\_PMUPWDN\\_FRO192M](#) (1UL << 4)  
*Power Down FRO 192 MHz.*
- #define [LOWPOWER\\_PMUPWDN\\_FRO1M](#) (1UL << 5)  
*Power Down FRO 1 MHz.*
- #define [LOWPOWER\\_PMUPWDN\\_GPADC](#) (1UL << 22)  
*Power Down General Purpose ADC.*
- #define [LOWPOWER\\_PMUPWDN\\_BODMEM](#) (1UL << 23)  
*Power Down Memories Brown Out Detector.*
- #define [LOWPOWER\\_PMUPWDN\\_BODCORE](#) (1UL << 24)  
*Power Down Core Logic Brown Out Detector.*
- #define [LOWPOWER\\_PMUPWDN\\_FRO32K](#) (1UL << 25)  
*Power Down FRO 32 KHz.*

- #define **LOWPOWER\_PMUPWDN\_XTAL32K** (1UL << 26)  
*Power Down Crystal 32 KHz.*
- #define **LOWPOWER\_PMUPWDN\_ANACOMP** (1UL << 27)  
*Power Down Analog Comparator.*
- #define **LOWPOWER\_PMUPWDN\_XTAL32M** (1UL << 28)  
*Power Down Crystal 32 MHz.*
- #define **LOWPOWER\_PMUPWDN\_TEMPSENSOR** (1UL << 29)  
*Power Down Temperature Sensor.*
- #define **LOWPOWER\_DIGPWDN\_FLASH** (1UL << 6)  
*Digital Power Domains Low Power Modes control.*
- #define **LOWPOWER\_DIGPWDN\_COMM0** (1UL << 7)  
*Power Down Digital COMM0 power domain (USART0, I2C0 and SPI0)*
- #define **LOWPOWER\_DIGPWDN\_MCU\_RET** (1UL << 8)  
*Power Down MCU Retention Power Domain (Disable Zigbee IP retention, ES1:Disable CPU retention \\  
flip-flops)*
- #define **LOWPOWER\_DIGPWDN\_ZIGBLE\_RET** (1UL << 9)  
*Power Down ZIGBEE/BLE retention Power Domain (Disable ZIGBEE/BLE retention flip-flops)*
- #define **LOWPOWER\_DIGPWDN\_SRAM0** (1UL << LOWPOWER\_DIGPWDN\_SRAM0\_IN-  
DEX)  
*Power Down SRAM 0 instance [Bank 0, 16 KB], (no retention)*
- #define **LOWPOWER\_DIGPWDN\_SRAM1** (1UL << 11)  
*Power Down SRAM 1 instance [Bank 0, 16 KB], (no retention)*
- #define **LOWPOWER\_DIGPWDN\_SRAM2** (1UL << 12)  
*Power Down SRAM 2 instance [Bank 0, 16 KB], (no retention)*
- #define **LOWPOWER\_DIGPWDN\_SRAM3** (1UL << 13)  
*Power Down SRAM 3 instance [Bank 0, 16 KB], (no retention)*
- #define **LOWPOWER\_DIGPWDN\_SRAM4** (1UL << 14)  
*Power Down SRAM 4 instance [Bank 0, 8 KB], (no retention)*
- #define **LOWPOWER\_DIGPWDN\_SRAM5** (1UL << 15)  
*Power Down SRAM 5 instance [Bank 0, 8 KB], (no retention)*
- #define **LOWPOWER\_DIGPWDN\_SRAM6** (1UL << 16)  
*Power Down SRAM 6 instance [Bank 0, 4 KB], (no retention)*
- #define **LOWPOWER\_DIGPWDN\_SRAM7** (1UL << 17)  
*Power Down SRAM 7 instance [Bank 0, 4 KB], (no retention)*
- #define **LOWPOWER\_DIGPWDN\_SRAM8** (1UL << 18)  
*Power Down SRAM 8 instance [Bank 1, 16 KB], (no retention)*
- #define **LOWPOWER\_DIGPWDN\_SRAM9** (1UL << 19)  
*Power Down SRAM 9 instance [Bank 1, 16 KB], (no retention)*
- #define **LOWPOWER\_DIGPWDN\_SRAM10** (1UL << 20)  
*Power Down SRAM 10 instance [Bank 1, 16 KB], (no retention)*
- #define **LOWPOWER\_DIGPWDN\_SRAM11** (1UL << 21)  
*Power Down SRAM 11 instance [Bank 1, 16 KB], (no retention)*
- #define **LOWPOWER\_DIGPWDN\_IO** (1UL << LOWPOWER\_DIGPWDN\_IO\_INDEX)  
*Power Down.*
- #define **LOWPOWER\_DIGPWDN\_NTAG\_FD** (1UL << LOWPOWER\_DIGPWDN\_NTAG\_F-  
D\_INDEX)  
*NTAG FD field detect Disable - need the IO source to be set too.*
- #define **LOWPOWER\_SRAM\_LPMODE\_MASK** (0xFUL)  
*LDO Voltage control in Low Power Modes.*
- #define **LOWPOWER\_VOLTAGE\_LDO\_PMU\_INDEX** 0  
*LDO Voltage control in Low Power Modes.*

## Overview

- #define [LOWPOWER\\_WAKEUPSRCINT0\\_SYSTEM\\_IRQ](#) (1UL << 0)  
*Low Power Modes Wake up Interrupt sources.*
- #define [LOWPOWER\\_WAKEUPSRCINT0\\_DMA\\_IRQ](#) (1UL << 1)  
*[DEEP SLEEP]*
- #define [LOWPOWER\\_WAKEUPSRCINT0\\_GINT\\_IRQ](#) (1UL << 2)  
*[DEEP SLEEP]*
- #define [LOWPOWER\\_WAKEUPSRCINT0\\_IRBLASTER\\_IRQ](#) (1UL << 3)  
*[DEEP SLEEP]*
- #define [LOWPOWER\\_WAKEUPSRCINT0\\_PINT0\\_IRQ](#) (1UL << 4)  
*[DEEP SLEEP]*
- #define [LOWPOWER\\_WAKEUPSRCINT0\\_PINT1\\_IRQ](#) (1UL << 5)  
*[DEEP SLEEP]*
- #define [LOWPOWER\\_WAKEUPSRCINT0\\_PINT2\\_IRQ](#) (1UL << 6)  
*[DEEP SLEEP]*
- #define [LOWPOWER\\_WAKEUPSRCINT0\\_PINT3\\_IRQ](#) (1UL << 7)  
*[DEEP SLEEP]*
- #define [LOWPOWER\\_WAKEUPSRCINT0\\_SPIFI\\_IRQ](#) (1UL << 8)  
*[DEEP SLEEP]*
- #define [LOWPOWER\\_WAKEUPSRCINT0\\_TIMER0\\_IRQ](#) (1UL << 9)  
*[DEEP SLEEP]*
- #define [LOWPOWER\\_WAKEUPSRCINT0\\_TIMER1\\_IRQ](#) (1UL << 10)  
*[DEEP SLEEP]*
- #define [LOWPOWER\\_WAKEUPSRCINT0\\_USART0\\_IRQ](#) (1UL << 11)  
*[DEEP SLEEP, POWER DOWN]*
- #define [LOWPOWER\\_WAKEUPSRCINT0\\_USART1\\_IRQ](#) (1UL << 12)  
*[DEEP SLEEP]*
- #define [LOWPOWER\\_WAKEUPSRCINT0\\_I2C0\\_IRQ](#) (1UL << 13)  
*[DEEP SLEEP, POWER DOWN]*
- #define [LOWPOWER\\_WAKEUPSRCINT0\\_I2C1\\_IRQ](#) (1UL << 14)  
*[DEEP SLEEP]*
- #define [LOWPOWER\\_WAKEUPSRCINT0\\_SPI0\\_IRQ](#) (1UL << 15)  
*[DEEP SLEEP, POWER DOWN]*
- #define [LOWPOWER\\_WAKEUPSRCINT0\\_SPI1\\_IRQ](#) (1UL << 16)  
*[DEEP SLEEP]*
- #define [LOWPOWER\\_WAKEUPSRCINT0\\_PWM0\\_IRQ](#) (1UL << 17)  
*[DEEP SLEEP]*
- #define [LOWPOWER\\_WAKEUPSRCINT0\\_PWM1\\_IRQ](#) (1UL << 18)  
*[DEEP SLEEP]*
- #define [LOWPOWER\\_WAKEUPSRCINT0\\_PWM2\\_IRQ](#) (1UL << 19)  
*[DEEP SLEEP]*
- #define [LOWPOWER\\_WAKEUPSRCINT0\\_PWM3\\_IRQ](#) (1UL << 20)  
*[DEEP SLEEP]*
- #define [LOWPOWER\\_WAKEUPSRCINT0\\_PWM4\\_IRQ](#) (1UL << 21)  
*[DEEP SLEEP]*
- #define [LOWPOWER\\_WAKEUPSRCINT0\\_PWM5\\_IRQ](#) (1UL << 22)  
*[DEEP SLEEP]*
- #define [LOWPOWER\\_WAKEUPSRCINT0\\_PWM6\\_IRQ](#) (1UL << 23)  
*[DEEP SLEEP]*
- #define [LOWPOWER\\_WAKEUPSRCINT0\\_PWM7\\_IRQ](#) (1UL << 24)  
*[DEEP SLEEP]*
- #define [LOWPOWER\\_WAKEUPSRCINT0\\_PWM8\\_IRQ](#) (1UL << 25)

- *[DEEP SLEEP]*  
• #define [LOWPOWER\\_WAKEUPSRCINT0\\_PWM9\\_IRQ](#) (1UL << 26)
- *[DEEP SLEEP]*  
• #define [LOWPOWER\\_WAKEUPSRCINT0\\_PWM10\\_IRQ](#) (1UL << 27)
- *[DEEP SLEEP]*  
• #define [LOWPOWER\\_WAKEUPSRCINT0\\_I2C2\\_IRQ](#) (1UL << 28)
- *[DEEP SLEEP]*  
• #define [LOWPOWER\\_WAKEUPSRCINT0\\_RTC\\_IRQ](#) (1UL << 29)
- *[DEEP SLEEP, POWER DOWN]*  
• #define [LOWPOWER\\_WAKEUPSRCINT0\\_NFCTAG\\_IRQ](#) (1UL << 30)
- *[DEEP SLEEP]*  
• #define [LOWPOWER\\_WAKEUPSRCINT0\\_MAILBOX\\_IRQ](#) (1UL << 31)  
*Mailbox, Wake-up from DEEP SLEEP and POWER DOWN low power mode [DEEP SLEEP, POWER DOWN].*
- #define [LOWPOWER\\_WAKEUPSRCINT1\\_ADC\\_SEQA\\_IRQ](#) (1UL << 0)
- *[DEEP SLEEP]*  
• #define [LOWPOWER\\_WAKEUPSRCINT1\\_ADC\\_SEQB\\_IRQ](#) (1UL << 1)
- *[DEEP SLEEP]*  
• #define [LOWPOWER\\_WAKEUPSRCINT1\\_ADC\\_THCMP\\_OVR\\_IRQ](#) (1UL << 2)
- *[DEEP SLEEP]*  
• #define [LOWPOWER\\_WAKEUPSRCINT1\\_DMIC\\_IRQ](#) (1UL << 3)
- *[DEEP SLEEP]*  
• #define [LOWPOWER\\_WAKEUPSRCINT1\\_HWVAD\\_IRQ](#) (1UL << 4)
- *[DEEP SLEEP]*  
• #define [LOWPOWER\\_WAKEUPSRCINT1\\_BLE\\_DP\\_IRQ](#) (1UL << 5)
- *[DEEP SLEEP]*  
• #define [LOWPOWER\\_WAKEUPSRCINT1\\_BLE\\_DP0\\_IRQ](#) (1UL << 6)
- *[DEEP SLEEP]*  
• #define [LOWPOWER\\_WAKEUPSRCINT1\\_BLE\\_DP1\\_IRQ](#) (1UL << 7)
- *[DEEP SLEEP]*  
• #define [LOWPOWER\\_WAKEUPSRCINT1\\_BLE\\_DP2\\_IRQ](#) (1UL << 8)
- *[DEEP SLEEP]*  
• #define [LOWPOWER\\_WAKEUPSRCINT1\\_BLE\\_LL\\_ALL\\_IRQ](#) (1UL << 9)
- *[DEEP SLEEP]*  
• #define [LOWPOWER\\_WAKEUPSRCINT1\\_ZIGBEE\\_MAC\\_IRQ](#) (1UL << 10)
- *[DEEP SLEEP]*  
• #define [LOWPOWER\\_WAKEUPSRCINT1\\_ZIGBEE\\_MODEM\\_IRQ](#) (1UL << 11)
- *[DEEP SLEEP]*  
• #define [LOWPOWER\\_WAKEUPSRCINT1\\_RFP\\_TMU\\_IRQ](#) (1UL << 12)
- *[DEEP SLEEP]*  
• #define [LOWPOWER\\_WAKEUPSRCINT1\\_RFP\\_AGC\\_IRQ](#) (1UL << 13)
- *[DEEP SLEEP]*  
• #define [LOWPOWER\\_WAKEUPSRCINT1\\_ISO7816\\_IRQ](#) (1UL << 14)
- *[DEEP SLEEP]*  
• #define [LOWPOWER\\_WAKEUPSRCINT1\\_ANA\\_COMP\\_IRQ](#) (1UL << 15)
- *[DEEP SLEEP]*  
• #define [LOWPOWER\\_WAKEUPSRCINT1\\_WAKE\\_UP\\_TIMER0\\_IRQ](#) (1UL << 16)
- *[DEEP SLEEP, POWER DOWN]*  
• #define [LOWPOWER\\_WAKEUPSRCINT1\\_WAKE\\_UP\\_TIMER1\\_IRQ](#) (1UL << 17)
- *[DEEP SLEEP, POWER DOWN]*  
• #define [LOWPOWER\\_WAKEUPSRCINT1\\_BLE\\_WAKE\\_TIMER\\_IRQ](#) (1UL << 22)

## Overview

- *[DEEP SLEEP, POWER DOWN]*  
• #define [LOWPOWER\\_WAKEUPSRCINT1\\_BLE\\_OSC\\_EN\\_IRQ](#) (1UL << 23)
- *[DEEP SLEEP, POWER DOWN]*  
• #define [LOWPOWER\\_WAKEUPSRCINT1\\_IO\\_IRQ](#) (1UL << 31)
- *[POWER DOWN, DEEP DOWN]*  
• #define [LOWPOWER\\_SLEEPPOSTPONE\\_FORCED](#) (1UL << 0)  
*Sleep Postpone.*
- #define [LOWPOWER\\_SLEEPPOSTPONE\\_PERIPHERALS](#) (1UL << 1)  
*USART0, USART1, SPI0, SPI1, I2C0, I2C1, I2C2 interrupts can postpone power down modes in case an \\ interrupt is pending when the processor request low power mode.*
- #define [LOWPOWER\\_SLEEPPOSTPONE\\_DMIC](#) (1UL << 0)  
*DMIC interrupt can postpone power down modes in case an interrupt is pending when the processor \\ request low power mode.*
- #define [LOWPOWER\\_SLEEPPOSTPONE\\_SDMA](#) (1UL << 1)  
*System DMA interrupt can postpone power down modes in case an interrupt is pending when the \\ processor request low power mode.*
- #define [LOWPOWER\\_SLEEPPOSTPONE\\_NFCTAG](#) (1UL << 0)  
*NFC Tag interrupt can postpone power down modes in case an interrupt is pending when the \\ processor request low power mode.*
- #define [LOWPOWER\\_SLEEPPOSTPONE\\_BLEOSC](#) (1UL << 1)  
*BLE\_OSC\_EN interrupt can postpone power down modes in case an interrupt is pending when the \\ processor request low power mode.*
- #define [LOWPOWER\\_WAKEUPIOSRC\\_PIO0](#) (1UL << 0)  
*Wake up I/O sources.*
- #define [LOWPOWER\\_GPIOLATCH\\_PIO0](#) (1UL << 0)  
*I/O whose state must be kept in Power Down mode.*
- #define [LOWPOWER\\_TIMERCFG\\_ENABLE\\_INDEX](#) 0  
*Wake up timers configuration in Low Power Modes.*
- #define [LOWPOWER\\_TIMERCFG\\_TIMER\\_ENABLE](#) 1  
*Wake Timer Enable.*
- #define [LOWPOWER\\_TIMERCFG\\_TIMER\\_WAKEUPTIMER0](#) 0  
*Primary Wake up timers configuration in Low Power Modes.*
- #define [LOWPOWER\\_TIMERCFG\\_TIMER\\_WAKEUPTIMER1](#) 1  
*Zigbee Wake up Counter 1 used as wake up source.*
- #define [LOWPOWER\\_TIMERCFG\\_TIMER\\_BLEWAKEUPTIMER](#) 2  
*BLE Wake up Counter used as wake up source.*
- #define [LOWPOWER\\_TIMERCFG\\_TIMER\\_RTC1KHZ](#) 3  
*1 KHz Real Time Counter (RTC) used as wake up source*
- #define [LOWPOWER\\_TIMERCFG\\_TIMER\\_RTC1HZ](#) 4  
*1 Hz Real Time Counter (RTC) used as wake up source*
- #define [LOWPOWER\\_TIMERCFG\\_2ND\\_TIMER\\_WAKEUPTIMER0](#) 0  
*Secondary Wake up timers configuration in Low Power Modes.*
- #define [LOWPOWER\\_TIMERCFG\\_2ND\\_TIMER\\_WAKEUPTIMER1](#) 1  
*Zigbee Wake up Counter 1 used as secondary wake up source.*
- #define [LOWPOWER\\_TIMERCFG\\_2ND\\_TIMER\\_BLEWAKEUPTIMER](#) 2  
*BLE Wake up Counter used as secondary wake up source.*
- #define [LOWPOWER\\_TIMERCFG\\_2ND\\_TIMER\\_RTC1KHZ](#) 3  
*1 KHz Real Time Counter (RTC) used as secondary wake up source*
- #define [LOWPOWER\\_TIMERCFG\\_2ND\\_TIMER\\_RTC1HZ](#) 4  
*1 Hz Real Time Counter (RTC) used as secondary wake up source*
- #define [LOWPOWER\\_TIMERCFG\\_OSC32K\\_FRO32KHZ](#) 0



- *Wake up Timers uses FRO 32 KHz as clock source.*
- #define [LOWPOWER\\_TIMERCFG\\_OSC32K\\_XTAL32KHZ](#) 1
- *Wake up Timers uses Chrystal 32 KHz as clock source.*
- #define [LOWPOWER\\_TIMERBLECFG\\_RADIOEN\\_INDEX](#) 0
- *BLE Wake up timers configuration in Low Power Modes.*
- #define [RD\\_RIGHT](#) (1<<0)
- *bits for access right*
- #define [PSECTOR\\_PAGE\\_WORDS](#) 30
- *PSECTOR\_PAGE\_WORDS number of 16 byte words available in page A page is 512 bytes in size.*
- #define [PSECTOR\\_PAGE0\\_MAGIC](#) 0xc51d8ca9
- *PSECTOR\_PAGE0\_MAGIC magic word to identify PAGE0 page in header.*
- #define [PSECTOR\\_PFLASH\\_MAGIC](#) 0xa7b4353d
- *PSECTOR\_PFLASH\_MAGIC magic word to identify PFLASH page in header.*
- #define [IMG\\_DIRECTORY\\_MAX\\_SIZE](#) 8
- *IMG\_DIRECTORY\_MAX\_SIZE max number of entries in image directory Concerns Secondary Stage Bootloader only.*
- #define [CERTIFICATE\\_MARKER](#) (0xCE27CE27)
- *CERTIFICATE\_MARKER magic value identifying certificate.*

## Typedefs

- typedef uint32\_t [ErrorCode\\_t](#)  
*enum defined in error.h*
- typedef uint32\_t(\* [IMAGE\\_VERIFY\\_T](#))([IMAGE\\_DATA\\_T](#) \*list\_head)  
*IMAGE\_VERIFY\_T function pointer : verification function e.g.*
- typedef [ISP\\_STATUS\\_T](#)(\* [ISP\\_EXTENSION\\_T](#))([ISP\\_STATE\\_T](#) \*state, [teFlashProgCommand](#) request, uint8\_t \*in\_data, uint16\_t in\_len, [teFlashProgCommand](#) \*response, uint8\_t \*out\_data, uint16\_t \*out\_len)  
*ISP\_EXTENSION\_T ISP extension function pointer prototype.*

## Enumerations

- enum [AES\\_MODE\\_T](#) { , [AES\\_MODE\\_UNUSED](#) = 0x7FFFFFFF }
- *AES setup modes.*
- enum [AES\\_KEY\\_SIZE\\_T](#) {  
[AES\\_KEY\\_128BITS](#) = 0,  
[AES\\_KEY\\_192BITS](#),  
[AES\\_KEY\\_256BITS](#),  
[AES\\_FVAL](#) = 0x7FFFFFFF }
- *Size of the AES key.*
- enum [teFlashProgCommand](#) { ,

## Overview

```
TYPE_SET_RESET_REQUEST = 20,
TYPE_SET_RESET_RESPONSE ,
TYPE_FP_RUN_REQUEST,
TYPE_FP_RUN_RESPONSE ,
TYPE_FL_SET_BAUD_REQUEST,
TYPE_FL_SET_BAUD_RESPONSE ,
TYPE_REG_READ_REQUEST,
TYPE_REG_READ_RESPONSE,
TYPE_REG_WRITE_REQUEST,
TYPE_REG_WRITE_RESPONSE,
TYPE_GET_CHIP_ID_REQUEST,
TYPE_GET_CHIP_ID_RESPONSE,
TYPE_GET_FUSE_SECURED_REQUEST,
TYPE_GET_FUSE_SECURED_RESPONSE,
TYPE_MEM_OPEN_REQUEST = 0x40,
TYPE_MEM_OPEN_RESPONSE,
TYPE_MEM_ERASE_REQUEST,
TYPE_MEM_ERASE_RESPONSE,
TYPE_MEM_BLANK_CHECK_REQUEST,
TYPE_MEM_BLANK_CHECK_RESPONSE,
TYPE_MEM_READ_REQUEST,
TYPE_MEM_READ_RESPONSE,
TYPE_MEM_WRITE_REQUEST,
TYPE_MEM_WRITE_RESPONSE,
TYPE_MEM_CLOSE_REQUEST,
TYPE_MEM_CLOSE_RESPONSE,
TYPE_MEM_GET_INFO_REQUEST,
TYPE_MEM_GET_INFO_RESPONSE,
TYPE_UNLOCK_ISP_REQUEST,
TYPE_UNLOCK_ISP_RESPONSE,
TYPE_USE_CERTIFICATE_REQUEST,
TYPE_USE_CERTIFICATE_RESPONSE,
TYPE_START_ENCRYPTION_REQUEST,
TYPE_START_ENCRYPTION_RESPONSE }
```

*ISP Message types Only a subset of message types below is supported.*

- enum `ISP_STATUS_T` {

```

ISP_OK,
NOT_SUPPORTED = -1,
WRITE_FAIL = -2,
INVALID_RESPONSE = -3,
CRC_ERROR = -4,
ASSERT_FAIL = -5,
USER_INTERRUPT = -6,
READ_FAIL = -7,
TST_ERR = -8,
ISP_NOT_AUTHORISED = -9,
NO_RESPONSE = -10,
ISP_MEM_INVALID = -11,
ISP_MEM_NOT_SUPPORTED = -12,
ISP_MEM_NO_ACCESS = -13,
ISP_MEM_OUT_OF_RANGE = -14,
ISP_MEM_TOO_LONG = -15,
ISP_MEM_BAD_STATE = -16,
ISP_MEM_INVALID_MODE = -17 }

```

*ISP Status types.*

- enum `ISP_MEMORY_TYPE_E` { , `ISP_MEM_SPIFI` }
- enum `MpuRegion_t` {  
`MPU_REGION_0`,  
`MPU_REGION_1`,  
`MPU_REGION_2`,  
`MPU_REGION_3`,  
`MPU_REGION_4`,  
`MPU_REGION_5`,  
`MPU_REGION_6`,  
`MPU_REGION_7` }

*enum `MpuRegion_t` index of ARM CM4 MPU regions The MPU can describe up to 8 region rules.*

- enum `psector_partition_id_t` {  
`PSECTOR_PAGE0_PART`,  
`PSECTOR_PFLASH_PART` }

*psector\_partition\_id\_t describes the 2 partitions of psectors.*

- enum `psector_page_state_t` {  
`PAGE_STATE_BLANK`,  
`PAGE_STATE_ERROR`,  
`PAGE_STATE_DEGRADED`,  
`PAGE_STATE_OK` }

*psector\_page\_state\_t describes the possible states of the psector partitions.*

- enum `psector_write_status_t` {

## Overview

```
WRITE_OK = 0x0,
WRITE_ERROR_BAD_MAGIC,
WRITE_ERROR_INVALID_PAGE_NUMBER,
WRITE_ERROR_BAD_VERSION,
WRITE_ERROR_BAD_CHECKSUM,
WRITE_ERROR_INCORRECT_UPDATE_MODE,
WRITE_ERROR_UPDATE_INVALID,
WRITE_ERROR_PAGE_ERROR }
```

*psector\_write\_status\_t status code of writes to update page.*

- enum `AuthMode_t` {  
    AUTH\_NONE = 0,  
    AUTH\_ON\_FW\_UPDATE = 1,  
    AUTH\_ALWAYS = 2 }

*AuthMode\_t authentication options.*

## Functions

- static `ErrorCode_t aesInit` (void)  
*Initialize the AES.*
- static void `aesWriteByte` (uint32\_t offset, uint8\_t val8)  
*AES control function, byte write (useful for writing configuration register)*
- static void `aesWrite` (uint32\_t offset, uint32\_t val32)  
*AES control function, word write.*
- static void `aesRead` (uint32\_t offset, uint32\_t \*pVal32)  
*AES control function, word read.*
- static void `aesWriteBlock` (uint32\_t offset, uint32\_t \*pVal32, uint32\_t numBytes)  
*AES control function, block write (used for multi-register block writes)*
- static void `aesReadBlock` (uint32\_t offset, uint32\_t \*pVal32, uint32\_t numBytes)  
*AES control function, block read (used for multi-register block read)*
- static `ErrorCode_t aesMode` (`AES_MODE_T` modeVal, uint32\_t flags)  
*Sets up the AES mode.*
- static `ErrorCode_t aesAbort` (int wipe)  
*Aborts optional AES operation and wipes AES engine.*
- static `ErrorCode_t aesLoadCounter` (uint32\_t counter)  
*Loads the increment that is used when in counter modes in the AES block.*
- static `ErrorCode_t aesLoadKeyFromSW` (`AES_KEY_SIZE_T` keySize, uint32\_t \*key)  
*Loads the passed (software) key into the AES block.*
- static `ErrorCode_t aesLoadIV` (uint32\_t \*pIv)  
*Loads the Initialization Vector (IV) into the AES block.*
- static `ErrorCode_t aesProcess` (uint32\_t \*pBlockIn, uint32\_t \*pBlockOut, uint32\_t numBlocks)  
*Process AES blocks (decrypt or encrypt)*
- static `ErrorCode_t aesWriteYInputGf128` (uint32\_t \*pYGf128)  
*Sets the Y input of the GF128 hash used in GCM mode.*
- static `ErrorCode_t aesReadGf128Hash` (uint32\_t \*pGf128Hash)  
*Reads the results of the GF128(Z) hash used in GCM mode.*
- static `ErrorCode_t aesReadGcmTag` (uint32\_t \*pGcmTag)  
*Reads the GCM tag.*
- static uint32\_t `aesGetDriverVersion` (void)  
*Returns the version of the AES driver in ROM.*
- static `ErrorCode_t aesIsSupported` (void)

- *Returns status of AES IP block (supported or not)*
- static uint32\_t **BOOT\_RemapAddress** (uint32\_t address)
  - *Convert logical address into physical address, based on SYSCOM MEMORYREMAP register.*
- static uint32\_t **boot\_Verify\_eScoreImageList** (IMAGE\_DATA\_T \*list\_head)
  - *Parse the image chained list and select the first valid entry.*
- static uint32\_t **BOOT\_FindImage** (uint32\_t start\_addr, uint32\_t end\_addr, uint32\_t signature, IMAGE\_VERIFY\_T verify)
  - *Search for a valid executable image between boundaries in internal flash.*
- static uint32\_t **BOOT\_GetStartPowerMode** (void)
  - *Retrieve LPMode value that has been saved previously in retained RAM bank.*
- static void **BOOT\_SetResumeStackPointer** (uint32\_t stack\_pointer)
  - *Sets the value of stack pointer to be restored on warm boot.*
- static void **ROM\_GetFlash** (uint32\_t \*address, uint32\_t \*size)
  - *Retrieve Internal flash address and size.*
- static void **ROM\_GetSRAM0** (uint32\_t \*address, uint32\_t \*size)
  - *Retrieve SRAM0 address and size.*
- static void **ROM\_GetSRAM1** (uint32\_t \*address, uint32\_t \*size)
  - *Retrieve SRAM1 address and size.*
- static int **ISP\_Entry** (ISP\_EXTENSION\_T isp\_extension)
  - *This function is invoked when ISP mode is requested.*
- static void **Chip\_LOWPOWER\_SetUpLowPowerModeWakeUpTimer** (LPC\_LOWPOWER\_T \*p\_lowpower\_cfg)
  - *Configure Wake or RTC timers. used for testing only.*
- static int **Chip\_LOWPOWER\_SetSystemFrequency** (uint32\_t frequency)
  - *Configure CPU and System Bus clock frequency.*
- static int **Chip\_LOWPOWER\_SetMemoryLowPowerMode** (uint32\_t p\_sram\_instance, uint32\_t p\_sram\_lp\_mode)
  - *Configure Memory instances Low Power Mode.*
- static void **Chip\_LOWPOWER\_GetSystemVoltages** (LPC\_LOWPOWER\_LDOVOLTAGE\_T \*p\_ldo\_voltage)
  - *Get System Voltages.*
- static void **Chip\_LOWPOWER\_SetSystemVoltages** (LPC\_LOWPOWER\_LDOVOLTAGE\_T \*p\_ldo\_voltage)
  - *Configure System Voltages.*
- static void **Chip\_LOWPOWER\_SetLowPowerMode** (LPC\_LOWPOWER\_T \*p\_lowpower\_cfg)
  - *Configure and enters in low power mode.*
- static void **Chip\_LOWPOWER\_ChipSoftwareReset** (void)
  - *Perform a Full chip reset using Software reset bit in PMC.*
- static void **Chip\_LOWPOWER\_ArmSoftwareReset** (void)
  - *Perform a digital System reset.*
- static int **MPU\_pSectorGrantAccessRights** (uint32\_t addr, size\_t sz, MPU\_reg\_settings\_t \*save\_rule)
  - *This function is used to grant access to the pSector region.*
- static int **MPU\_pSectorWithdrawAccessRights** (MPU\_reg\_settings\_t \*save\_rule)
  - *This function is used to withdraw access to the pSector region.*
- static void **MPU\_GetCurrentSettings** (MPU\_Settings\_t \*settings)
  - *This function is used to read the MPU settings into a RAM structure.*
- static int **MPU\_AllocateRegionDesc** (void)
  - *This function is used to select the first available rule.*
- static uint32\_t **pmc\_reset\_get\_cause** (void)

## Overview

- Get the cause of the reset.*
- static void [pmc\\_reset\\_clear\\_cause](#) (uint32\_t mask)  
*Clear the cause of the reset.*
- static [psector\\_write\\_status\\_t](#) [psector\\_WriteUpdatePage](#) ([psector\\_partition\\_id\\_t](#) part\_index, [psector\\_page\\_t](#) \*page)  
*This function is used to validate a page content and write it to the update page.*
- static void [psector\\_EraseUpdate](#) (void)  
*This function is used to validate a page content and write it to the update page.*
- static [psector\\_page\\_state\\_t](#) [psector\\_ReadData](#) ([psector\\_partition\\_id\\_t](#) part\_index, int page\_number, uint32\_t offset, uint32\_t size, void \*data)  
*This function is used to read data from a psector partition.*
- static uint32\_t [psector\\_CalculateChecksum](#) ([psector\\_page\\_t](#) \*psector\_page)  
*This function is used to calculate a page checksum.*
- static uint64\_t [psector\\_Read\\_CustomerId](#) (void)  
*This function returns the CustomerId field.*
- static int [psector\\_Read\\_RomPatchInfo](#) (uint32\_t \*patch\_region\_sz, uint32\_t \*patch\_region\_addr, uint32\_t \*patch\_checksum, uint32\_t \*patch\_checksum\_valid)  
*This function returns the ROM patch information read from the PFLASH.*
- static uint16\_t [psector\\_Read\\_ImgAuthLevel](#) (void)  
*This function returns the image authentication level from the PFLASH.*
- static uint32\_t [psector\\_Read\\_AppSearchGranularity](#) (void)  
*This function returns the app search granularity value from the PFLASH.*
- static uint32\_t [psector\\_Read\\_QspiAppSearchGranularity](#) (void)  
*This function returns the Qspi app search granularity value from the PFLASH.*
- static uint64\_t [psector\\_Read\\_DeviceId](#) (void)  
*This function returns the DeviceId value from the PFLASH.*
- static int [psector\\_Read\\_UnlockKey](#) (int \*valid, uint8\_t key[256], bool raw)  
*This function returns the unlock key value from the PFLASH.*
- static int [psector\\_Read\\_ISP\\_protocol\\_key](#) (uint8\_t key[16])  
*This function returns the ISP protocol AES key from PFLASH.*
- static uint64\_t [psector\\_ReadIeee802\\_15\\_4\\_MacId1](#) (void)  
*This function returns the IEEE-802.15.4 Mac address first instance from PFLASH.*
- static uint64\_t [psector\\_ReadIeee802\\_15\\_4\\_MacId2](#) (void)  
*This function returns the IEEE-802.15.4 Mac address second instance from PFLASH.*
- static uint64\_t [psector\\_Read\\_MinDeviceId](#) (void)  
*This function returns the Min Device id from PFLASH.*
- static uint64\_t [psector\\_Read\\_MaxDeviceId](#) (void)  
*This function returns the Max Device id from PFLASH.*
- static uint32\_t [psector\\_Read\\_MinVersion](#) (void)  
*This function returns the Min Version from PAGE0.*
- static [psector\\_write\\_status\\_t](#) [psector\\_SetEscoreImageData](#) (uint32\_t image\_addr, uint32\_t min\_version)  
*This function is used to set the selected image address and MinVersion into PAGE0.*
- static [psector\\_page\\_state\\_t](#) [psector\\_ReadEscoreImageData](#) (uint32\_t \*image\_addr, uint32\_t \*min\_version)  
*This function returns the image address and min version value from PAGE0.*
- static int [psector\\_Read\\_ImagePubKey](#) (int \*valid, uint8\_t key[256], bool raw)  
*This function returns the unlock key value from PAGE0.*
- static uint32\_t [secure\\_VerifySignature](#) (uint8\_t \*hash, const uint8\_t \*signature, const uint32\_t \*key)  
*This function performs an RSA 2048 signature verification.*
- static uint32\_t [secure\\_VerifyBlock](#) (uint8\_t \*start, uint32\_t length, const uint32\_t \*key, const uint8\_t

\_t \*signature)

*This function performs an RSA 2048 signature verification over specified data block.*

- static uint32\_t [secure\\_VerifyCertificate](#) (const [IMAGE\\_CERT\\_T](#) \*certificate, const uint32\_t \*key, const uint8\_t \*cert\_signature)

*This function performs an RSA 2048 signature verification.*

- static uint32\_t [secure\\_VerifyImage](#) (uint32\_t image\_addr, const [IMAGE\\_CERT\\_T](#) \*root\_cert)
- This function verifies image authenticity.*

## Variables

- uint32\_t [IMAGE\\_DATA\\_T::version](#)  
*version number found in image*
- uint32\_t [IMAGE\\_DATA\\_T::address](#)  
*start address of image*
- struct \_image\_data\_t \* [IMAGE\\_DATA\\_T::next](#)  
*pointer on next [IMAGE\\_DATA\\_T](#) in list*
- uint32\_t [ISP\\_MEM\\_INFO\\_T::base\\_address](#)  
*base address of memory bank*
- uint32\_t [ISP\\_MEM\\_INFO\\_T::length](#)  
*total size*
- uint32\_t [ISP\\_MEM\\_INFO\\_T::block\\_size](#)  
*block size : flash page size*
- uint16\_t [ISP\\_MEM\\_INFO\\_T::flags](#)  
*unused*
- [ISP\\_MEMORY\\_TYPE\\_E](#) [ISP\\_MEM\\_INFO\\_T::type](#)  
*memory type : note that EFUSE bank is not a memory as such - SPIFI is unimplemented*
- uint8\_t [ISP\\_MEM\\_INFO\\_T::access](#)  
*bitfield of access rights:*
- uint8\_t [ISP\\_MEM\\_INFO\\_T::auth\\_access](#)  
*similar to access for authenticated commands*
- [ISP\\_MEM\\_FUNC\\_T](#) \* [ISP\\_MEM\\_INFO\\_T::func](#)  
*set of function pointers of this memory type see @ [ISP\\_MEM\\_FUNC\\_T](#)*
- const char \* [ISP\\_MEM\\_INFO\\_T::name](#)  
*name of memory bank*
- uint32\_t [ISP\\_ENC\\_STATE\\_T::mode](#)  
*0: none - 1: AES CTR*
- uint32\_t [ISP\\_ENC\\_STATE\\_T::start](#)  
*start address of cipher/decipher operation*
- uint32\_t [ISP\\_ENC\\_STATE\\_T::end](#)  
*end address of cipher/decipher operation*
- uint32\_t [ISP\\_ENC\\_STATE\\_T::iv](#) [4]  
*Initialization vector IV : 16 bytes.*
- uint32\_t [ISP\\_ENC\\_STATE\\_T::key](#) [8]  
*AES Key - key[4..7] unused.*
- [ISP\\_GET\\_MEMORY\\_T](#) [ISP\\_STATE\\_T::get\\_memory](#)  
*Function pointer to get\_memory.*
- [ISP\\_EXTENSION\\_T](#) [ISP\\_STATE\\_T::extension](#)  
*Function pointer to extension.*
- uint32\_t \* [ISP\\_STATE\\_T::buffer](#)  
*buffer holding command (in stack)*
- [ISP\\_ENC\\_STATE\\_T](#) [ISP\\_STATE\\_T::enc\\_state](#)



## Overview

- Embedded ciphering structure see @ [ISP\\_ENC\\_STATE\\_T](#).*
- [IMAGE\\_CERT\\_T ISP\\_STATE\\_T::certificate](#)  
*Certificate used to authenticate ISP commands it is composed of the customer identifier and the unlock public key found in PFLASH.*
- [uint8\\_t ISP\\_STATE\\_T::stored\\_hash](#) [32]  
*SHA=256 hash storage.*
- [uint8\\_t ISP\\_STATE\\_T::mode](#)  
*mode 0x00: inactive*
- [uint8\\_t ISP\\_STATE\\_T::isp\\_level](#)  
*ISP level as restrained by EFUSE configuration and PFLASH parameter.*
- [uint16\\_t ISP\\_STATE\\_T::buffer\\_size](#)  
*size of buffer : normally 1024*
- [uint8\\_t ISP\\_STATE\\_T::unlock\\_disable](#)  
*unlock forbidden by EFUSE*
- [uint8\\_t ISP\\_STATE\\_T::SWD\\_disable](#)  
*SWD Debug interface disabled.*
- [uint32\\_t MPU\\_Settings\\_t::ctrl](#)  
*MPU Ctrl register.*
- [uint32\\_t MPU\\_Settings\\_t::rbar](#) [8]  
*MPU RBAR array for the 8 rules.*
- [uint32\\_t MPU\\_Settings\\_t::rshr](#) [8]  
*MPU RASR array for the 8 rules.*
- [uint32\\_t image\\_directory\\_entry\\_t::img\\_base\\_addr](#)  
*image start address in internal Flash or QSPI flash*
- [uint16\\_t image\\_directory\\_entry\\_t::img\\_nb\\_pages](#)  
*image number of 512 byte pages*
- [uint8\\_t image\\_directory\\_entry\\_t::flags](#)  
*IMG\_FLAG\_BOOTABLE : bit 0, other TBD.*
- [uint8\\_t image\\_directory\\_entry\\_t::img\\_type](#)  
*image type*
- [uint32\\_t psector\\_header\\_t::checksum](#)  
*page checksum*
- [uint32\\_t psector\\_header\\_t::magic](#)  
*magic: PSECTOR\_PAGE0\_MAGIC or PSECTOR\_PFLASH\_MAGIC*
- [uint16\\_t psector\\_header\\_t::page\\_number](#)  
*should be 0 because both partitions contain a single page*
- struct {  
    [psector\\_page\\_data\\_t::page0\\_v2](#)  
}
- Deprecated form kept for backward compatibility.*
- [uint32\\_t psector\\_page\\_data\\_t::SelectedImageAddress](#)  
*Address of image to be loaded by boot ROM offset 0x20.*
- [uint32\\_t psector\\_page\\_data\\_t::preferred\\_app\\_index](#)  
*for use with SSBL: index of application to select from image directory value 0..8 offset 0x24*
- [image\\_directory\\_entry\\_t psector\\_page\\_data\\_t::ota\\_entry](#)  
*New image written by OTA : SSBL to check validity and authentication offset 0x28.*
- [uint32\\_t psector\\_page\\_data\\_t::MinVersion](#)  
*Minimum version accepted : application's version number must be greater than this one to be accepted.*
- [uint32\\_t psector\\_page\\_data\\_t::img\\_pk\\_valid](#)  
*Image public key valid offset 0x34.*
- [uint32\\_t psector\\_page\\_data\\_t::flash\\_audit\\_done](#)



- Flash audit done: already sought for wrongly initialized pages offset 0x38.
- uint32\_t psector\_page\_data\_t::RESERVED1  
padding reserved word
- uint8\_t psector\_page\_data\_t::image\_pubkey [256]  
RSA Public Key to be used to verify authenticity offset 0x40.
- uint8\_t psector\_page\_data\_t::zigbee\_install\_code [36]  
Zigbee install code offset 0x140.
- uint32\_t psector\_page\_data\_t::RESERVED3 [3]  
padding reserved wordes
- uint8\_t psector\_page\_data\_t::zigbee\_password [16]  
Zigbee password offset 0x170.
- image\_directory\_entry\_t psector\_page\_data\_t::img\_directory [IMG\_DIRECTORY\_MAX\_SIZE]  
< Image directory entries array, used by OTA process to locate images and/or blobs offset 0x180
- uint32\_t psector\_page\_data\_t::rom\_patch\_region\_addr  
ROM patch entry point address.
- uint32\_t psector\_page\_data\_t::rom\_patch\_checksum\_valid  
ROM patch checksum valid: 0 means invalid Any other value means valid.
- uint32\_t psector\_page\_data\_t::ISP\_access\_level  
ISP access level: 0 means full access, unsecure 0x01010101 means full access, secure 0x02020202 means write only, unsecure 0x03030303 means write only, secure 0x04040404 means locked Any other value means disabled.
- uint16\_t psector\_page\_data\_t::application\_flash\_sz  
Application flash size, in kilobytes.
- uint16\_t psector\_page\_data\_t::image\_authentication\_level  
Image authentication level: 0 means check only header validity 1 means check signature of whole image if image has changed 2 means check signature of whole image on every cold start.
- uint16\_t psector\_page\_data\_t::unlock\_key\_valid  
0: unlock key is not valid, >= 1: is present
- uint16\_t psector\_page\_data\_t::ram1\_bank\_sz  
RAM bank 1 size, in kilobytes.
- uint32\_t psector\_page\_data\_t::app\_search\_granularity  
Application search granularity (increment), in bytes.
- uint8\_t psector\_page\_data\_t::ISP\_protocol\_key [16]  
ISP protocol key: key used to encrypt messages over ISP UART with secure access level.
- uint64\_t psector\_page\_data\_t::ieee\_mac\_id1  
IEEE\_MAC\_ID\_1 (Used to over-ride MAC ID\_1 in N-2 page)
- uint64\_t psector\_page\_data\_t::ieee\_mac\_id2  
IEEE\_MAC\_ID\_2 if second MAC iID is required.
- uint64\_t psector\_page\_data\_t::ble\_mac\_id  
BLE device address : only 6 LSB bytes are significant.
- uint8\_t psector\_page\_data\_t::reserved2 [104]  
Reserved for future use.
- uint64\_t psector\_page\_data\_t::customer\_id  
Customer ID, used for secure handshake.
- uint64\_t psector\_page\_data\_t::min\_device\_id  
Min Device ID, used for secure handshake - Certificate compatibility.
- uint64\_t psector\_page\_data\_t::device\_id  
Device ID, used for secure handshake.
- uint64\_t psector\_page\_data\_t::max\_device\_id  
Max Device ID, used for secure handshake - Certificate compatibility.
- uint8\_t psector\_page\_data\_t::unlock\_key [256]

## Data Structure Documentation

- `uint32_t IMAGE_CERT_T::certificate_marker`  
*2048-bit public key for secure handshake (equivalent to 'unlock' key).*
- `uint32_t IMAGE_CERT_T::certificate_id`  
*Certificate marker: magic see @ CERTIFICATE\_MARKER.*
- `uint32_t IMAGE_CERT_T::usage_flags`  
*Certificate id.*
- `uint64_t IMAGE_CERT_T::customer_id`  
*Usage flags: mostly used in the unlocking procedure.*
- `uint64_t IMAGE_CERT_T::min_device_id`  
*Customer Id: customer chosen identifier.*
- `uint64_t IMAGE_CERT_T::max_device_id`  
*Min device id: min device version from which certificate applies.*
- `uint32_t IMAGE_CERT_T::public_key` [SIGNATURE\_LEN/4]  
*Max device id: max device version up to which certificate applies.*
- `IMAGE_CERT_T ImageAuthTrailer_t::certificate`  
*RSA-2048 public key.*
- `uint8_t ImageAuthTrailer_t::cert_signature` [SIGNATURE\_LEN]  
*The certificate see @ IMAGE\_CERT\_T.*
- `uint8_t ImageAuthTrailer_t::img_signature` [SIGNATURE\_LEN]  
*The signature of the certificate.*
- `uint8_t ImageAuthTrailer_t::img_signature` [SIGNATURE\_LEN]  
*The image signature.*

## Driver version

- `#define FSL_JN_ROMAPI_DRIVER_VERSION (MAKE_VERSION(2, 0, 0))`  
*JN\_ROMAPI driver version 2.0.0.*

## 37.2 Data Structure Documentation

### 37.2.1 struct IMAGE\_DATA\_T

#### Data Fields

- `uint32_t version`  
*version number found in image*
- `uint32_t address`  
*start address of image*
- `struct _image_data_t * next`  
*pointer on next IMAGE\_DATA\_T in list*

### 37.2.2 struct ISP\_MEM\_FUNC\_T

### 37.2.3 struct ISP\_MEM\_INFO\_T

#### Data Fields

- `uint32_t base_address`

- *base address of memory bank*  
uint32\_t [length](#)
- *total size*  
uint32\_t [block\\_size](#)
- *block size : flash page size*  
uint16\_t [flags](#)
- *unused*  
[ISP\\_MEMORY\\_TYPE\\_E](#) type
- *memory type : note that EFUSE bank is not a memory as such - SPIFI is unimplemented*  
uint8\_t [access](#)
- *bitfield of access rights:*  
uint8\_t [auth\\_access](#)
- *similar to access for authenticated commands*  
[ISP\\_MEM\\_FUNC\\_T](#) \* [func](#)
- *set of function pointers of this memory type see @ [ISP\\_MEM\\_FUNC\\_T](#)*  
const char \* [name](#)
- *name of memory bank*

### 37.2.4 struct ISP\_ENC\_STATE\_T

#### Data Fields

- uint32\_t [mode](#)  
*0: none - 1: AES CTR*
- uint32\_t [start](#)  
*start address of cipher/decipher operation*
- uint32\_t [end](#)  
*end address of cipher/decipher operation*
- uint32\_t [iv](#) [4]  
*Initialization vector IV : 16 bytes.*
- uint32\_t [key](#) [8]  
*AES Key - key[4..7] unused.*

### 37.2.5 struct ISP\_STATE\_T

Note: this context is held in RAM is the stack so is lost after ISP\_Entry is exited.

#### Data Fields

- [ISP\\_GET\\_MEMORY\\_T](#) [get\\_memory](#)  
*Function pointer to get\_memory.*
- [ISP\\_EXTENSION\\_T](#) [extension](#)  
*Function pointer to extension.*
- uint32\_t \* [buffer](#)  
*buffer holding command (in stack)*
- [ISP\\_ENC\\_STATE\\_T](#) [enc\\_state](#)

- *Embedded ciphering structure see @ [ISP\\_ENC\\_STATE\\_T](#).*
- [IMAGE\\_CERT\\_T](#) certificate  
*Certificate used to authenticate ISP commands it is composed of the customer identifier and the unlock public key found in PFLASH.*
- [uint8\\_t stored\\_hash](#) [32]  
*SHA=256 hash storage.*
- [uint8\\_t mode](#)  
*mode 0x00: inactive*
- [uint8\\_t isp\\_level](#)  
*ISP level as restrained by EFUSE configuration and PFLASH parameter.*
- [uint16\\_t buffer\\_size](#)  
*size of buffer : normally 1024*
- [uint8\\_t unlock\\_disable](#)  
*unlock forbidden by EFUSE*
- [uint8\\_t SWD\\_disable](#)  
*SWD Debug interface disabled.*

### 37.2.6 struct LPC\_LOWPOWER\_T

#### Data Fields

- [uint32\\_t CFG](#)  
*Low Power Mode Configuration, and miscallenous options.*
- [uint32\\_t PMUPWDN](#)  
*Analog Power Domains (analog components in Power Management Unit) Low Power Modes.*
- [uint32\\_t DIGPWDN](#)  
*Digital Power Domains Low Power Modes.*
- [uint32\\_t VOLTAGE](#)  
*LDO Voltage control in Low Power Modes.*
- [uint32\\_t WAKEUPSRCINT0](#)  
*Wake up sources Interrupt control.*
- [uint32\\_t WAKEUPSRCINT1](#)  
*Wake up sources Interrupt control.*
- [uint32\\_t SLEEPPOSTPONE](#)  
*Interrupt that can postpone power down modes in case an interrupt is pending when the processor request deepsleep.*
- [uint32\\_t WAKEUIOSRC](#)  
*Wake up I/O sources.*
- [uint32\\_t GPIOLATCH](#)  
*I/Os which outputs level must be kept (in Power Down mode)*
- [uint32\\_t TIMERCFG](#)  
*Wake up timers configuration.*
- [uint32\\_t TIMERBLECFG](#)  
*BLE wake up timer configuration (OSC\_EN and RADIO\_EN)*
- [uint32\\_t TIMERCOUNTLSB](#)  
*Wake up Timer LSB.*
- [uint32\\_t TIMERCOUNTMSB](#)  
*Wake up Timer MSB.*
- [uint32\\_t TIMER2NDCOUNTLSB](#)

- *Second Wake up Timer LSB.*  
uint32\_t [TIMER2NDCOUNTMSB](#)  
*Second Wake up Timer MSB.*

### 37.2.7 struct LPC\_LOWPOWER\_LDOVOLTAGE\_T

#### Data Fields

- uint8\_t [LDOPMU](#)  
*Always-ON domain LDO voltage configuration.*
- uint8\_t [LDOPMUBOOST](#)  
*Always-ON domain LDO Boost voltage configuration.*
- uint8\_t [LDOMEM](#)  
*Memories LDO voltage configuration.*
- uint8\_t [LDOMEMBOOST](#)  
*Memories LDO Boost voltage configuration.*
- uint8\_t [LDOCORE](#)  
*Core Logic domain LDO voltage configuration.*
- uint8\_t [LDOFLASHNV](#)  
*Flash NV domain LDO voltage configuration.*
- uint8\_t [LDOFLASHCORE](#)  
*Flash Core domain LDO voltage configuration.*
- uint8\_t [LDOADC](#)  
*General Purpose ADC LDO voltage configuration.*
- uint8\_t [LDOPMUBOOST\\_ENABLE](#)  
*Force Boost activation on LDOPMU.*

### 37.2.8 struct MPU\_Settings\_t

#### Data Fields

- uint32\_t [ctrl](#)  
*MPU Ctrl register.*
- uint32\_t [rbar](#) [8]  
*MPU RBAR array for the 8 rules.*
- uint32\_t [rasr](#) [8]  
*MPU RASR array for the 8 rules.*

### 37.2.9 struct image\_directory\_entry\_t

#### Data Fields

- uint32\_t [img\\_base\\_addr](#)  
*image start address in internal Flash or QSPI flash*
- uint16\_t [img\\_nb\\_pages](#)

## Macro Definition Documentation

- `uint8_t flags`  
*image number of 512 byte pages*
- `uint8_t img_type`  
*IMG\_FLAG\_BOOTABLE : bit 0, other TBD.*  
*image type*

### 37.2.10 struct psector\_header\_t

#### Data Fields

- `uint32_t checksum`  
*page checksum*
- `uint32_t magic`  
*magic: PSECTOR\_PAGE0\_MAGIC or PSECTOR\_PFLASH\_MAGIC*
- `uint16_t page_number`  
*should be 0 because both partitions contain a single page*

### 37.2.11 struct IMAGE\_CERT\_T

#### Data Fields

- `uint32_t certificate_marker`  
*Certificate marker: magic see @ CERTIFICATE\_MARKER.*
- `uint32_t certificate_id`  
*Certificate id.*
- `uint32_t usage_flags`  
*Usage flags: mostly used in the unlocking procedure.*
- `uint64_t customer_id`  
*Customer Id: customer chosen identifier.*
- `uint64_t min_device_id`  
*Min device id: min device version from which certificate applies.*
- `uint64_t max_device_id`  
*Max device id: max device version up to which certificate applies.*
- `uint32_t public_key` [SIGNATURE\_LEN/4]  
*RSA-2048 public key.*

## 37.3 Macro Definition Documentation

### 37.3.1 #define FSL\_JN\_ROMAPI\_DRIVER\_VERSION (MAKE\_VERSION(2, 0, 0))

### 37.3.2 #define ISP\_FLAG\_HAS\_CRC32 (1 << 0)

CRC32 Now deprecated

**37.3.3 #define LOWPOWER\_CFG\_FLASHPWDNMODE\_FLASHPWDN 0**

Only valid in DEEP SLEEP mode

**37.3.4 #define LOWPOWER\_PMUPWDN\_DCDC (1UL << 0)**

Power Down DCDC Converter

**37.3.5 #define LOWPOWER\_DIGPWDN\_FLASH (1UL << 6)**

Power Down Flash Power Domain (Flash Macro, Flash Controller and/or FLash LDOs, depending on \ \ LOWPOWER\_CFG\_FLASHPWDNMODE parameter)

**37.3.6 #define LOWPOWER\_WAKEUPSRCINT0\_SYSTEM\_IRQ (1UL << 0)**

BOD, Watchdog Timer, Flash controller, Firewall [DEEP SLEEP] BOD [POWER\_DOWN]

**37.3.7 #define LOWPOWER\_SLEEPPOSTPONE\_FORCED (1UL << 0)**

Forces postpone of power down modes in case the processor request low power mode

**37.3.8 #define LOWPOWER\_TIMERCFG\_TIMER\_WAKEUPTIMER0 0**

Zigbee Wake up Counter 0 used as wake up source

**37.3.9 #define LOWPOWER\_TIMERCFG\_2ND\_TIMER\_WAKEUPTIMER0 0**

Zigbee Wake up Counter 0 used as secondary wake up source

**37.3.10 #define PSECTOR\_PAGE\_WORDS 30**

That is 32x16 bytes. The first 32 bytes contain the page header, which leaves 30x16bytes for storage. Thence the 30.

## Enumeration Type Documentation

### 37.4 Typedef Documentation

#### 37.4.1 typedef uint32\_t(\* IMAGE\_VERIFY\_T)(IMAGE\_DATA\_T \*list\_head)

boot\_Verify\_eScoreImageList

### 37.5 Enumeration Type Documentation

#### 37.5.1 enum AES\_MODE\_T

Enumerator

**AES\_MODE\_UNUSED** Not used, but forces enum to 32-bit size.

#### 37.5.2 enum AES\_KEY\_SIZE\_T

Enumerator

**AES\_KEY\_128BITS** KEY size 128 bits.

**AES\_KEY\_192BITS** KEY size 192 bits.

**AES\_KEY\_256BITS** KEY size 256 bits.

**AES\_FVAL** Not used, but forces enum to 32-bit size and unsigned.

#### 37.5.3 enum teFlashProgCommand

Enumerator

**TYPE\_SET\_RESET\_REQUEST** ISP Set Reset request.

**TYPE\_SET\_RESET\_RESPONSE** ISP Set Reset response.

**TYPE\_FP\_RUN\_REQUEST** ISP FP Run request : jump to address if ISP access level and authentication allow it.

**TYPE\_FP\_RUN\_RESPONSE** ISP FP Run response.

**TYPE\_FL\_SET\_BAUD\_REQUEST** ISP FL Set Baud request : set UART speed.

**TYPE\_FL\_SET\_BAUD\_RESPONSE** ISP FL Set Baud response.

**TYPE\_REG\_READ\_REQUEST** not implemented

**TYPE\_REG\_READ\_RESPONSE** not implemented

**TYPE\_REG\_WRITE\_REQUEST** not implemented

**TYPE\_REG\_WRITE\_RESPONSE** not implemented

**TYPE\_GET\_CHIP\_ID\_REQUEST** ISP chip id request.

**TYPE\_GET\_CHIP\_ID\_RESPONSE** ISP chip id response.

**TYPE\_GET\_FUSE\_SECURED\_REQUEST** not implemented

**TYPE\_GET\_FUSE\_SECURED\_RESPONSE** not implemented

**TYPE\_MEM\_OPEN\_REQUEST** ISP memory open request.

**TYPE\_MEM\_OPEN\_RESPONSE** ISP memory open response.



***TYPE\_MEM\_ERASE\_REQUEST*** ISP memory erase request, applies to internal flash only.

***TYPE\_MEM\_ERASE\_RESPONSE*** ISP memory erase response.

***TYPE\_MEM\_BLANK\_CHECK\_REQUEST*** ISP memory blank check request, applies to internal flash only.

***TYPE\_MEM\_BLANK\_CHECK\_RESPONSE*** ISP memory blank check response to request.

***TYPE\_MEM\_READ\_REQUEST*** ISP memory read request, applies to all memory types.

***TYPE\_MEM\_READ\_RESPONSE*** ISP memory read response.

***TYPE\_MEM\_WRITE\_REQUEST*** ISP memory write request, applies to all memory types except EFUSE.

***TYPE\_MEM\_WRITE\_RESPONSE*** ISP memory read response.

***TYPE\_MEM\_CLOSE\_REQUEST*** ISP memory close request.

***TYPE\_MEM\_CLOSE\_RESPONSE*** ISP memory close response.

***TYPE\_MEM\_GET\_INFO\_REQUEST*** ISP memory get information of memory geometry and accessibility.

***TYPE\_MEM\_GET\_INFO\_RESPONSE*** ISP memory get information response.

***TYPE\_UNLOCK\_ISP\_REQUEST*** ISP unlock request: reset a locked device to its pristine state.

***TYPE\_UNLOCK\_ISP\_RESPONSE*** ISP unlock response.

***TYPE\_USE\_CERTIFICATE\_REQUEST*** ISP Use Certificate request.

***TYPE\_USE\_CERTIFICATE\_RESPONSE*** ISP Use Certificate response.

***TYPE\_START\_ENCRYPTION\_REQUEST*** ISP Start Encryption request.

***TYPE\_START\_ENCRYPTION\_RESPONSE*** ISP Start Encryption response.

### 37.5.4 enum ISP\_STATUS\_T

Enumerator

***ISP\_OK*** ISP operation successful.

***NOT\_SUPPORTED*** ISP operation not supported.

***WRITE\_FAIL*** ISP write failure when writing to FLASH, PSECT, or PFLASH.

***INVALID\_RESPONSE*** ISP invalid response : not used.

***CRC\_ERROR*** ISP command received CRC incorrect.

***ASSERT\_FAIL*** ISP received too long a message.

***USER\_INTERRUPT*** ISP User aborted operation: not used.

***READ\_FAIL*** ISP Flash blank check error or Flash excessive ECC errors.

***TST\_ERR*** not used

***ISP\_NOT\_AUTHORIZED*** ISP order authentication failure.

***NO\_RESPONSE*** not used

***ISP\_MEM\_INVALID*** ISP message malformed : addressed to non-existent memory.

***ISP\_MEM\_NOT\_SUPPORTED*** ISP order not supported for memory.

***ISP\_MEM\_NO\_ACCESS*** ISP access level insufficient.

***ISP\_MEM\_OUT\_OF\_RANGE*** ISP order addressing memory outside the intended range.

***ISP\_MEM\_TOO\_LONG*** ISP buffer insufficient to read requested amount of memory.

***ISP\_MEM\_BAD\_STATE*** Memory in a state that cannot support operation e.g. opening an errored PSECT or PFLASH, closing a memory that was not opened

## Enumeration Type Documentation

***ISP\_MEM\_INVALID\_MODE*** ISP order is malformed : mode incorrect.

### 37.5.5 enum ISP\_MEMORY\_TYPE\_E

Enumerator

***ISP\_MEM\_SPIFI*** Unused SPIFI not handled by ISP.

### 37.5.6 enum MpuRegion\_t

This function is used to set access rights to a region.

Note that a higher order rule prevails over the lower ones. The boot ROM makes use of upper order rules : 5 .. 7. Rule 0 is a 'background' rule that opens the whole memory plane.

Parameters

|                      |                                                                                                  |
|----------------------|--------------------------------------------------------------------------------------------------|
| <i>region_id,:</i>   | 0 .. 7 see                                                                                       |
| <i>addr,:</i>        | address of region                                                                                |
| <i>sz,:</i>          | region size                                                                                      |
| <i>rw_x_rights,:</i> | bit field <a href="#">RD_RIGHT(0)</a> - <a href="#">WR_RIGHT(1)</a> - <a href="#">X_RIGHT(2)</a> |
| <i>save_rule,:</i>   | save a copy of previous rule                                                                     |

Returns

-1 if failure, if succesful return the size of the region.

Called after previous call to see

Parameters

|                    |                                                                                                                          |
|--------------------|--------------------------------------------------------------------------------------------------------------------------|
| <i>region_id,:</i> | 0 .. 7 see                                                                                                               |
| <i>save_rule,:</i> | saved copy of previous rule to be restored. if this parameter is NULL, RBAR and RASR of the given region_id are cleared. |

Returns

-1 if failure, if succesful return the size of the region.

Enumerator

***MPU\_REGION\_0*** Boot Reserved: background rule

***MPU\_REGION\_1*** General purpose rule  
***MPU\_REGION\_2*** General purpose rule  
***MPU\_REGION\_3*** General purpose rule  
***MPU\_REGION\_4*** General purpose rule  
***MPU\_REGION\_5*** Boot Reserved  
***MPU\_REGION\_6*** Boot Reserved  
***MPU\_REGION\_7*** Boot Reserved

### 37.5.7 enum psector\_partition\_id\_t

Note: PAGE0 is termed PSECT in the FlashProgrammer, whereas PFLASH remains PFLASH.

Enumerator

***PSECTOR\_PAGE0\_PART*** Page0 partition : termed PSECT by FLashProgramemr tool Image related data.  
***PSECTOR\_PFLASH\_PART*** PFLASH : Custommer configuration data.

### 37.5.8 enum psector\_page\_state\_t

Enumerator

***PAGE\_STATE\_BLANK*** Page has never been programmed or has been erased.  
***PAGE\_STATE\_ERROR*** Both subpages constituting the psector contain unrecoverable errors that ECC/parity cannot mend.  
***PAGE\_STATE\_DEGRADED*** One subpage contains unrecoverable errors or is blank.  
***PAGE\_STATE\_OK*** Both subpages are correct.

### 37.5.9 enum psector\_write\_status\_t

Enumerator

***WRITE\_OK*** Succeeded in writing page.  
***WRITE\_ERROR\_BAD\_MAGIC*** Magic word incorrect in page header.  
***WRITE\_ERROR\_INVALID\_PAGE\_NUMBER*** Invalid page number (higher than partition size)  
***WRITE\_ERROR\_BAD\_VERSION*** Invalid version number: must increment monotonically.  
***WRITE\_ERROR\_BAD\_CHECKSUM*** Invalid checksum.  
***WRITE\_ERROR\_INCORRECT\_UPDATE\_MODE*** Update mode incorrect.  
***WRITE\_ERROR\_UPDATE\_INVALID*** Update invalid.  
***WRITE\_ERROR\_PAGE\_ERROR*** Failure to program page in flash.

## Function Documentation

### 37.5.10 enum AuthMode\_t

Enumerator

*AUTH\_NONE* no authentication is performed

*AUTH\_ON\_FW\_UPDATE* authentication is performed on firmware update

*AUTH\_ALWAYS* authentication is performed at each Cold boot

## 37.6 Function Documentation

### 37.6.1 static ErrorCode\_t aesInit ( void ) [inline], [static]

Returns

LPC\_OK on success, or an error code (ERRORCODE\_T) on failure

Note

Driver does not enable AES clock, power, or perform reset peripheral (if needed).

### 37.6.2 static void aesWriteByte ( uint32\_t offset, uint8\_t val8 ) [inline], [static]

offset : Register offset in AES, 32-bit aligned value val8 : 8-bit value to write

Returns

Nothing

Note

This is an obfuscated function available from the ROM API as a 2nd level API call. An application can use it to perform byte level write access to a register. This function is not meant to be public.

### 37.6.3 static void aesWrite ( uint32\_t offset, uint32\_t val32 ) [inline], [static]

offset : Register offset in AES, 32-bit aligned value val32 : 32-bit value to write

Returns

Nothing

Note

This is an obfuscated function available from the ROM API as a 2nd level API call. An application can use it for write access to a register. This function is not meant to be public.

#### 37.6.4 **static void aesRead ( uint32\_t *offset*, uint32\_t \* *pVal32* ) [inline], [static]**

*offset* : Register offset in AES, 32-bit aligned value *pVal32* : Pointer to 32-bit area to read into

Returns

Nothing

Note

This is an obfuscated function available from the ROM API as a 2nd level API call. An application can use it for read access to a register. This function is not meant to be public.

#### 37.6.5 **static void aesWriteBlock ( uint32\_t *offset*, uint32\_t \* *pVal32*, uint32\_t *numBytes* ) [inline], [static]**

*offset* : Register offset in AES, 32-bit aligned value *pVal32* : Pointer to 32-bit array to write *numBytes* : Number of bytes to write, must be 32-bit aligned

Returns

Nothing

Note

This is an obfuscated function available from the ROM API as a 2nd level API call. An application can use it for write access to a register. This function is not meant to be public. Writes occur in 32-bit chunks.

#### 37.6.6 **static void aesReadBlock ( uint32\_t *offset*, uint32\_t \* *pVal32*, uint32\_t *numBytes* ) [inline], [static]**

*offset* : Register offset in AES, 32-bit aligned value *pVal32* : Pointer to 32-bit array to read into *numBytes* : Number of bytes to read, must be 32-bit aligned

Returns

Nothing

Note

This is an obfuscated function available from the ROM API as a 2nd level API call. An application can use it for read access to a register. This function is not meant to be public. Reads occur in 32-bit chunks. Read data is undefined if AES is not present.

**37.6.7**   **static ErrorCode\_t aesMode ( AES\_MODE\_T *modeVal*, uint32\_t *flags* )**  
          **[inline], [static]**

## Parameters

|              |                                                                                          |
|--------------|------------------------------------------------------------------------------------------|
| <i>wipe</i>  | : use true to invalidate AES key and disable cipher                                      |
| <i>flags</i> | : Applies extra flags (Or'ed in config), normally should be 0, useful for swap bits only |

## Returns

LPC\_OK on success, or an error code (ERRORCODE\_T) on failure

### 37.6.8 static ErrorCode\_t aesAbort ( int *wipe* ) [inline], [static]

## Parameters

|             |                                                     |
|-------------|-----------------------------------------------------|
| <i>wipe</i> | : use true to invalidate AES key and disable cipher |
|-------------|-----------------------------------------------------|

## Returns

LPC\_OK on success, or an error code (ERRORCODE\_T) on failure

### 37.6.9 static ErrorCode\_t aesLoadCounter ( uint32\_t *counter* ) [inline], [static]

## Parameters

|                |                                          |
|----------------|------------------------------------------|
| <i>counter</i> | : 32-bit initial increment counter value |
|----------------|------------------------------------------|

## Returns

LPC\_OK on success, or an error code (ERRORCODE\_T) on failure

### 37.6.10 static ErrorCode\_t aesLoadKeyFromSW ( AES\_KEY\_SIZE\_T *keySize*, uint32\_t\* *key* ) [inline], [static]

## Function Documentation

### Parameters

|                |                                                                                           |
|----------------|-------------------------------------------------------------------------------------------|
| <i>keySize</i> | : 0 = 128-bits, 1 = 192-bits, 2 = 256-bits, all other values are invalid (AES_KEY_SIZE_T) |
| <i>key</i>     | : Pointer to up to a 256-bit key array                                                    |

### Returns

LPC\_OK on success, or an error code (ERRORCODE\_T) on failure

### 37.6.11 static ErrorCode\_t aesLoadIV ( uint32\_t \* *pIv* ) [inline], [static]

### Parameters

|           |                                |
|-----------|--------------------------------|
| <i>iv</i> | : 32-bit initialization vector |
|-----------|--------------------------------|

### Returns

LPC\_OK on success, or an error code (ERRORCODE\_T) on failure

### 37.6.12 static ErrorCode\_t aesProcess ( uint32\_t \* *pBlockIn*, uint32\_t \* *pBlockOut*, uint32\_t *numBlocks* ) [inline], [static]

### Parameters

|                  |                                                      |
|------------------|------------------------------------------------------|
| <i>pBlockIn</i>  | : 32-bit aligned pointer to input block of data      |
| <i>pBlockOut</i> | : 32-bit aligned pointer to output block of data     |
| <i>numBlocks</i> | : Number of blocks to process, block size = 128 bits |

### Returns

LPC\_OK on success, or an error code (ERRORCODE\_T) on failure

### Note

The AES mode and key must be setup prior to calling this function. For encryption, the plain text is used as the input and encrypted text is output. For decryption, plain text is output while encrypted text is input.

### 37.6.13 static ErrorCode\_t aesWriteYInputGf128 ( uint32\_t \* *pYGf128* ) [inline], [static]



## Parameters

|                |                                          |
|----------------|------------------------------------------|
| <i>pYGf128</i> | : Y input of GF128 hash (4x32-bit words) |
|----------------|------------------------------------------|

## Returns

LPC\_OK on success, or an error code (ERRORCODE\_T) on failure

## Note

Calling this function will reset the hash logic.

### 37.6.14 static ErrorCode\_t aesReadGf128Hash ( uint32\_t \* *pGf128Hash* ) [inline], [static]

## Parameters

|                   |                                             |
|-------------------|---------------------------------------------|
| <i>pGf128Hash</i> | : Array of 4x32-bit words to read hash into |
|-------------------|---------------------------------------------|

## Returns

LPC\_OK on success, or an error code (ERRORCODE\_T) on failure

## Note

Value is undefined if AES is not present.

### 37.6.15 static ErrorCode\_t aesReadGcmTag ( uint32\_t \* *pGcmTag* ) [inline], [static]

## Parameters

|                |                                                 |
|----------------|-------------------------------------------------|
| <i>pGcmTag</i> | : Array of 4x32-bit words to read GCM tage into |
|----------------|-------------------------------------------------|

## Returns

LPC\_OK on success, or an error code (ERRORCODE\_T) on failure

## Note

The GCM tage is an XOR value of the Output Text and GF128(Z) hash value. Value is undefined if AES is not present.

## Function Documentation

### 37.6.16 static uint32\_t aesGetDriverVersion ( void ) [inline], [static]

Returns

Driver version, example 0x00000100 = v1.0

### 37.6.17 static ErrorCode\_t aesIsSupported ( void ) [inline], [static]

Returns

LPC\_OK if enabled, ERR\_SEC\_AES\_NOT\_SUPPORTED if not supported

### 37.6.18 static uint32\_t BOOT\_RemapAddress ( uint32\_t address ) [inline], [static]

The chip has a remapping capability that allows to remap internal flash areas. This feature is part of the firmware update mechanism (OTA).

Parameters

|                |                            |
|----------------|----------------------------|
| <i>address</i> | logical address to convert |
|----------------|----------------------------|

Returns

physical address

### 37.6.19 static uint32\_t boot\_Verify\_eScoreImageList ( IMAGE\_DATA\_T \* list\_head ) [inline], [static]

The image list is already sorted by version number. Compare image version against Min version read from PSECT. If it is greater than or equal to Min version, perform the RSA authentication over the image using the ket found in PFLASH if any. see IMAGE\_VERIFY\_T

Parameters

|                  |                       |
|------------------|-----------------------|
| <i>list_head</i> | sorted list of images |
|------------------|-----------------------|

Returns

selected image start address

### 37.6.20 **static uint32\_t BOOT\_FindImage ( uint32\_t *start\_addr*, uint32\_t *end\_addr*, uint32\_t *signature*, IMAGE\_VERIFY\_T *verify* ) [inline], [static]**

This function is involved in the search of a bootable image. It is called by the boot ROM on Cold boot but can be called by the Selective OTA.

The application granularity parameter is read from the PSECT, this is used as the increment used to hop to next position in case of failure. The function builds up a chained list of image descriptors that it sorts by version number. The intent is that the most recent version is at the head of the list.

Parameters

|                       |                                                                                                                                                             |
|-----------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>start_addr</i>     | address from which to start search                                                                                                                          |
| <i>end_addr</i>       | address from which to start search                                                                                                                          |
| <i>signature</i>      | magic identifier : constant 0x98447902                                                                                                                      |
| <i>IMAGE_VERIFY_T</i> | verification function pointer (see ) This parameter cannot be NULL. The implementer may opt for a version that simply returns the head of the chained list. |

Returns

image address if valid, IMAGE\_INVALID\_ADDR (0xffffffff) otherwise

### 37.6.21 **static uint32\_t BOOT\_GetStartPowerMode ( void ) [inline], [static]**

This is mostly used to determine in which power mode the PMC was before reset, i.e. whether is is a cold or warm reset. This is to be invoked from ResetISR2

Parameters

|             |  |
|-------------|--|
| <i>none</i> |  |
|-------------|--|

Returns

LPMMode

### 37.6.22 **static void BOOT\_SetResumeStackPointer ( uint32\_t *stack\_pointer* ) [inline], [static]**

## Function Documentation

### Parameters

|                      |                                                                                          |
|----------------------|------------------------------------------------------------------------------------------|
| <i>stack_pointer</i> | address to be written in retained RAM bank so that boot ROM restores value on warm start |
|----------------------|------------------------------------------------------------------------------------------|

### Returns

none

### 37.6.23 static void ROM\_GetFlash ( uint32\_t \* *address*, uint32\_t \* *size* ) [inline], [static]

The internal flash start address is necessarily 0. Its size may vary depending on chip options. The size returned is the number of bytes usable for program and data. The maximum possible value is 0x9dc00.

### Parameters

|                |                                               |
|----------------|-----------------------------------------------|
| <i>address</i> | pointer on location to store returned address |
| <i>size</i>    | pointer on location to store returned size    |

### Returns

\*address is 0x00000000UL and \*size is up to 0x9dc00

### 37.6.24 static void ROM\_GetSRAM0 ( uint32\_t \* *address*, uint32\_t \* *size* ) [inline], [static]

### Parameters

|                |                                               |
|----------------|-----------------------------------------------|
| <i>address</i> | pointer on location to store returned address |
| <i>size</i>    | pointer on location to store returned size    |

### Returns

\*address is 0x04000000UL and \*size is 88k (0x16000)

### 37.6.25 static void ROM\_GetSRAM1 ( uint32\_t \* *address*, uint32\_t \* *size* ) [inline], [static]

SRAM1 presence is optional depending on chip variant

## Parameters

|                |                                               |
|----------------|-----------------------------------------------|
| <i>address</i> | pointer on location to store returned address |
| <i>size</i>    | pointer on location to store returned size    |

## Returns

if SRAM1 not present \*address is 0 and \*size is 0, otherwise \*address is 0x04020000UL and \*size is up to 64k (0x10000)

### 37.6.26 static int ISP\_Entry ( ISP\_EXTENSION\_T *isp\_extension* ) [inline], [static]

The ISP mode is requested when GPIO 5 is held down on rest or when no valid image can be found in the in the internal flash.

## Parameters

|                      |                                                                                                                                                                                                                                                                     |
|----------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>isp_extension</i> | function pointer on extension function. ISP_INVALID_EXTENSION (0) : no extension requested is the only implemented choice Note: ISP_Entry reads from vector table [13] in order to find a possible extension function. The boot ROM has a 0 value at that location. |
|----------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

## Returns

status 0: ISP entered successfully, otherwise error was detected (ISP disabled)

### 37.6.27 static void Chip\_LOWPOWER\_SetUpLowPowerModeWakeUpTimer ( LPC\_LOWPOWER\_T \* *p\_lowpower\_cfg* ) [inline], [static]

## Parameters

|                         |                                                                    |
|-------------------------|--------------------------------------------------------------------|
| <i>p_lowpower_cfg</i> : | pointer to a structure that contains all low power mode parameters |
|-------------------------|--------------------------------------------------------------------|

## Returns

Nothing

### 37.6.28 static int Chip\_LOWPOWER\_SetSystemFrequency ( uint32\_t *frequency* ) [inline], [static]

## Function Documentation

### Parameters

|                  |   |
|------------------|---|
| <i>Frequency</i> | : |
|------------------|---|

### Returns

Nothing

**37.6.29** `static int Chip_LOWPOWER_SetMemoryLowPowerMode ( uint32_t  
p_sram_instance, uint32_t p_sram_lp_mode ) [inline], [static]`

### Parameters

|                           |                                                                                                                                           |
|---------------------------|-------------------------------------------------------------------------------------------------------------------------------------------|
| <i>p_sram_instance</i> ,: | SRAM instance number, between 0 and 11.                                                                                                   |
| <i>p_sram_lp_mode</i>     | : Low power mode : LOWPOWER_SRAM_LPMODE_ACTIVE, LOWPOWER_SRAM_LPMODE_SLEEP, LOWPOWER_SRAM_LPMODE_DEEPSLEEP, LOWPOWER_SRAM_LPMODE_SHUTDOWN |

### Returns

Status code

**37.6.30** `static void Chip_LOWPOWER_GetSystemVoltages ( LPC_LO-  
WPOWER_LDOVOLTAGE_T * p_ldo_voltage ) [inline],  
[static]`

### Parameters

|                         |                                                                  |
|-------------------------|------------------------------------------------------------------|
| <i>p_ldo_voltage</i> ,: | pointer to a structure to fill with current voltages on the chip |
|-------------------------|------------------------------------------------------------------|

### Returns

Nothing

**37.6.31** `static void Chip_LOWPOWER_SetSystemVoltages ( LPC_LO-  
WPOWER_LDOVOLTAGE_T * p_ldo_voltage ) [inline],  
[static]`

## Parameters

|                         |                                                                 |
|-------------------------|-----------------------------------------------------------------|
| <i>p_ldo_voltage</i> ,: | pointer to a structure that contains new voltages to be applied |
|-------------------------|-----------------------------------------------------------------|

## Returns

Nothing

### 37.6.32 static void Chip\_LOWPOWER\_SetLowPowerMode ( LPC\_LOWPOWER\_T \* *p\_lowpower\_cfg* ) [inline], [static]

## Parameters

|                          |                                                                    |
|--------------------------|--------------------------------------------------------------------|
| <i>p_lowpower_cfg</i> ,: | pointer to a structure that contains all low power mode parameters |
|--------------------------|--------------------------------------------------------------------|

## Returns

Nothing

### 37.6.33 static void Chip\_LOWPOWER\_ChipSoftwareReset ( void ) [inline], [static]

Power down the flash then perform the full chip reset as POR or Watchdog do, The reset includes JTAG debugger, Digital units and Analog modules. Use the Software reset bit in PMC

## Returns

Nothing

### 37.6.34 static void Chip\_LOWPOWER\_ArmSoftwareReset ( void ) [inline], [static]

Power down the flash then perform the Full chip reset as POR or Watchdog, The reset includes the digital units but excludes the JTAG debugger, and the analog modules. Use the system reset bit in PMC and ARM reset

## Returns

Nothing

## Function Documentation

### 37.6.35 static int MPU\_pSectorGrantAccessRights ( uint32\_t *addr*, size\_t *sz*, MPU\_reg\_settings\_t \* *save\_rule* ) [inline], [static]

Note: The pSector region is 'special' because counter intuitively it requires Write access in order to be able to read from it using the flash controller indirect method. The previously applied policy. pSector region is protected under rule 7 (highest precedence). The previous rule 7 is saved in RAM before changing it.

#### Parameters

|                     |                                     |
|---------------------|-------------------------------------|
| <i>addr</i> ,:      | address of area to grant access to. |
| <i>sz</i> ,:        | size in number of bytes of area.    |
| <i>save_rule</i> ,: | save a copy of previous rule        |

#### Returns

-1 if failure, if succesful return the size of the region.

### 37.6.36 static int MPU\_pSectorWithdrawAccessRights ( MPU\_reg\_settings\_t \* *save\_rule* ) [inline], [static]

The pSector region is 'special' because counter intuitively it requires Write access in order to be able to read from it using the flash controller indirect method.

#### Parameters

|                     |                                                                                                                                                                 |
|---------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>save_rule</i> ,: | pointer on RAM MPU_reg_settings_t structure saved by MPU_pSectorGrantAccessRights used to restore previous settings of region 7 and restrict access to pSector. |
|---------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------|

#### Returns

-1 if failure, if succesful return the size of the region.

### 37.6.37 static void MPU\_GetCurrentSettings ( MPU\_Settings\_t \* *settings* ) [inline], [static]



## Parameters

|                   |                                                          |
|-------------------|----------------------------------------------------------|
| <i>settings,:</i> | pointer of structure to receive the MPU register values. |
|-------------------|----------------------------------------------------------|

## Returns

none

### 37.6.38 static int MPU\_AllocateRegionDesc ( void ) [inline], [static]

Checks if rules have their RASR enable bit set. This for the application to find free riules that were left unused by the ROM code. Implicitly MPU\_ClearRegionSetting releases an allocate rule.

## Parameters

|             |  |
|-------------|--|
| <i>none</i> |  |
|-------------|--|

## Returns

-1 if none free, value between 1..4 if succesful.

### 37.6.39 static uint32\_t pmc\_reset\_get\_cause ( void ) [inline], [static]

## Returns

Reset cause value.

## Return values

|            |                                                                                          |
|------------|------------------------------------------------------------------------------------------|
| <i>0x1</i> | POR - The last chip reset was caused by a Power On Reset.                                |
| <i>0x2</i> | PADRESET - The last chip reset was caused by a Pad Reset.                                |
| <i>0x4</i> | BODRESET - The last chip reset was caused by a Brown Out Detector.                       |
| <i>0x8</i> | SYSTEMRESET - The last chip reset was caused by a System Reset requested by the ARM CPU. |

## Function Documentation

|             |                                                                                                                       |
|-------------|-----------------------------------------------------------------------------------------------------------------------|
| <i>0x10</i> | WDTRESET - The last chip reset was caused by the Watchdog Timer.                                                      |
| <i>0x20</i> | WAKEUPIORESET - The last chip reset was caused by a Wake-up I/O (GPIO or internal NTAG FD INT).                       |
| <i>0x40</i> | WAKEUPPWDNRESET - The last CPU reset was caused by a Wake-up from Power down (many sources possible: timer, IO, ...). |
| <i>0x80</i> | SWRRESET - The last chip reset was caused by a Software.                                                              |

### 37.6.40 static void pmc\_reset\_clear\_cause ( uint32\_t *mask* ) [inline], [static]

#### Parameters

|             |                                                  |
|-------------|--------------------------------------------------|
| <i>mask</i> | The mask of reset cause which you want to clear. |
|-------------|--------------------------------------------------|

#### Returns

none

### 37.6.41 static psector\_write\_status\_t psector\_WriteUpdatePage ( psector\_partition\_id\_t *part\_index*, psector\_page\_t \* *page* ) [inline], [static]

The actual write to the partition will be effective after a reset only. Among other checks, the page must have a correct magic, a correct checksum

#### Parameters

|                     |                                                        |
|---------------------|--------------------------------------------------------|
| <i>part_index,:</i> | PSECTOR_PAGE0_PART or PFLASH_PAGE0_PART.               |
| <i>page,:</i>       | psector_page_t RAM buffer to be written to update page |

#### Returns

status code see @ psector\_write\_status\_t.

### 37.6.42 static void psector\_EraseUpdate ( void ) [inline], [static]

The actual write to the partition will be effective after a reset only.

## Parameters

|                     |                                                        |
|---------------------|--------------------------------------------------------|
| <i>part_index,:</i> | PSECTOR_PAGE0_PART or PFLASH_PAGE0_PART.               |
| <i>page,:</i>       | psector_page_t RAM buffer to be written to update page |

## Returns

status code see @ psector\_write\_status\_t.

**37.6.43 static psector\_page\_state\_t psector\_ReadData ( psector\_partition\_id\_t *part\_index*, int *page\_number*, uint32\_t *offset*, uint32\_t *size*, void \* *data* ) [inline], [static]**

## Parameters

|                      |                                                           |
|----------------------|-----------------------------------------------------------|
| <i>part_index,:</i>  | PSECTOR_PAGE0_PART or PFLASH_PAGE0_PART.                  |
| <i>page_number,:</i> | necessarily 0 since partitions now contain 1 single page. |
| <i>offset,:</i>      | offset of data from which data is to be read              |
| <i>size,:</i>        | number of bytes to be read                                |
| <i>data,:</i>        | pointer on RAM buffer used to copy retrived data.         |

## Returns

status code see @ psector\_page\_state\_t if PAGE\_STATE\_DEGRADED or PAGE\_STATE\_OK, data is available. if PAGE\_STATE\_ERROR or PAGE\_STATE\_BLANK, no data was read

**37.6.44 static uint32\_t psector\_CalculateChecksum ( psector\_page\_t \* *psector\_page* ) [inline], [static]**

It is essential to recalculate the checksum when performing a psector page update, failing to update this field, the write operation would be rejected.

## Parameters

|                       |                                                     |
|-----------------------|-----------------------------------------------------|
| <i>psector_page,:</i> | pointer on page over which computation is required. |
|-----------------------|-----------------------------------------------------|

## Returns

checksum value to be checked or to replace checksum field of psector header

**37.6.45**   `static uint64_t psector_Read_CustomerId ( void ) [inline], [static]`

## Parameters

|             |  |
|-------------|--|
| <i>none</i> |  |
|-------------|--|

## Returns

CustomerId on 64 bit word

**37.6.46** `static int psector_Read_RomPatchInfo ( uint32_t * patch_region_sz,  
uint32_t * patch_region_addr, uint32_t * patch_checksum, uint32_t *  
patch_checksum_valid ) [inline], [static]`

## Parameters

|                                 |                                                                       |
|---------------------------------|-----------------------------------------------------------------------|
| <i>patch_region_-sz,:</i>       | pointer on unsigned long to return ROM patch size                     |
| <i>patch_region_-addr,:</i>     | pointer on unsigned long to return ROM patch address                  |
| <i>patch_-checksum,:</i>        | pointer on unsigned long to return ROM patch checksum value           |
| <i>patch_-checksum_-valid,:</i> | pointer on unsigned long to return ROM patch checksum validity (0..1) |

## Returns

-1 if error is found (any of the input parameters is NULL) or PFLASH is unreadable.

**37.6.47** `static uint16_t psector_Read_ImgAuthLevel ( void ) [inline],  
[static]`

## Parameters

|              |  |
|--------------|--|
| <i>none.</i> |  |
|--------------|--|

## Returns

AUH\_NONE if PFLASH unreadable, or the image\_authentication\_level field value if readable.

```
37.6.48 static uint32_t psector_Read_AppSearchGranularity(void) [inline],
[static]
```

## Parameters

|              |  |
|--------------|--|
| <i>none.</i> |  |
|--------------|--|

## Returns

0 if PFLASH unreadable, or the app\_search\_granularity field value if not 0 or 4096 if 0.

### 37.6.49 static uint32\_t psector\_Read\_QspiAppSearchGranularity ( void ) [inline], [static]

## Parameters

|              |  |
|--------------|--|
| <i>none.</i> |  |
|--------------|--|

## Returns

0 if PFLASH unreadable, or the qspi\_app\_search\_granularity field value.

### 37.6.50 static uint64\_t psector\_Read\_DeviceId ( void ) [inline], [static]

## Parameters

|              |  |
|--------------|--|
| <i>none.</i> |  |
|--------------|--|

## Returns

0 if PFLASH unreadable, or the device\_id field value.

### 37.6.51 static int psector\_Read\_UnlockKey ( int \* valid, uint8\_t key[256], bool raw ) [inline], [static]

## Parameters

|                |                                                                                          |
|----------------|------------------------------------------------------------------------------------------|
| <i>valid,:</i> | pointer on int to store validity of key (unlock_key_valid field)                         |
| <i>key,:</i>   | pointer on 256 byte storage to receive the key read from PFLASH                          |
| <i>raw,:</i>   | raw if raw is not requested (0), the key is deciphered using the internal AES fused key. |

## Function Documentation

### Returns

-1 if read error occurred, 0 otherwise

**37.6.52 static int psector\_Read\_ISP\_protocol\_key ( uint8\_t key[16] ) [inline],  
[static]**

### Parameters

|              |                                                                 |
|--------------|-----------------------------------------------------------------|
| <i>key,:</i> | pointer on 16 byte storage to receive the key read from PFLASH. |
|--------------|-----------------------------------------------------------------|

### Returns

-1 if read error occurred, 0 otherwise

**37.6.53 static uint64\_t psector\_Readleee802\_15\_4\_MacId1 ( void ) [inline],  
[static]**

### Parameters

|              |  |
|--------------|--|
| <i>none.</i> |  |
|--------------|--|

### Returns

64 bit word 0 if field unreadable, otherwise MAC address contained in ieee\_mac\_id1 field.

**37.6.54 static uint64\_t psector\_Readleee802\_15\_4\_MacId2 ( void ) [inline],  
[static]**

### Parameters

|              |  |
|--------------|--|
| <i>none.</i> |  |
|--------------|--|

### Returns

64 bit word 0 if field unreadable, otherwise MAC address contained in ieee\_mac\_id2 field.

**37.6.55 static uint64\_t psector\_Read\_MinDeviceId ( void ) [inline],  
[static]**



## Parameters

|              |  |
|--------------|--|
| <i>none.</i> |  |
|--------------|--|

## Returns

0 if PFLASH unreadable, otherwise min\_device\_id field content.

**37.6.56** `static uint64_t psector_Read_MaxDeviceId ( void ) [inline],  
[static]`

## Parameters

|              |  |
|--------------|--|
| <i>none.</i> |  |
|--------------|--|

## Returns

0 if PFLASH unreadable, otherwise max\_device\_id field content.

**37.6.57** `static uint32_t psector_Read_MinVersion ( void ) [inline], [static]`

## Parameters

|              |  |
|--------------|--|
| <i>none.</i> |  |
|--------------|--|

## Returns

if PAGE0 unreadable, otherwise MinVersion field content.

**37.6.58** `static psector_write_status_t psector_SetEscoreImageData ( uint32_t  
image_addr, uint32_t min_version ) [inline], [static]`

## Parameters

---

## Function Documentation

|                      |                                                   |
|----------------------|---------------------------------------------------|
| <i>image_addr,:</i>  | 32 bit value to be written to SelectImageAddress. |
| <i>min_version,:</i> | 32 bit value to be written to MinVersion.         |

Returns

psector\_write\_status\_t status of operation see

**37.6.59 static psector\_page\_state\_t psector\_ReadEscoreImageData ( uint32\_t \* *image\_addr*, uint32\_t \* *min\_version* ) [inline], [static]**

Parameters

|                       |                                                             |
|-----------------------|-------------------------------------------------------------|
| <i>image_addr,:</i>   | pointer on 32 bit word to receive SelectImageAddress value. |
| <i>min_versionm,:</i> | pointer on 32 bit word to receive SelectImageAddress value. |

Returns

-1 if read error occurred, 0 otherwise

**37.6.60 static int psector\_Read\_ImagePubKey ( int \* *valid*, uint8\_t *key*[256], bool *raw* ) [inline], [static]**

Parameters

|                |                                                                                          |
|----------------|------------------------------------------------------------------------------------------|
| <i>valid,:</i> | pointer on int to store validity of key (img_pk_valid field)                             |
| <i>key,:</i>   | pointer on 256 byte storage to receive the key read from PAGE0                           |
| <i>raw,:</i>   | raw if raw is not requested (0), the key is deciphered using the internal AES fused key. |

Returns

-1 if read error occurred, 0 otherwise

**37.6.61 static uint32\_t secure\_VerifySignature ( uint8\_t \* *hash*, const uint8\_t \* *signature*, const uint32\_t \* *key* ) [inline], [static]**

Verify a signature by encrypting it using the provided public key and validating the output matches the provided hash resulting from the SHA-256

## Parameters

|                    |                                             |
|--------------------|---------------------------------------------|
| <i>hash,:</i>      | pointer on computed SHA-256 hash            |
| <i>signature,:</i> | pointer on RSA-2048 signature to be checked |
| <i>key,:</i>       | pointer on public key                       |

## Returns

1: if correct, 0: otherwise.

**37.6.62** `static uint32_t secure_VerifyBlock ( uint8_t * start, uint32_t length, const uint32_t * key, const uint8_t * signature ) [inline], [static]`

Verify a data block with appended signature. Computes the SHA-256 hash. calls see

## Parameters

|                    |                                             |
|--------------------|---------------------------------------------|
| <i>start,:</i>     | pointer on start of data block              |
| <i>length,:</i>    | length of data block                        |
| <i>key,:</i>       | pointer on public key                       |
| <i>signature,:</i> | pointer on RSA-2048 signature to be checked |

## Returns

1: if correct, 0: otherwise.

**37.6.63** `static uint32_t secure_VerifyCertificate ( const IMAGE_CERT_T * certificate, const uint32_t * key, const uint8_t * cert_signature ) [inline], [static]`

Verify certificate is valid for this device and is authentic. Certificate is checked for validity against customer and device ID stored in PFLASH.

## Parameters

---

## Variable Documentation

|                          |                                             |
|--------------------------|---------------------------------------------|
| <i>certificate,:</i>     | pointer on computed SHA-256 hash            |
| <i>key,:</i>             | pointer on public key                       |
| <i>cert_-signature,:</i> | pointer on RSA-2048 signature to be checked |

### Returns

1: if certificate is valid, 0: otherwise.

### 37.6.64 static uint32\_t secure\_VerifyImage ( uint32\_t *image\_addr*, const IMAGE\_CERT\_T \* *root\_cert* ) [inline], [static]

The function retrieves the certificate pointed by the boot block certificate offset field. If present it has to be verified using the root certificate.

### Parameters

|                     |                                                                                                                         |
|---------------------|-------------------------------------------------------------------------------------------------------------------------|
| <i>image_addr,:</i> | pointer on start of image to be checked.                                                                                |
| <i>root_cert,:</i>  | pointer on root certificate (that contains a public key). If the root certificate is present the key is gotten from it. |

### Returns

1: if certificate is valid, 0: otherwise.

## 37.7 Variable Documentation

### 37.7.1 uint8\_t ISP\_MEM\_INFO\_T::access

- bit 0: Read access
- bit 1: Write access
- bit 2: Erase right
- bit 3: Erase all right
- bit 4: blank check right A value of 0 denotes that access is closed

### 37.7.2 uint8\_t ISP\_STATE\_T::mode

- 0x01: Default ISP mode
- 0x7f: unlock mode
- 0x80 or higher: treated by extension function if any

**37.7.3 uint32\_t psector\_page\_data\_t::MinVersion**

offset 0x30

**37.7.4 uint32\_t { ... } ::MinVersion**

offset 0x30

**37.7.5 uint32\_t psector\_page\_data\_t::rom\_patch\_region\_addr**

A value outside of the address range used to store the ROM patch binary shall be deemed invalid

**37.7.6 uint32\_t { ... } ::rom\_patch\_region\_addr**

A value outside of the address range used to store the ROM patch binary shall be deemed invalid

**37.7.7 uint16\_t psector\_page\_data\_t::application\_flash\_sz**

0 is interpreted as maximum (640). This is intended to provide an alternative way of restricting the flash size on a device, and to greater granularity, than the eFuse bit. The actual level of granularity that can be obtained is dependent upon the MPU region configuration

**37.7.8 uint16\_t { ... } ::application\_flash\_sz**

0 is interpreted as maximum (640). This is intended to provide an alternative way of restricting the flash size on a device, and to greater granularity, than the eFuse bit. The actual level of granularity that can be obtained is dependent upon the MPU region configuration

**37.7.9 uint16\_t psector\_page\_data\_t::ram1\_bank\_sz**

This is intended to provide an alternative way of restricting the RAM size on a device, and to greater granularity, than the eFuse bit. The actual level of granularity that can be obtained is dependent upon the MPU region configuration

## Variable Documentation

### 37.7.10 uint16\_t { ... } ::ram1\_bank\_sz

This is intended to provide an alternative way of restricting the RAM size on a device, and to greater granularity, than the eFuse bit. The actual level of granularity that can be obtained is dependent upon the MPU region configuration

### 37.7.11 uint32\_t psector\_page\_data\_t::app\_search\_granularity

Value of 0 shall be equated to 4096. Other values are to be used directly; configurations that are not using hardware remapping do not require hard restrictions

### 37.7.12 uint32\_t { ... } ::app\_search\_granularity

Value of 0 shall be equated to 4096. Other values are to be used directly; configurations that are not using hardware remapping do not require hard restrictions

### 37.7.13 uint8\_t psector\_page\_data\_t::unlock\_key[256]

Stored encrypted, using the AES key in eFuse

### 37.7.14 uint8\_t { ... } ::unlock\_key[256]

Stored encrypted, using the AES key in eFuse

## 37.8 Sha\_algorithm\_level\_api

### 37.8.1 Overview

#### SHA Functional Operation

- **status\_t SHA\_Init** (SHA\_Type \*base, sha\_ctx\_t \*ctx, sha\_algo\_t algo)  
*Initialize HASH context.*
- **status\_t SHA\_Update** (SHA\_Type \*base, sha\_ctx\_t \*ctx, const uint8\_t \*message, size\_t message-Size)  
*Add data to current HASH.*
- **status\_t SHA\_Finish** (SHA\_Type \*base, sha\_ctx\_t \*ctx, uint8\_t \*output, size\_t \*outputSize)  
*Finalize hashing.*
- void **SHA\_ClkInit** (SHA\_Type \*base)  
*Start SHA clock.*
- void **SHA\_ClkDeinit** (SHA\_Type \*base)  
*Stop SHA clock.*

### 37.8.2 Function Documentation

#### 37.8.2.1 status\_t SHA\_Init ( SHA\_Type \* base, sha\_ctx\_t \* ctx, sha\_algo\_t algo )

This function initializes new hash context.

Parameters

|     |             |                                                                            |
|-----|-------------|----------------------------------------------------------------------------|
|     | <i>base</i> | SHA peripheral base address                                                |
| out | <i>ctx</i>  | Output hash context                                                        |
|     | <i>algo</i> | Underlying algorithm to use for hash computation. Either SHA-1 or SHA-256. |

Returns

Status of initialization

#### 37.8.2.2 status\_t SHA\_Update ( SHA\_Type \* base, sha\_ctx\_t \* ctx, const uint8\_t \* message, size\_t messageSize )

Add data to current HASH. This can be called repeatedly with an arbitrary amount of data to be hashed.

## Sha\_algorithm\_level\_api

### Parameters

|                |                    |                                |
|----------------|--------------------|--------------------------------|
|                | <i>base</i>        | SHA peripheral base address    |
| <i>in, out</i> | <i>ctx</i>         | HASH context                   |
|                | <i>message</i>     | Input message                  |
|                | <i>messageSize</i> | Size of input message in bytes |

### Returns

Status of the hash update operation

### 37.8.2.3 status\_t SHA\_Finish ( SHA\_Type \* *base*, sha\_ctx\_t \* *ctx*, uint8\_t \* *output*, size\_t \* *outputSize* )

Outputs the final hash and erases the context. SHA-1 or SHA-256 padding bits are automatically added by this function.

### Parameters

|                |                   |                                                                                                                          |
|----------------|-------------------|--------------------------------------------------------------------------------------------------------------------------|
|                | <i>base</i>       | SHA peripheral base address                                                                                              |
| <i>in, out</i> | <i>ctx</i>        | HASH context                                                                                                             |
| <i>out</i>     | <i>output</i>     | Output hash data                                                                                                         |
| <i>in, out</i> | <i>outputSize</i> | On input, determines the size of bytes of the output array. On output, tells how many bytes have been written to output. |

### Returns

Status of the hash finish operation

### 37.8.2.4 void SHA\_ClkInit ( SHA\_Type \* *base* )

Start SHA clock

### Parameters

---



|             |                             |
|-------------|-----------------------------|
| <i>base</i> | SHA peripheral base address |
|-------------|-----------------------------|

#### 37.8.2.5 void SHA\_ClkDeinit ( SHA\_Type \* *base* )

Stop SHA clock

Parameters

|             |                             |
|-------------|-----------------------------|
| <i>base</i> | SHA peripheral base address |
|-------------|-----------------------------|



## Chapter 38 GenericList

### 38.1 Overview

#### Data Structures

- struct [list\\_handle\\_t](#)  
*The list structure. [More...](#)*
- struct [list\\_element\\_handle\\_t](#)  
*The list element. [More...](#)*

#### Enumerations

- enum [list\\_status\\_t](#) {  
    [kLIST\\_Ok](#) = kStatus\_Success,  
    [kLIST\\_DuplicateError](#) = MAKE\_STATUS(kStatusGroup\_LIST, 1),  
    [kLIST\\_Full](#) = MAKE\_STATUS(kStatusGroup\_LIST, 2),  
    [kLIST\\_Empty](#) = MAKE\_STATUS(kStatusGroup\_LIST, 3),  
    [kLIST\\_OrphanElement](#) = MAKE\_STATUS(kStatusGroup\_LIST, 4) }

#### Functions

- void [LIST\\_Init](#) (list\_handle\_t list, uint32\_t max)
- list\_handle\_t [LIST\\_GetList](#) (list\_element\_handle\_t element)  
*Gets the list that contains the given element.*
- list\_status\_t [LIST\\_AddHead](#) (list\_handle\_t list, list\_element\_handle\_t element)  
*Links element to the head of the list.*
- list\_status\_t [LIST\\_AddTail](#) (list\_handle\_t list, list\_element\_handle\_t element)  
*Links element to the tail of the list.*
- list\_element\_handle\_t [LIST\\_RemoveHead](#) (list\_handle\_t list)  
*Unlinks element from the head of the list.*
- list\_element\_handle\_t [LIST\\_GetHead](#) (list\_handle\_t list)  
*Gets head element handle.*
- list\_element\_handle\_t [LIST\\_GetNext](#) (list\_element\_handle\_t element)  
*Gets next element handle for given element handle.*
- list\_element\_handle\_t [LIST\\_GetPrev](#) (list\_element\_handle\_t element)  
*Gets previous element handle for given element handle.*
- list\_status\_t [LIST\\_RemoveElement](#) (list\_element\_handle\_t element)  
*Unlinks an element from its list.*
- list\_status\_t [LIST\\_AddPrevElement](#) (list\_element\_handle\_t element, list\_element\_handle\_t new-Element)  
*Links an element in the previous position relative to a given member of a list.*
- uint32\_t [LIST\\_GetSize](#) (list\_handle\_t list)  
*Gets the current size of a list.*

## Enumeration Type Documentation

- uint32\_t [LIST\\_GetAvailableSize](#) (list\_handle\_t list)  
*Gets the number of free places in the list.*

## 38.2 Data Structure Documentation

### 38.2.1 struct list\_label\_t

#### Data Fields

- struct list\_element\_tag \* [head](#)  
*list head*
- struct list\_element\_tag \* [tail](#)  
*list tail*
- uint16\_t [size](#)  
*list size*
- uint16\_t [max](#)  
*list max number of elements*

### 38.2.2 struct list\_element\_t

#### Data Fields

- struct list\_element\_tag \* [next](#)  
*next list element*
- struct list\_element\_tag \* [prev](#)  
*previous list element*
- struct list\_label \* [list](#)  
*pointer to the list*

## 38.3 Enumeration Type Documentation

### 38.3.1 enum list\_status\_t

Include

Public macro definitions

Public type definitions

The list status

Enumerator

***kLIST\_Ok*** Success.  
***kLIST\_DuplicateError*** Duplicate Error.  
***kLIST\_Full*** FULL.  
***kLIST\_Empty*** Empty.  
***kLIST\_OrphanElement*** Orphan Element.

## 38.4 Function Documentation

### 38.4.1 void LIST\_Init ( list\_handle\_t *list*, uint32\_t *max* )

Public prototypes

Initialize the list.

This function initialize the list.

Parameters

|             |                                                        |
|-------------|--------------------------------------------------------|
| <i>list</i> | - List handle to initialize.                           |
| <i>max</i>  | - Maximum number of elements in list. 0 for unlimited. |

### 38.4.2 list\_handle\_t LIST\_GetList ( list\_element\_handle\_t *element* )

Parameters

|                |                          |
|----------------|--------------------------|
| <i>element</i> | - Handle of the element. |
|----------------|--------------------------|

Return values

|             |                                                                        |
|-------------|------------------------------------------------------------------------|
| <i>NULL</i> | if element is orphan, Handle of the list the element is inserted into. |
|-------------|------------------------------------------------------------------------|

### 38.4.3 list\_status\_t LIST\_AddHead ( list\_handle\_t *list*, list\_element\_handle\_t *element* )

Parameters

|                |                          |
|----------------|--------------------------|
| <i>list</i>    | - Handle of the list.    |
| <i>element</i> | - Handle of the element. |

Return values

|                   |                                                        |
|-------------------|--------------------------------------------------------|
| <i>kLIST_Full</i> | if list is full, kLIST_Ok if insertion was successful. |
|-------------------|--------------------------------------------------------|

### 38.4.4 list\_status\_t LIST\_AddTail ( list\_handle\_t *list*, list\_element\_handle\_t *element* )

## Function Documentation

### Parameters

|                |                          |
|----------------|--------------------------|
| <i>list</i>    | - Handle of the list.    |
| <i>element</i> | - Handle of the element. |

### Return values

|                   |                                                        |
|-------------------|--------------------------------------------------------|
| <i>kLIST_Full</i> | if list is full, kLIST_Ok if insertion was successful. |
|-------------------|--------------------------------------------------------|

### 38.4.5 list\_element\_handle\_t LIST\_RemoveHead ( list\_handle\_t *list* )

#### Parameters

|             |                       |
|-------------|-----------------------|
| <i>list</i> | - Handle of the list. |
|-------------|-----------------------|

#### Return values

|             |                                                                                 |
|-------------|---------------------------------------------------------------------------------|
| <i>NULL</i> | if list is empty, handle of removed element(pointer) if removal was successful. |
|-------------|---------------------------------------------------------------------------------|

### 38.4.6 list\_element\_handle\_t LIST\_GetHead ( list\_handle\_t *list* )

#### Parameters

|             |                       |
|-------------|-----------------------|
| <i>list</i> | - Handle of the list. |
|-------------|-----------------------|

#### Return values

|             |                                                                                 |
|-------------|---------------------------------------------------------------------------------|
| <i>NULL</i> | if list is empty, handle of removed element(pointer) if removal was successful. |
|-------------|---------------------------------------------------------------------------------|

### 38.4.7 list\_element\_handle\_t LIST\_GetNext ( list\_element\_handle\_t *element* )

#### Parameters

---

|                |                          |
|----------------|--------------------------|
| <i>element</i> | - Handle of the element. |
|----------------|--------------------------|

Return values

|             |                                                                                 |
|-------------|---------------------------------------------------------------------------------|
| <i>NULL</i> | if list is empty, handle of removed element(pointer) if removal was successful. |
|-------------|---------------------------------------------------------------------------------|

#### 38.4.8 list\_element\_handle\_t LIST\_GetPrev ( list\_element\_handle\_t *element* )

Parameters

|                |                          |
|----------------|--------------------------|
| <i>element</i> | - Handle of the element. |
|----------------|--------------------------|

Return values

|             |                                                                                 |
|-------------|---------------------------------------------------------------------------------|
| <i>NULL</i> | if list is empty, handle of removed element(pointer) if removal was successful. |
|-------------|---------------------------------------------------------------------------------|

#### 38.4.9 list\_status\_t LIST\_RemoveElement ( list\_element\_handle\_t *element* )

Parameters

|                |                          |
|----------------|--------------------------|
| <i>element</i> | - Handle of the element. |
|----------------|--------------------------|

Return values

|                            |                                     |
|----------------------------|-------------------------------------|
| <i>kLIST_OrphanElement</i> | if element is not part of any list. |
| <i>kLIST_Ok</i>            | if removal was successful.          |

#### 38.4.10 list\_status\_t LIST\_AddPrevElement ( list\_element\_handle\_t *element*, list\_element\_handle\_t *newElement* )

Parameters

## Function Documentation

|                   |                                                  |
|-------------------|--------------------------------------------------|
| <i>element</i>    | - Handle of the element.                         |
| <i>newElement</i> | - New element to insert before the given member. |

Return values

|                            |                                     |
|----------------------------|-------------------------------------|
| <i>kLIST_OrphanElement</i> | if element is not part of any list. |
| <i>kLIST_Ok</i>            | if removal was successful.          |

### 38.4.11 uint32\_t LIST\_GetSize ( list\_handle\_t *list* )

Parameters

|             |                       |
|-------------|-----------------------|
| <i>list</i> | - Handle of the list. |
|-------------|-----------------------|

Return values

|                |                   |
|----------------|-------------------|
| <i>Current</i> | size of the list. |
|----------------|-------------------|

### 38.4.12 uint32\_t LIST\_GetAvailableSize ( list\_handle\_t *list* )

Parameters

|             |                       |
|-------------|-----------------------|
| <i>list</i> | - Handle of the list. |
|-------------|-----------------------|

Return values

|                  |                   |
|------------------|-------------------|
| <i>Available</i> | size of the list. |
|------------------|-------------------|



## Chapter 39

### UART\_Adapter

#### 39.1 Overview

##### Data Structures

- struct [hal\\_uart\\_config\\_t](#)  
*UART configuration structure. [More...](#)*
- struct [hal\\_uart\\_transfer\\_t](#)  
*UART transfer structure. [More...](#)*

##### Macros

- #define [UART\\_ADAPTER\\_NON\\_BLOCKING\\_MODE](#) (0U)  
*Enable or disable UART adapter non-blocking mode (1 - enable, 0 - disable)*
- #define [HAL\\_UART\\_TRANSFER\\_MODE](#) (0U)  
*Whether enable transactional function of the UART.*

##### Typedefs

- typedef void(\* [hal\\_uart\\_transfer\\_callback\\_t](#) )(hal\_uart\_handle\_t handle, [hal\\_uart\\_status\\_t](#) status, void \*callbackParam)  
*UART transfer callback function.*

##### Enumerations

- enum [hal\\_uart\\_status\\_t](#) {  
    [kStatus\\_HAL\\_UartSuccess](#) = kStatus\_Success,  
    [kStatus\\_HAL\\_UartTxBusy](#) = MAKE\_STATUS(kStatusGroup\_HAL\_UART, 1),  
    [kStatus\\_HAL\\_UartRxBusy](#) = MAKE\_STATUS(kStatusGroup\_HAL\_UART, 2),  
    [kStatus\\_HAL\\_UartTxIdle](#) = MAKE\_STATUS(kStatusGroup\_HAL\_UART, 3),  
    [kStatus\\_HAL\\_UartRxIdle](#) = MAKE\_STATUS(kStatusGroup\_HAL\_UART, 4),  
    [kStatus\\_HAL\\_UartBaudrateNotSupport](#),  
    [kStatus\\_HAL\\_UartProtocolError](#),  
    [kStatus\\_HAL\\_UartError](#) = MAKE\_STATUS(kStatusGroup\_HAL\_UART, 7) }  
*UART status.*
- enum [hal\\_uart\\_parity\\_mode\\_t](#) {  
    [kHAL\\_UartParityDisabled](#) = 0x0U,  
    [kHAL\\_UartParityEven](#) = 0x1U,  
    [kHAL\\_UartParityOdd](#) = 0x2U }  
*UART parity mode.*
- enum [hal\\_uart\\_stop\\_bit\\_count\\_t](#) {  
    [kHAL\\_UartOneStopBit](#) = 0U,  
    [kHAL\\_UartTwoStopBit](#) = 1U }  
*UART stop bit count.*

### Initialization and deinitialization

- [hal\\_uart\\_status\\_t HAL\\_UartInit](#) (hal\_uart\_handle\_t handle, [hal\\_uart\\_config\\_t](#) \*config)  
*Initializes a UART instance with the UART handle and the user configuration structure.*
- [hal\\_uart\\_status\\_t HAL\\_UartDeinit](#) (hal\_uart\_handle\_t handle)  
*Deinitializes a UART instance.*

### Blocking bus Operations

- [hal\\_uart\\_status\\_t HAL\\_UartReceiveBlocking](#) (hal\_uart\_handle\_t handle, uint8\_t \*data, size\_t length)  
*Reads RX data register using a blocking method.*
- [hal\\_uart\\_status\\_t HAL\\_UartSendBlocking](#) (hal\_uart\_handle\_t handle, const uint8\_t \*data, size\_t length)  
*Writes to the TX register using a blocking method.*

## 39.2 Data Structure Documentation

### 39.2.1 struct hal\_uart\_config\_t

#### Data Fields

- uint32\_t [srcClock\\_Hz](#)  
*Source clock.*
- uint32\_t [baudRate\\_Bps](#)  
*Baud rate.*
- [hal\\_uart\\_parity\\_mode\\_t](#) [parityMode](#)  
*Parity mode, disabled (default), even, odd.*
- [hal\\_uart\\_stop\\_bit\\_count\\_t](#) [stopBitCount](#)  
*Number of stop bits, 1 stop bit (default) or 2 stop bits.*
- uint8\_t [enableRx](#)  
*Enable RX.*
- uint8\_t [enableTx](#)  
*Enable TX.*
- uint8\_t [instance](#)  
*Instance (0 - UART0, 1 - UART1, ...), detail information please refer to the SOC corresponding RM.*

#### 39.2.1.0.0.2 Field Documentation

##### 39.2.1.0.0.2.1 uint8\_t hal\_uart\_config\_t::instance

Invalid instance value will cause initialization failure.

### 39.2.2 struct hal\_uart\_transfer\_t

#### Data Fields

- uint8\_t \* [data](#)

- *The buffer of data to be transfer.*  
size\_t [dataSize](#)  
*The byte count to be transfer.*

### 39.2.2.0.0.3 Field Documentation

39.2.2.0.0.3.1 uint8\_t\* hal\_uart\_transfer\_t::data

39.2.2.0.0.3.2 size\_t hal\_uart\_transfer\_t::dataSize

## 39.3 Macro Definition Documentation

### 39.3.1 #define HAL\_UART\_TRANSFER\_MODE (0U)

(0 - disable, 1 - enable)

## 39.4 Typedef Documentation

39.4.1 typedef void(\* hal\_uart\_transfer\_callback\_t)(hal\_uart\_handle\_t handle, hal\_uart\_status\_t status, void \*callbackParam)

## 39.5 Enumeration Type Documentation

### 39.5.1 enum hal\_uart\_status\_t

Enumerator

**kStatus\_HAL\_UartSuccess** Successfully.  
**kStatus\_HAL\_UartTxBusy** TX busy.  
**kStatus\_HAL\_UartRxBusy** RX busy.  
**kStatus\_HAL\_UartTxIdle** HAL UART transmitter is idle.  
**kStatus\_HAL\_UartRxIdle** HAL UART receiver is idle.  
**kStatus\_HAL\_UartBaudrateNotSupport** Baudrate is not support in current clock source.  
**kStatus\_HAL\_UartProtocolError** Error occurs for Noise, Framing, Parity, etc. For transactional transfer, The up layer needs to abort the transfer and then starts again  
**kStatus\_HAL\_UartError** Error occurs on HAL UART.

### 39.5.2 enum hal\_uart\_parity\_mode\_t

Enumerator

**kHAL\_UartParityDisabled** Parity disabled.  
**kHAL\_UartParityEven** Parity even enabled.  
**kHAL\_UartParityOdd** Parity odd enabled.

## Function Documentation

### 39.5.3 enum hal\_uart\_stop\_bit\_count\_t

Enumerator

***kHAL\_UartOneStopBit*** One stop bit.  
***kHAL\_UartTwoStopBit*** Two stop bits.

## 39.6 Function Documentation

### 39.6.1 hal\_uart\_status\_t HAL\_UartInit ( hal\_uart\_handle\_t *handle*, hal\_uart\_config\_t \* *config* )

This function configures the UART module with user-defined settings. The user can configure the configuration structure. The parameter handle is a pointer to point to a memory space of size #HAL\_UART\_HANDLE\_SIZE allocated by the caller. Example below shows how to use this API to configure the UART.

```
* uint8_t g_UartHandleBuffer[HAL_UART_HANDLE_SIZE];
* hal_uart_handle_t g_UartHandle = &g_UartHandleBuffer[0];
* hal_uart_config_t config;
* config.srcClock_Hz = 48000000;
* config.baudRate_Bps = 115200U;
* config.parityMode = kHAL_UartParityDisabled;
* config.stopBitCount = kHAL_UartOneStopBit;
* config.enableRx = 1;
* config.enableTx = 1;
* config.instance = 0;
* HAL_UartInit(g_UartHandle, &config);
*
```

Parameters

|               |                                                                                           |
|---------------|-------------------------------------------------------------------------------------------|
| <i>handle</i> | Pointer to point to a memory space of size #HAL_UART_HANDLE_SIZE allocated by the caller. |
| <i>config</i> | Pointer to user-defined configuration structure.                                          |

Return values

|                                            |                                                  |
|--------------------------------------------|--------------------------------------------------|
| <i>kStatus_HAL_Uart-BaudrateNotSupport</i> | Baudrate is not support in current clock source. |
| <i>kStatus_HAL_Uart-Success</i>            | UART initialization succeed                      |

### 39.6.2 hal\_uart\_status\_t HAL\_UartDeinit ( hal\_uart\_handle\_t *handle* )

This function waits for TX complete, disables TX and RX, and disables the UART clock.

## Parameters

|               |                      |
|---------------|----------------------|
| <i>handle</i> | UART handle pointer. |
|---------------|----------------------|

## Return values

|                                 |                                |
|---------------------------------|--------------------------------|
| <i>kStatus_HAL_Uart-Success</i> | UART de-initialization succeed |
|---------------------------------|--------------------------------|

### 39.6.3 `hal_uart_status_t HAL_UartReceiveBlocking ( hal_uart_handle_t handle, uint8_t * data, size_t length )`

This function polls the RX register, waits for the RX register to be full or for RX FIFO to have data, and reads data from the RX register.

## Note

The function [HAL\\_UartReceiveBlocking](#) and the function `#HAL_UartTransferReceiveNon-Blocking` cannot be used at the same time. And, the function `#HAL_UartTransferAbortReceive` cannot be used to abort the transmission of this function.

## Parameters

|               |                                                         |
|---------------|---------------------------------------------------------|
| <i>handle</i> | UART handle pointer.                                    |
| <i>data</i>   | Start address of the buffer to store the received data. |
| <i>length</i> | Size of the buffer.                                     |

## Return values

|                                     |                                               |
|-------------------------------------|-----------------------------------------------|
| <i>kStatus_HAL_UartError</i>        | An error occurred while receiving data.       |
| <i>kStatus_HAL_UartParity-Error</i> | A parity error occurred while receiving data. |
| <i>kStatus_HAL_Uart-Success</i>     | Successfully received all data.               |

### 39.6.4 `hal_uart_status_t HAL_UartSendBlocking ( hal_uart_handle_t handle, const uint8_t * data, size_t length )`

This function polls the TX register, waits for the TX register to be empty or for the TX FIFO to have room and writes data to the TX buffer.

## Function Documentation

### Note

The function [HAL\\_UartSendBlocking](#) and the function `#HAL_UartTransferSendNonBlocking` cannot be used at the same time. And, the function `#HAL_UartTransferAbortSend` cannot be used to abort the transmission of this function.

### Parameters

|               |                                     |
|---------------|-------------------------------------|
| <i>handle</i> | UART handle pointer.                |
| <i>data</i>   | Start address of the data to write. |
| <i>length</i> | Size of the data to write.          |

### Return values

|                                 |                             |
|---------------------------------|-----------------------------|
| <i>kStatus_HAL_Uart-Success</i> | Successfully sent all data. |
|---------------------------------|-----------------------------|

***How to Reach Us:*****Home Page:**[nxp.com](http://nxp.com)**Web Support:**[nxp.com/support](http://nxp.com/support)

Information in this document is provided solely to enable system and software implementers to use NXP products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits based on the information in this document.

NXP makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does NXP assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. "Typical" parameters that may be provided in NXP data sheets and/or specifications can and do vary in different applications, and actual performance may vary over time. All operating parameters, including "typicals," must be validated for each customer application by customer's technical experts. NXP does not convey any license under its patent rights nor the rights of others. NXP sells products pursuant to standard terms and conditions of sale, which can be found at the following address:  
[nxp.com/SalesTermsandConditions](http://nxp.com/SalesTermsandConditions).

While NXP has implemented advanced security features, all products may be subject to unidentified vulnerabilities. Customers are responsible for the design and operation of their applications and products to reduce the effect of these vulnerabilities on customer's applications and products, and NXP accepts no liability for any vulnerability that is discovered. Customers should implement appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP, the NXP logo, NXP SECURE CONNECTIONS FOR A SMARTER WORLD, Freescale, the Freescale logo, Kinetis, Processor Expert, and Tower are trademarks of NXP B.V. All other product or service names are the property of their respective owners. Arm, Cortex, Keil, Mbed, Mbed Enabled, and Vision are trademarks or registered trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere. The related technology may be protected by any or all of patents, copyrights, designs and trade secrets. All rights reserved. Oracle and Java are registered trademarks of Oracle and/or its affiliates. The Power Architecture and Power.org word marks and the Power and Power.org logos and related marks are trademarks and service marks licensed by Power.org.

© 2018 NXP B.V.